**POLITECNICO DI MILANO**
**Corso di Laurea Magistrale in Ingegneria Informatica**
**Dipartimento di Elettronica e Informazione**

# A comparison of different neural networks for agricultural image segmentation

**AI & R Lab**
**Laboratorio di Intelligenza Artificiale**
**e Robotica del Politecnico di Milano**

**Relatore: Ing. Matteo Matteucci**

**Tesi di Laurea di:**
**Stefano Cereda**
**Matricola 837031**

**Anno Accademico 2015-2016**

*A Mario Locati*

# Sommario

In questa tesi vengo confrontate diverse reti neurali nell'ambito della segmentazione di immagini agricole. L'automazione di alcune operazioni frequenti in agricoltura può infatti portare ad importanti benefici sia dal punto di vista economico che ambientale. Attraverso l'uso di immagini provenienti da dataset rivolti al problema dell'identificazione di malerbe e all'identificazione di frutti maturi, vengono confrontati sia classificatori basati su un insieme di features sia diverse architetture neurali. Scopo della tesi è individuare l'architettura neurale più idonea a questi scopi, sia dal punto di vista dell'accuratezza metrica sia dal punto di vista della qualità visiva delle maschere di segmentazione prodotte. Vengono inoltre confrontate la complessità temporale e la dipendenza a variazioni degli iperparametri delle varie reti. Le architetture confrontate comprendono un approccio di tipo sliding window, una rete puramente convoluzionale e diverse reti ricorrenti. Vengono inoltre sperimentate architetture che combinano i diversi approcci.

# Abstract

In this work we compare several neural networks for agricultural images segmentation. The automation of some agricultural tasks can, in fact, lead to both economical and environmental benefits. By using datasets for both the problem of weed detection and growth status recognition, we compare baseline classifiers based on fixed sets of hand-crafted features with several deep neural architectures. Aim of the thesis is therefore to identify the best suited architecture both from a metrical and a qualitative point of view. Moreover, we compare the temporal complexity and the dependence to hyperparameters variations. The implemented neural networks comprehend a sliding window network, a purely convolutional network and several recurrent networks. Moreover, we try to combine the different approaches.

# Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno aiutato durante lo sviluppo di questo lavoro.

Innanzitutto ringrazio il mio relatore, professor Matteucci, per avermi dato la possibilità di lavorare ad un progetto con importanti risvolti pratici e per avermi fornito il supporto di cui avevo bisogno. Ringrazio inoltre Marco Ciccone e Francesco Visin per avermi guidato nell'utilizzo di cineca e delle reti ricorrenti.

Proseguo ringraziando Sebastian Haug e Grzegorz Cielniak, sia per aver condiviso i loro dataset sia per avermi aiutato nell'utilizzo delle loro tecniche.

Un ringraziamento speciale va agli amici che mi hanno aiutato sia in questo lavoro sia nei cinque anni passati: Alessandro, Andrea, Leonardo, Matteo e Valerio. Avete reso migliori questi anni.

Ringrazio inoltre tutte le persone che mi hanno indirizzato verso questo percorso, in particolare la professoressa Sallitto, la professoressa Salone e il professor Rota Sperti.

Vorrei infine ringraziare le persone a me più care: i miei amici, i miei genitori, Francesca e Beatrice per avermi sempre sostenuto, supportato e sopportato.

# Contents

# Acronyms

In this work we will sometimes use the following acronyms:

**CWFID** Crop/Weed Field Image Dataset used for the weed recognition problem [15]

**SFO** Scale Free Only, indicating the scale invariant subset of features as defined in Table 3.1 on page 33

**RFC** Random Forest Classifier

**SVM** Support Vector Machine

**XGB** Gradient boosted classifier implemented with the XGBoost library

**SWind** Sliding Window network like the one proposed in [36]

**U-Net** U shaped neural architecture like the one proposed in [37]

**U-ReNet** Novel neural network that mixes the U-shaped architecture of U-Net with the ReNet layer

**U-ReNet2** U-ReNet variation with a different number of neurons on each layer

**ReSeg** The recurrent network proposed in [53]

**ReConv** The ReSeg network stacked on top of pre trained convolutional layers

**ReConcat** The ReConv network with an input concatenation

# Chapter 1

# Introduction

*"If God said in plain language, "I'm giving you a choice, forgive or die," a lot of people would go ahead and order their coffin."*

The secret Life of Bees

## 1.1 General overview

Fine-grained image classification and segmentation (i.e., pixel-wise classification) have received considerable attention in the late years: the joint conditions of having large amounts of data and great computational resources lead the research toward complex convolutional neural networks classifiers that broke the records on many image classification competitions ([19], [49], [17]). One of the field that could receive a great benefit from these technologies is agriculture, where the automation of some tasks can lead to reduced costs both environmentally and economically.

In the context of agriculture automation, weed detection plays a relevant role. In order to achieve better crop yields, weeds must be removed so as to leave more nutrients to the interesting plants. This is typically done by spraying agricultural chemicals all over the field, which poses many environmental and economical concerns. In fact, considering that just some parts of the field are covered with weeds, a great amount of herbicides is wasted. Moreover, the soil and the crops get damaged by those chemicals. A solution is therefore to separately classify each plant and spray the herbicide just on the weeds, but, since doing this manually requires costly hand work, the automation of such a task is really interesting.

Another relevant problem is the automation of harvesting, which can also lead to economical advantages. The harvesting process can be conducted in

two ways: in "slaughter harvesting" an entire field is harvested in a single pass, whereas selective harvesting methods choose only mature crops. This second method has clearly some advantages, but requires a system able not only to detect crops, as done in the weed detection problem, but also to estimate their growth stage so to separate and harvest only the mature ones.

The simplest way to acquire data that can be processed to detect the interesting plants is to take some images while going trough the field and process those images in real time. The aim of the current thesis is therefore to compare several methods for agricultural image segmentation both on the problem of weed detection and selective harvesting.

The system we propose is able to perform segmentation on images taken in a real field, under artificial lighting, and containing red and near-infrared channels or red, green and blue channels. We also investigate whether the addition of a depth channel can easily lead to substantial improvements in terms of classification accuracy. The system is able to distinguish between soil, weeds and crops when performing weed detection and to detect crops when performing selective harvesting. The separation between different kinds of weeds and the estimation of the growth stage of the crops is left to future development.

We have compared several classifiers: a random forest classifier; a support vector machine; a gradient boosted classifier and several deep neural networks. The aim is to select the best suited neural architecture.

For the weed detection problem, the system has been trained and tested on CWFID [15], a recently released public dataset consisting of 60 images collected in an organic carrot field under artificial lighting. Given the small dimension of the dataset, we heavily exploited data augmentation techniques for the neural network training, thus also testing the ability of the various architectures to extract relevant informations from a small number of images. For the selective harvesting problem, the system has been trained and tested on the Broccoli dataset [22], a 3D dataset from which we have extracted 300 2D RGB images. In order to increase the quality of the data, we have created a 2D specific ground truth by manually annotating each image. We have also tested whether the addition of a fourth channel representing the distance from the camera leads to better classification accuracies.

The system performance has been measured according to the metrics proposed in [15], we have also evaluated how each classifier is sensible to hyperparameter variations and tried to visualize the features extracted by the deep networks.

Future development should be focused toward the ability of distinguish different kind of weeds and to recognize the growth status of a crop. During

our experiments we also noticed that a different metric could help to better highlight the differences between the classifiers. Moreover, our experiments suggest that a proper mix of the presented architectures would maintain the main benefits of each network.

## 1.2 Brief description of the work

The detection of weed can be divided into two different problems. The first one is to detect weeds between crop rows or between widely spaced crop plants. The second one is to recognize weeds mixed with plants at random positions, which is a more complex task as we do not have any a priori information about weed position and can exploit just the plant's image in order to classify it. This work is focused on the second problem.

A peculiarity of the weed recognition field is the difficulty of collecting data. In fact, the image acquisition process is difficult, as it requires complex hardware, access to farm and must be synchronized to the crop growth stage (once a year for many cultures). Moreover, experts are needed to define a suitable ground truth. In order to solve those problems, a public dataset has been recently released by Haug and Ostermann [15], providing data that can be used to develop classifiers and to compare their performance.

The dataset consists of 60 images taken in an organic carrot field with the presence of intra-row and close-to-crop weeds. The images are composed by a red and a near-infrared channel and each image comes with a vegetation mask and a crop/weed pixel-based annotation. Along with the dataset the researchers also provided some initial results and proposed some metrics in order to produce comparable results.

Similarly to the automation of weed detection, automatic selective harvesting can lead to economical benefits. For this task we have used the Broccoli dataset [22]: a dataset of 3D images taken in several broccoli fields. The authors also provided 2D images of one field, which we have used for our work: we have selected 300 images and manually annotated each one of them. The task of distinguish a broccoli head from the surrounding leaves is pretty easy, but the idea is to be able to separate the mature heads. This requires to be able to give an accurate estimate of the head dimension, which implies being able to recognize an head even when it is occluded by leaves. Therefore, when annotating the images, we have tried to annotate the full heads, even in the areas where they are covered by leaves and therefore not visible.

The system we implemented exploits a pipeline similar to the one proposed by Haug et al. [16], where a similar problem was addressed. The first step

of the cited pipeline is the removal of the soil, which produces a mask for the vegetation. A grid is then applied on the masked image, and for each keypoint located over a biomass pixel a tile is extracted. The tile represents the neighborhood of the central point and is thus used to extract several features, which are given to a random forest that discriminates between the available classes. The resulting probability is assigned to the central pixel and, once all the grid points have received their probability, they are used to compute all the remaining pixels by performing a spatial smoothing and then using a nearest neighbor interpolator.

Our system also contains different classifiers and a novel smoothing process. Moreover, our system can classify an image using several "deep" neural networks, meaning that the original image is directly segmented at pixel level. The resulting annotation can be easily mixed with the result of a feature based classifier.

For what concerns the features computed on the tiles, we have considered the work of Hall et al. [13], where different features for leaf classification are compared. The idea is to understand which set of features is more robust to condition variations such as translation, scaling, rotation, shading and occlusion. To do that, random forests have been trained using different sets of features. The researchers showed that the best solution is to use the last layer of a convolutional network and a scale-robust subset of the features adopted in [16].

Potena, Nardi, and Pretto [36] pushed further the deep neural network approach, proposing a system composed by two neural networks. The vegetation mask is firstly computed using the same approach of [16], but using a more conservative threshold. The mask is then cleaned by extracting a tile over each pixel and running a convolutional neural network to clean out false positives. From the resulting mask they extract some connected regions, and in each region some points are randomly sampled. Around each point a tile is extracted and it is given to another convolutional neural network that discriminates between soil, crop and weed. Each region is then classified using majority voting over its randomly sampled points. In our system we have implement a similar network.

Another network that we tested is similar to the one proposed by Ronneberger, Fischer, and Brox [38] while developing a classifier for biomedical image segmentation. This network uses a different approach: the image is run just through a single fully convolutional neural network without having to extract tiles nor to further process the result. This kind of approach is more elegant as it avoids repeating the same calculations over and over again as it would occur in a purely sliding window approach. Compared to the

approach used by Potena, Nardi, and Pretto [36], it is also more precise as every pixel of the image is considered instead of just some randomly sampled regions.

Visin et al. proposed [52] an alternative deep neural network architecture for object recognition. This architecture, called ReNet, can be used as a replacement for the convolution+pooling layers commonly used in deep convolutional neural network. It works by exploiting four recurrent neural networks that sweep horizontally and vertically in both directions across the whole image, thus extracting more long distance informations than a convolutional network, which exploits just local informations. By stacking three ReNet layers on top of each other, the authors propose a state of the art network for image classification. Visin et al. also proposed ReSeg [53], a deep neural network architecture that makes use of multiple ReNet layers and of an upsampling layer to perform image segmentation. The authors tested the architecture on more datasets, finding that this architecture can receive an advantage in the performance when stacked on the top of some pre-trained convolutional layers. We have therefore tested both the architectures and also tried to combine them with the U-Net one.

In this work we propose a system able to segment agricultural images using a pipeline that combines the approaches of the cited works. The system is composed both of deep neural networks and of classifiers working on a fixed set of hand crafted features computed on tiles extracted from a grid. The results of the different subsystems are averaged and spatially smoothed to produce a pixel-wise classification.

We have experimented both with the set of features proposed in [16] and the ones selected in [13]. Those features can be given to different classifiers: a random forest, a gradient boosted forest and a support vector machine. As for the neural networks, we have implemented a network like the one proposed in [36], working on the same tiles used to extract the features using a sliding window approach, and other networks working on the full image. Moreover, we have implemented a slightly modified version of U-Net [38], two variations of the U-Net architecture using ReNet [52] layers instead of convolutions, two version of the ReSeg [53] architecture, one of which exploits the first layers of U-Net, and a last network where the output of the ReSeg network (stacked on top of the convolutional layer) is concatenated to the input image before computing the output so to produce a more accurate result.

The output of the classifiers working on the tiles can be averaged and spatially smoothed to reduce the noise in the predictions. Afterwards a nearest neighbor interpolation is used to obtain pixel-wise predictions. At

this point, the results of the networks that produce a full scale segmented image can be averaged to produce the final segmentation mask.

We have then evaluated the various classifiers, comparing them both from the point of view of metrical results and from a purely qualitative analysis of the produced segmentation masks. Moreover, we have compared the temporal complexity and the dependence to hyperparameters variations.

We are interested in several comparisons. More specifically, we want to better understand the differences between the sliding window approach and the direct segmentation of an entire image, to compare convolutional and recurrent neurons and to see whether the upsampling strategy adopted by U-Net is better that the direct upsampling used in ReSeg. Moreover, for the Broccoli dataset, we investigated whether the addition of a depth channel can easily improve the performance.

We have found that an U-shaped architecture, like U-Net proposed in [37], generally delivers the best performances. However, the ReNet layer [52] is able to better detect the shape of broccoli heads, which is important in order to compute their size and their growth status. Albeit we have already implemented two networks that mix these ideas, future developments should also try to investigate whether an U-shaped network consisting of both recurrent and convolutional neurons could lead to better results.

## 1.3 Structure of the thesis

This work is structured in the following way:

- in Section 2 we give a preliminary introduction to the problem of agricultural image segmentation, analyzing the current state of the art. Moreover, we introduce the neural architectures and methodologies exploited in the current work;

- in Section 3 we give a detailed explanation of the various components present in our system, also describing the adopted metrics and the decisions taken during the implementation;

- in Section 4 we report the practical experiments we have performed to build our system, such as the dataset annotation and the training of the classifiers;

- in Section 5 we compare, from several point of views, the obtained classifiers;

- in Section 6 we summarize the obtained results and propose future extensions of the work;
- in appendix A we report some data that justify the decisions taken during the training process explained in Section 4.

# Chapter 2

# State of the art

*"But that's the wonderful thing about man; he never gets so discouraged or disgusted that he gives up doing it all over again, because he knows very well it is important and worth the doing."*

Fahrenheit 451

In this section we report the current state of the art. More specifically, we are interested in two fields: precision agriculture and deep neural networks.

## 2.1 Precision agriculture

Several technological innovations can help to obtain a sustainable growth of agriculture. In this work we are mainly concerned with the recognition of weeds and the recognition of harvestable plants and both problems requires the ability to recognize and classify the single plants. Therefore we have considered previous works in both the domains.

Automatic weed detection is a problem that is being addressed since 1991, when Thompson, Stafford, and Miller [51] understood the great economic and environmental potential of automated selective spraying of weeds in agricultural fields. A general overview of weed control systems can be found in [44], where the whole weed control pipeline is analyzed: guidance systems, recognition of plant species, removal mechanisms and GPS mapping. In this work we are concerned only with the plant recognition problem. As reported in the cited works, many visual characteristics have been used to identify plant species, and they can be divided into three broad categories: biological morphology, spectral characteristics and visual texture.

### 2.1.1 Biological morphology

For what concerns biological morphology, most of the proposed systems work at leaf geometry level, while plant geometry is exploited less frequently. A detailed overview of such systems can be found in [4]. Generally speaking, those kind of systems achieve really high recognition rates under ideal conditions, but strive to adapt to real situations, where plants are occluded or overlapping. In order to better deal with variability, Søgaard [46] proposed a system based on active shape models. The shape models describe the leaf shape and whole plant structure. The classification process is based both on the amount of deformation required to obtain an optimum fit and on the final level of match between the deformed shape and the plant being identified.

Lee, Slaughter, and Giles [24] used the watershed algorithm in order to separate occluded leaves. The idea is to consider the image as a topographical surface where the proximity of each pixel from the background represents its elevation. Starting from the local minima, the image gets flooded and dams are built where two different lakes merge. At the end, those dams provide separation lines, each region is considered as a single leaf and can be classified. In order to avoid over separation of the leaves, they smoothed the distance image by performing a morphological opening before the flooding step. Berge, Aastveit, and Fykse [2] used shape parameters to detect broad-leaved patches in cereal crop. Success rate was between 84 and 90%.

Knowledge about leaf orientation and height can help to obtain more robust classification. Sanchez and Marchant [40] described the possibility of detecting weeds by a fixed threshold on plant height on stereoscopic images of plants in laboratory conditions. Nielsen et al. [29] studied the detection of weeds among tomato plants by analyzing stereoscopic images acquired in the field by a trinocular camera. The authors recognized the negative effect of ground irregularity on classification. Piron, Der Heijden, and Destain [33] proposed to compute a "corrected plant height" parameter to accurately represent plant height taking into account ground irregularities. The introduction of this parameter pushed the overall accuracy of their system from 66% to 83%.

### 2.1.2 Spectral characteristics

Methods based on spectral characteristics are appealing as they are robust to partial occlusion, in addition they tend to be less computationally intensive. Many studies have used various kinds of indexes, typically ratios of broadband reflectance values in visible and near-infrared lights. Scotford and Miller [41]

gave a list of the most commonly studied indexes. The idea is to exploit the fact that chlorophyll reflects the near-infrared light and absorbs the red one. Therefore, as the biomass density increases, the red vs. near-infrared ratio changes and it can be exploited to separate vegetation from soil. In fact, this is done as a first step in most of the systems based on biological morphology. Similarly, different types of soil and different crops can also have small spectral characteristics differences that can be exploited. However, few studies have achieved good accuracies by exploiting color segmentation alone to distinguish crops from weeds.

Guerrero et al. [12] combined color informations with support vector machines, developing a system able to detect plants even when they are covered with dirt or soil. Feyaerts and Van Gool [9] developed a spectrograph able to discriminate beets from five weed species. Classification accuracy was good (up to 86%), however, it required six narrow spectral bands, which is impractical. Piron et al. [34] proposed a combination of three wide-band filters to detect weeds located within carrot rows and found a classification accuracy of only 72%.

### 2.1.3 Visual texture

A few studies have tried to identify plant species by computing visual textures. Shearer and Holmes [42] developed a classification method based on color co-occurrence matrices in intensity, hue and saturation spaces. Eleven textural features were extracted from each co-occurrence matrix, obtaining an overall classification accuracy of 91%. However, the computation of the features was really expensive, and the authors suggested to use a smaller set of texture features. Meyer et al. [27] also used co-occurrence matrix, extracting four textural features to classify grass and broadleaf categories of plants, with classification accuracies of 93 and 85%, respectively. The time required to classify an image ranged from 20 to 30 seconds. Lottes et al. [25] proposed a classification system based on [16], computing more features. In order to compute statistical features for the texture, they employ Local Binary Patterns (LBP) according to Ojala and Pietikäinen [30].

### 2.1.4 Classification without segmentation

Many of the systems that can be found in the literature work by exploiting spectral characteristics to separate soil from vegetation and then computing morphological and textural features on the vegetation to discriminate the single plant. Being based on morphological characteristics, such an approach works very well in ideal conditions, but needs to be refined when the leaves

are occluded or overlapping. Most of the systems presented so far work by trying to separate overlapping and occluded leaves, for example by using watershed algorithm or with morphological openings and closures of the image. Instead of separating and labeling each plant, Cheng and Matson [5] proposed a different solution: they first detect the corner points (i.e. tips of leaf) using Harris Corner Algorithm. Afterward, they extract some features from tiles built around the corners. The features are evaluated and the corner classified. As a final step, to clean the results, they apply DBSCAN to remove noise.

Haug et al. [16] also showed that it is possible to achieve a good accuracy by computing features of patches representing the neighborhood of sparse keypoints. Once removed the background soil, they generate a sparse grid over the image. Every keypoint of the grid that is above a vegetation pixel is considered as the center of a tile that is used to extract some features to classify the keypoint. The results are then spatially smoothed and interpolated to produce a pixel-wise classification. The same approach has been used by Lottes et al. [25], implementing a similar system with much more features.

Potena, Nardi, and Pretto [36] also proposed a system that does not need to explicitly deal with overlapping leaves. Their system is based on two neural networks. The first one (reported in Figure 2.1) is able to discriminate between soil and vegetation by using a sliding window approach. Using the classified points, the system identifies connected regions in the vegetation and randomly selects some points in each region. Around each point a patch is extracted, which is then given to the second neural network (reported in Figure 2.2) to separate crop from weeds. As a final step, each region is labeled by means of majority voting among its pixels.

### 2.1.5 Dataset

In machine vision, open datasets play an important role as they open challenging questions to a wider community and allow direct comparison of different algorithms to the state of the art. However, for agricultural task there are not many available datasets. Söderkvist's Swedish leaf dataset [45] was one of the first available datasets and contains leaf images of Swedish trees. The Flavia dataset by Wu et al. [54] is a newer popular dataset for leaf classification tasks. Kumar et al. developed a smartphone application for leaf classification called Leafsnap [21] and published their dataset. However, the availability of dataset for real-life agricultural tasks is much more limited.

Haug and Ostermann recently released CWFID [15], a dataset with 60

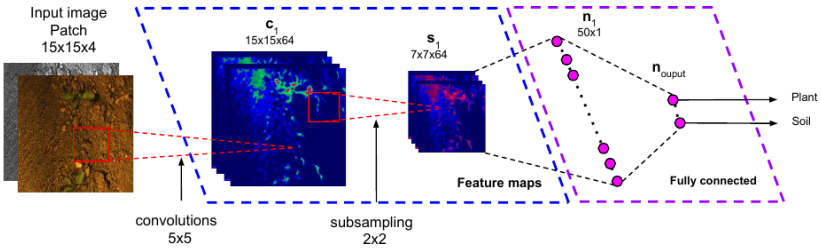



*Figure 2.1: Convolutional network used in [36] to remove the soil. Image taken from [36]*

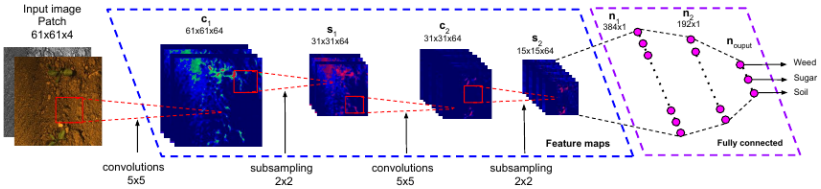



*Figure 2.2: Convolutional network used in [36] to classify the patches. Image taken from [36]*

images taken under artificial lighting in an organic carrot field containing a red and a near-infrared channel. Being the only public dataset available for weed detection, it has been adopted for the current work. Along with the dataset, the authors also provided some initial results and some metrics so to make it possible to compare different classifiers.

Kusumam et al. [22] developed a classifier able to recognize harvestable broccoli heads in real fields and made their dataset publicly available. The dataset consists of several videos captured with a Kinect 2 in various broccoli fields under artificial lighting conditions, therefore providing RGB images with a $1920 \times 1080$ pixels resolution and a $512 \times 242$ depth resolution.

## 2.2 Classifiers working with features

Features extracted from images must be given to a classifier in order to derive a suitable prediction for the image. Several classifiers can be used, in this work we have used random forest classifiers, support vector machines and gradient boosted trees. Since the present work is focused on neural networks, these classifiers should be considered baselines.

### 2.2.1 Random forest

Random forest are an ensemble of decision trees. A decision tree [28] can be represented as a set of if-then rules. Each node in the tree specifies a test of some attributes, and each branch descending from that node corresponds

to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node. At the end, a leaf node will provide the classification of the given instance. In order to build the tree, each split is computed by selecting the attribute that brings to the highest information gain, which can be measured in different ways.

Random forests [10] are a substantial modification of the bagging schema applied to decision trees. Bagging is a technique that works by building many models and then average their results to decrease the final variance. In order to build different models, each model is trained by using a different subset of the available data (bootstrap). Moreover, random forests introduce another technique to further reduce variance: for each tree, before computing each split, a subset of the available features is randomly selected. In the end, the results of the various trees can be combined by means of majority voting. Moreover, by counting the fraction of trees that voted for a class, they can be easily used to provide a probability estimate.

An important feature of random forests is their use of out-of-bag samples: the error estimate for a given observation can be obtained by averaging the results of those trees corresponding to bootstrap samples in which the observation did not appear. An out of bag error estimate is almost identical to that obtained by N-fold cross-validation, but is much faster to compute.

### 2.2.2 Support vector machines

A support vector machine performs binary classification by finding the hyperplane that provides the best separation between the samples of two classes. This is done through the concept of *margin*, which is defined as the smallest distance between the decision boundary and any of the samples [3]. In support vector machines the decision boundary is chosen to be the one for which the margin is maximized.

The decision boundary is linear in the feature space, but, by exploiting the kernel trick [3], it can be made nonlinear in the input space.

Platt scaling can be used to produce a probability estimate for a given sample [35]. The basic idea is to take the outputs of the support vector (i.e., distances from the decision boundary) and use them to fit a logistic regression model. In this way, the support vector outputs are transformed into class probabilities.

### 2.2.3 Gradient boosted trees

Boosting builds a powerful model by ensembling several weaker models. Differently from bagging, it does not work by randomly selecting a different bootstrap for each model but, instead, the various models are built sequentially and each time the samples are selected according to the performance of previous models (i.e., misclassified samples are more important).

Gradient boosting combines this idea with a differentiable loss function, selecting, at each step, the weak classifier that mostly reduce the loss. Moreover, by exploiting the differentiability of the loss function, each classifier can be assigned a weight that maximally reduce the loss [10].

## 2.3 Deep neural networks

In the last years, deep convolutional networks have outperformed state of the art classifiers in many visual recognition tasks. Even if they have been introduced a long time ago [23], their recent fame is mostly due to the availability of both large datasets and massive amounts of computational powers. The impressive result obtained by Krizhevsky, Sutskever, and Hinton [19] have been achieved by training a network with 8 layers with 1 million training images. Since then, even larger and deeper networks have been trained.

Convolutional networks have been firstly applied on image classification tasks, where the goal is to assign a label to an image. However, in many tasks, the desired output is a class assigned to each pixel of the image, which goes under the name of image segmentation. One of the techniques that can be applied to segment images is the so called sliding window approach, as done in [36] when segmenting agricultural images. The idea is to extract a window around each pixel, and use a convolutional network to classify the window. The result is assigned to the central pixel and the window is moved to the next pixel. The advantage of this approach is that it allows to use already existent architectures and, moreover, it increases the amount of available training data, due to the vast overlap of the windows. The disadvantage is that this approach is slow, as the network must be run again for every window.

The sliding window approach also contains a trade-off between the use of context information that is available with larger patches and the localization accuracy that is possible with smaller patches. To overcome this limitation, classifiers that work with features from various layers have been recently proposed [14].

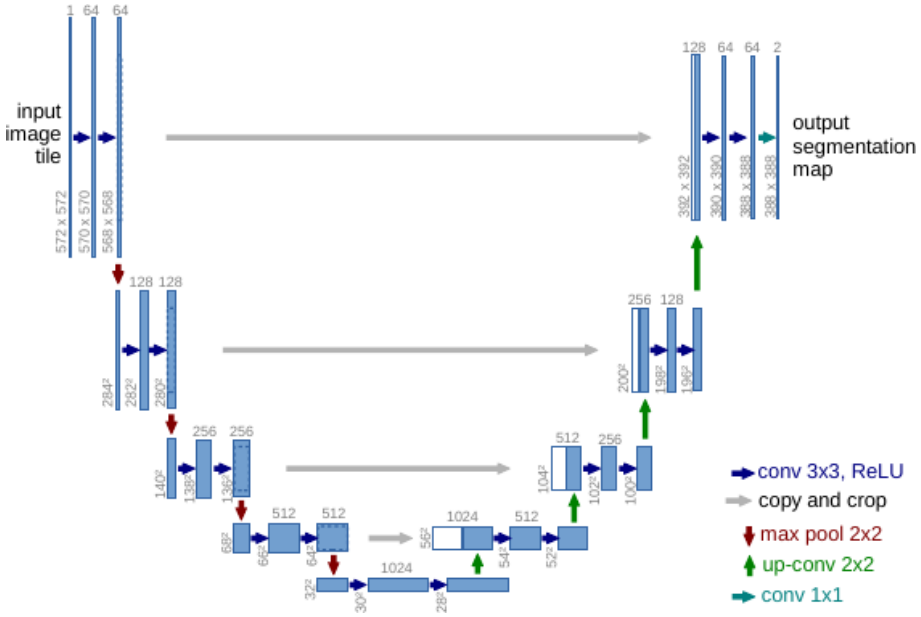



*Figure 2.3: U-net architecture. Image taken from [37]*

Moreover, it is possible to stage an upsampling layer after some convolutional layers, thus directly obtaining a segmentation mask for the whole image. This approach also contains a trade-off. In fact, to extract more complex features we would need to employ more convolutional layer, but this would make harder to accurately upsample the annotation.

### 2.3.1 U-Net

While developing a system for biomedical image segmentation, Ronneberger, Fischer, and Brox [37] proposed a fully convolutional neural network composed by a contracting path able to capture context and a symmetric expanding path that enables precise localization. This approach allows to directly segment an entire image without extracting any window. Heavily relying on data augmentation techniques, they showed how the network can be trained end-to-end with very few images (their training data consisted of 30 images), obtaining a network that outperformed the prior best method based on a sliding window approach, while also being faster. The network architecture can be found in Figure 2.3.

### 2.3.2 ReNet

Visin et al. proposed [52] the ReNet layer, a recurrent neural architecture that can be used as a drop-in replacement for the usually adopted convolu-

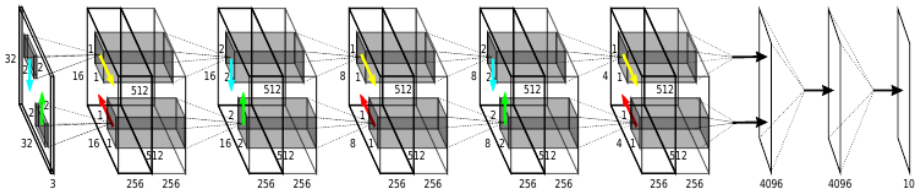



*Figure 2.4: ReNet architecture. Image taken from [52]*

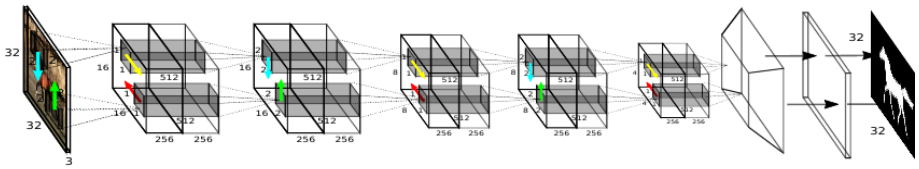



*Figure 2.5: ReSeg architecture. Image taken from [53]*

tion+pooling layer. The layer consists of four recurrent neural networks that sweep over the whole image vertically and horizontally. This architecture allows each feature activation to be a function of the whole image, thus capturing long distance dependences, in contrast with the convolutional approach that works with a local context window. Stacking this kind of layers with some fully connected layer, it is possible to obtain a state of the art network for image classification. The architecture presented in the cited work is reported in Figure 3.10 and is composed of three ReNet layers followed by three fully connected layers.

### 2.3.3 ReSeg

Visin et al. used their work on ReNet to propose ReSeg [53], a recurrent network able to perform image segmentation. The network is composed by 3 successive ReNet layers and an upsampling layer that brings the image back to its original size. The authors also showed how adding some pre-trained convolutional layers at the beginning of the network can help to maximize its performance and obtain a state of the art network for image segmentation. The architecture presented in the cited work is reported in Figure 2.5.

### 2.3.4 Common tricks to avoid overfitting

As said, the recent successes of the deep nural networks are mostly due to the conjunct availability of very large dataset and massive amounts of computational power. However, obtaining an agricultural dataset is a really hard task, and thus the availability of training data for this work is much limited. Being the current work based on small datasets [15] [22], it is crucial

to exploit all the techniques that can help to reduce overfitting during the training phase and successfully train a network with few images.

Szegedy et al. showed [50] that an efficient way to reduce the number of parameters in a network while maintaining the same expressive power is to substitute $n \times n$ convolutions with an $n \times 1$ layer followed by a $1 \times n$ layer. This allows to have an activation function with the same spatial extent in the input image, while having less parameters to train and an additional non-linear stage. With the same principle, small convolutions should be preferred to large ones. For example, a $5 \times 5$ convolution involves 25 parameters and can be substituted with two $3 \times 3$ ones that have just 9 parameters each. Those convolutions can then be factored in $3 \times 1$ and $1 \times 3$ convolutions, with 3 parameters each. From the original 25 parameters we have thus moved to 12, simplifying the learning process.

Patrice Y. Simard showed [32] how elastic distortions can be used to generate additional data. The idea behind is to slightly distort the images to augment the available amount of data, this is done by generating random displacement fields $\Delta_x(x, y)$ and $\Delta_y(x, y)$ which represent, for every pixel $(x, y)$, how it should be moved to its new position (i.e. $(x, y)$ becomes $(x + \Delta_x(x, y), y + \Delta_y(x, y))$. The two fields are then convolved with a gaussian of standard deviation $\sigma$. With higher values of $\sigma$ this is similar to a translation, while smaller values recall a random field. Intermediate values looks like an elastic deformation, we provide some sample images in Figure 3.13c and Figure 3.13d on page 44.

Another commonly used technique that helps to reduce overfitting is dropout [48]. This technique consists in randomly shutting down the output of some neurons during the training phase. This makes it impossible for the network to build specialized paths that just recognize the input data. Another view of the technique is related to bagging: shutting down some paths we always train a different network. Therefore, in the final network we will have several different networks, each one trained on a slightly different training dataset.

### 2.3.5 Training algorithms

In order to train a neural network it is necessary to have an algorithm that allows to update the weights following the loss gradient w.r.t. the parameters[1].

The most basic approach is to simply change the parameters along the

[1] All the reported formulas have been adapted from the slides of Stanford's course CS231n [7].

negative gradient direction. If we define the parameter vector as $\omega$ and the loss gradient w.r.t. $\omega$ as $\partial L$ the update becomes:

$$\omega - = LR \cdot \partial L,$$

where $LR$, the *learning rate*, is an hyperparameter. A usually faster convergence can be reached by considering the momentum update, where basically the negative gradient is used as a force, thus building up a velocity for the parameters vector and avoid getting stuck in local optima:

$$v = \mu \cdot v - LR \cdot \partial L,$$

$$\omega + = v.$$

A slightly different approach is Nesterov momentum, whose core idea is to evaluate the gradient after having considered the momentum, therefore evaluating it closer to our next position:

$$\omega' = \omega + \mu \cdot v,$$

$$v = \mu \cdot v - LR \cdot \partial L(\omega'),$$

$$\omega + = v.$$

All this approaches need a learning rate: an hyperparameter that is multiplied by the negative gradient in order to obtain the actual weight update. An higher learning rate will help to converge faster, but it can overshoot the target and thus not converge at all. On the other hand, a smaller learning rate will make convergence slower and can get stuck in local minima. Intuition suggests that it is good to have a high learning rate during the first optimization steps and progressively reduce it in later steps. The strategy used to decrease the learning rate is an hyperparameter itself and its tuning is an expensive process, therefore much work has been done to propose methodologies that can adaptively tune the learning rate, and even do so per parameter. Those methods usually requires additional hyperparameters, but the idea is that they will perform acceptably on a broader range of hyperparameters, thus making the selection process easier.

Adagrad [8] keeps track of per-parameter sum of squared gradients, using it to modify the learning rate. The parameters that have already received high gradients will have a reduced learning rate, while the parameters that

receive small or infrequent updates will have a boosted learning rate:

$$cache+ = \partial L^2,$$

$$\omega- = \frac{LR \cdot \partial L}{\sqrt{cache} + \epsilon}.$$

where $\epsilon$ is a small value (i.e., somewhere between $10^{-4}$ and $10^{-8}$) to avoid divisions by zero. The drawback is that this method is usually too aggressive and stops the learning too soon.

RMSProp tries to solve this issue by keeping a moving average of squared gradients:

$$cache = decay \cdot cache + (1 - decay) \cdot \partial L^2.$$

$$\omega- = \frac{LR \cdot \partial L}{\sqrt{cache} + \epsilon}.$$

Adam [18] is another method that looks like RMSProp combined with momentum:

$$m = \beta_1 \cdot m + (1 - \beta_1) \cdot \partial L,$$

$$v = \beta_2 \cdot v + (1 - \beta_2) \cdot \partial L^2,$$

$$\omega- = \frac{LR \cdot m}{\sqrt{v} + \epsilon}.$$

The actual formula is slightly different as in the first epochs the vectors $m$ and $v$ are both initialized to zero and thus require a correction mechanism. Recommended values in the paper are $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. In practice Adam is currently recommended as the default algorithm to use, and often works slightly better than RMSProp.

### 2.3.6 Activation functions

An artificial neuron works by performing the weighted sum of its inputs (plus a bias) and by transforming the result with an activation function, which must be differentiable. Moreover, it is common to use a non-linear activation function, since, otherwise, the resulting network would be equivalent to a linear combination of the inputs[2].

The *sigmoid* function has been widely used historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

[2]As for the learning algorithms, we refer to the CS231n course material [7].

- It saturates killing the gradients. When the neuron's activation goes in either one of the tails, the local gradient becomes nearly zero. Therefore, when back propagating the gradients, a saturated neuron effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data.

- It is not zero-centered. This is a problem because, if a neuron receives data that are always positives, it will force its weights to be all updated either in a positive or in a negative way. This inconvenience is mitigated by the fact that, once these gradients are added up across a batch of data, the final update for the weights can have variable signs.

The *Tanh* function is also a commonly used nonlinearity. As the sigmoid neuron, its activations saturate, but, unlike the sigmoid neuron, its output is zero-centered.

The Rectified Linear Unit ($ReLU$) has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. It has several properties:

- It greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid/tanh functions [19].

- Can be implemented by simply thresholding a matrix of activations at zero.

- Unfortunately, ReLU units can "die". A large gradient flowing trough a ReLU may modify its weights in such a way that the neuron will never activate again on the entire dataset. From that point on, the gradient flowing through the unit will always be zero. With a proper setting of the learning rate this is less frequently an issue.

In practice, ReLU is currently the recommended nonlinearity.

### 2.3.7 Weight initialization

Before starting to train the network, its parameters have to be initialized. The first idea that one could have is to initialize all the weights to zero, expecting that, at the end of training, the number of positive parameters will be approximately equal to the number of negative ones. This would be a great mistake, since, if all the parameters are equal, all the neurons compute the same output and, therefore, the same gradients, leading to the same parameter updates.

Therefore, in order to introduce an asymmetry, it is common to initialize the weights to small random values sampled from a Gaussian distribution with unitary variance.

The problem of the previous solution is that the variance of a neuron output grows with the number of its inputs, which has been empirically proven to slow down the rate of convergence [7]. The raw activation value $s$ of a neuron (i.e., before the nonlinear function) is computed as the inner product $s = \sum_i^n \omega_i x_i$ between the weights $\omega$ and input $x$. Assuming zero mean inputs and weights we can thus compute the variance of $s$:

$$Var(s) = Var(\sum_i^n \omega_i x_i)$$

$$= \sum_i^n Var(\omega_i x_i)$$

$$= \sum_i^n [E(\omega_i)]^2 Var(x_i) + [E(x_i)]^2 Var(\omega_i) + Var(x_i)Var(\omega_i)$$

$$= \sum_i^n Var(x_i)Var(\omega_i)$$

$$= (nVar(\omega))Var(x)$$

We can thus see that, if we want $s$ to have the same variance of its input $x$, then we have to make sure, during initialization, that the variance of $\omega$ is $1/n$, which can be obtained by sampling each weight from a Gaussian with standard deviation equal to $\sqrt{1/n}$.

Notice that the assumption of zero mean inputs does not always hold. For example, ReLU units do not have a zero average. He et al. derived [17] an initialization specifically for ReLU neurons, reaching the conclusion that the variance of $\omega$ should be equal to $2/n$.

### 2.3.8 Visualization techniques

One of the main drawbacks of neural networks is that they work like black-boxes. We give them an input image and obtain a classification without knowing which kind of features are being exploited. Therefore, a lot of recent works have focused on finding ways to visualize the features extrapolated by deep neural classifiers [7].

One of the first methods [11] consists in finding, in the dataset, the samples that maximally activate some neurons. The idea is to take a neuron and use it as a feature detector: its output is evaluated in a large set of held-out regions proposals which are then sorted according to the activation of the neuron. By looking at the proposals with the highest score it is possible

to form an idea of the feature extracted by the neuron.

Other methodologies focus in finding images that are classified in a similar way by the network. For instance, [20] considers the activations of the last fully connected layer as an high dimensional vector. For different images, they compute the euclidean distance between the two vectors and thus understand which images are considered similar by the network. Similarly, t-SNE [26] can be used to visualize high dimensional datapoints by projecting them in a two or three dimensional space conserving, locally, pairwise distances.

Neural classifiers can also be understood trough occlusion sensitivity [56]: by systematically occluding with a gray square different parts of the input image and monitoring the output of the classifiers it is possible to produce an heat map that shows how much every part of the image is important for the final classification. In this way, it is possible to understand if the network is effectively recognizing the object or just the surrounding context.

All this methods requires to use existing images. However, there is also a common methodology that consists in the direct visualization of the convolutional weights [20]. Usually this is done just for the first convolutional layer as its weights are easy to understand since they capture easily recognizable shapes like oriented edges and opponent colors. The subsequent layers can also be visualized, but their interpretation becomes more challenging since they are based on the previous layers. Moreover, this visualization works just with convolutional networks and is difficult to adapt to factored convolutions.

There are also some visualization methods ([56], [43], [47]) that exploit the differentiability of the networks. In fact, what is usually done, during training, is to compute the derivative of the loss w.r.t. the network's weight. However, it is also possible to compute the derivative of the loss w.r.t. the input image and thus visualize the gradient of a specific neuron, understanding which part of the image is exciting it. Since the correct backpropagation formula for ReLU leads to noisy images, the cited works propose modified versions that increase the visual quality and thus actually show what is liked by a neuron.

Following the same approach, it is possible [43] to actually compute an image that maximizes a class score. In fact, once we have the gradient of an output neuron w.r.t. the image, it can be used to modify the image so to increase the class score. This, in principle, would allow us to find an image that, according to the network, optimally represents its abstract idea of the class. However, the results tend to be a bit noisy since high frequency informations tend to excite more the neurons and thus are boosted in this backpropagation. The most basic solution consists in adding an $l_2$ regularization term so to prefer sparse input images. Another solution [55]

consists in blurring the image after each update (to increase the importance of low frequency informations) and put to zero every pixel with a small norm (to increase sparsity).

In our work, we have tried to adapt the last two methodologies to our segmentation problem.

# Chapter 3

# Proposed system

> *"Andrew: But is there a line? You know, maybe you go too far, and you discourage the next Charlie Parker from ever becoming Charlie Parker?*
> *Terence Fletcher: No, man, no. Because the next Charlie Parker would never be discouraged."*
>
> Whiplash

In this section we describe the various parts of the proposed system.

## 3.1 Datasets description

We have trained and tested our system on two datasets: CWFID [15] and the Broccoli dataset [22].

### 3.1.1 CWFID

The CWFID dataset [15] is composed of 60 $1296 \times 966$ pixels images containing red and near-infrared channels. A sample image can be seen in Figure 3.1, where the near-infrared channel has been mapped to the green one and the red channel has been mapped to the red and blue ones.

CWFID is a public dataset created with the aim of making it possible to compare different classification systems for the agricultural domain. Therefore, in the proposing article, the dataset has been released with a specific splitting to be used. More precisely, the 60 images contained in the dataset can be split in two ways:

1. the first 20 images for training, the last 40 for testing;
2. 40 randomly selected images for training and the remaining 20 for testing.

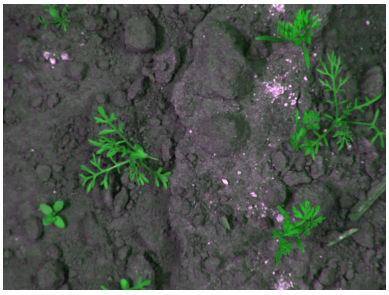


*Figure 3.1: CWFID image 027 as an example*

For the first splitting there are some initial results shipped with the dataset, for the second one there is still no result publicly available. We have therefore used both the proposed splittings, the first one to make our results comparable to the available ones and the second one to see how much the various networks suffer from the lack of big training datasets. From the training datasets of both the splittings we have selected some images for validation purposes. More specifically, we have selected the images 6, 12 and 18 for the first splitting and the images 11, 20, 41 and 52 for the second splitting.

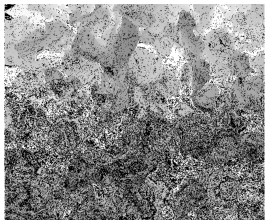
### 3.1.2 Broccoli dataset

From the original Broccoli dataset we have derived 300 $1920 \times 1080$ RGB images plus a depth channel with a resolution of $512 \times 242$ pixels. A sample image can be seen in Figure 3.2.

We have also tried to see whether the addition of the depth information can lead to substantially better results. This has been done by building a fourth channel representing the distance of the pixel from the camera. Since the depth and the color camera are placed in different position and have a different resolution, the depth channel is defined only for the central part of the image. Experiments shown that the the depth channel contains too much noise and thus does not help to segment images when used in their raw version.

For the Broccoli dataset we can only compare to the results released in [22], that have been obtained on 3D cloudpoints and with different ground truth annotation data, therefore the results are comparable just on a qualitative basis. From 300 images we have selected the first 150 images as training, the following 50 for validation and the remaining 100 for testing. The splitting has been done in a sequential way as the images are overlapping.

(a) Broccoli image 001



(b) Depth channel of image 001

Figure 3.2: Broccoli image 001

## 3.2 Adopted metrics

To measure the performance of the various classifiers on the CWFID dataset, we have adopted the metrics proposed in [15]. In detail, the system is measured in two stages: the first metric measures how much the system is able to separate the vegetation from the surrounding soil, whereas the second metric measures the ability of the system to correctly classify the various kinds of vegetations.

The first metric is the Jaccard index, and is computed as

$$seg_{acc} = \frac{true_{pos}}{true_{pos} + false_{pos} + false_{neg}}$$

(where the positive class is the vegetation and the negative is the soil) and averaged across all the test images.

The second set of metrics is composed by the average accuracy, precision, recall and f1 score averaged across all the test images. Considering that we have already measured the ability to discriminate the soil, those metrics are evaluated just on the vegetation pixels. Therefore we measure them only on the intersection between our vegetation mask and the ground truth for the same mask. The problem becomes binary and we consider the weed as the positive class. The metrics are thus computed as:

$$accuracy = \frac{true_{pos} + true_{neg}}{true_{pos} + true_{neg} + false_{pos} + false_{neg}},$$

$$precision = \frac{true_{pos}}{true_{pos} + false_{pos}},$$

$$recall = \frac{true_{pos}}{true_{pos} + false_{neg}},$$

$$f_1 = \frac{2 \times precision \times recall}{precision + recall}.$$

In [15] a border of 40 pixels is masked and ignored during the evaluation phase as the proposed system is not able to produce predictions there. Our system, instead, is able to produce accurate results even on the border of the images, but in order to have comparable results we adapt to the proposed evaluation model and mask the border.

For the Broccoli dataset we cannot use this evaluation as we do not have a three class problems, but a binary one. Therefore we just evaluate accuracy, precision, recall and f1 considering the broccoli heads as the positive class and removing a 40 pixels border around the images.

## 3.3 System description

This work is focused on the comparison of different classifiers for agricultural tasks. In order to compare them we have implemented a common system where new classifiers can be easily integrated and cooperate with existing ones. The system we propose can be divided in two separate subsystems: one based on neural networks and a more traditional one working on handcrafted features used to produce baseline results. The general structure of the system can be observed in Figure 3.3. The two subsystems are identified by the left- and right-side of Figure 3.3 and can be run independently.

To better explain the various stages of the pipeline, here we will present it assuming that the system is working on an image taken from CWFID. Images from the Broccoli dataset are treated in a similar way, the differences will be detailedly explained in Section 3.4.

### 3.3.1 Hand crafted features subsystem

The first subsystem contains most of the ideas that can be found in other state of the art systems. The pipeline is the following:

- the input image is transformed into a grayscale one computing a spectral index;
- the greyscale image is used to compute a mask that removes soil;
- the mask is cleaned to reduce false positives;
- a grid is drawn on the image;
- a tile is extracted above every unmasked (i.e., vegetation) *keypoint*[1];

[1]With *keypoints* we mean the intersection points identified by the grid.

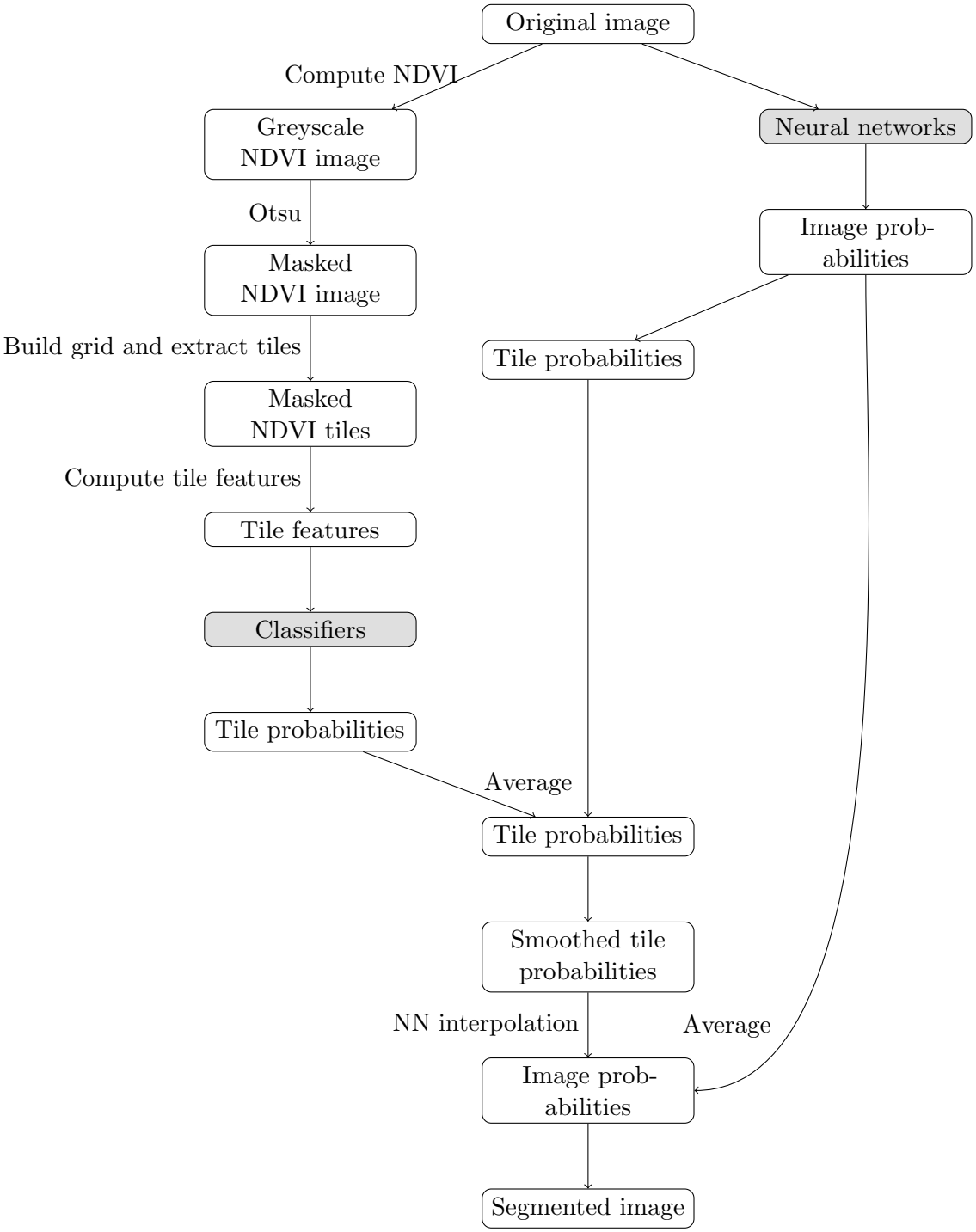



*Figure 3.3: General architecture of the implemented system*

- the NDVI channel of the tile is used to extract some features;
- the features are given to several classifiers;
- the results of the various classifiers are averaged and assigned to the center of the tile;
- the probabilities are spatially smoothed;
- each point of the image is assigned a probability using nearest neighbor interpolation.

### NDVI

As a first step, the system has to be able to distinguish vegetation from soil. To do that, the different reflectance of vegetation and soil in the red and near-infrared light is exploited [41]. To highlight this difference, the 2-channels images are transformed into single-channel ones by computing the Normalized Difference Vegetation Index (NDVI) at each pixel:

$$I_{NDVI} = \frac{I_{NIR} - I_R}{I_{NIR} + I_R}$$

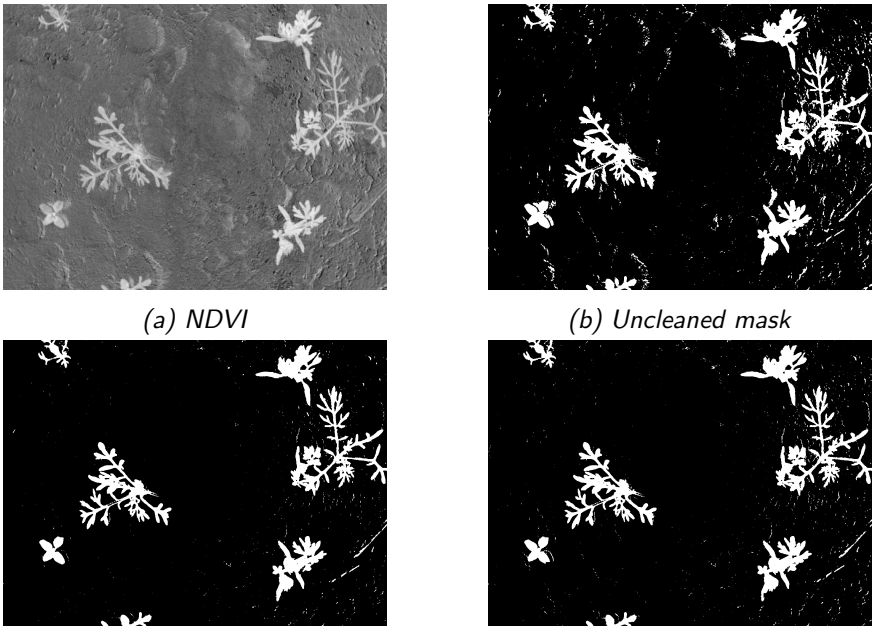
Where $I_R$ and $I_{NIR}$ respectively indicates the red and the near-infrared channels of the image. A sample image can be seen in Figure 3.4a.

### Binary mask for vegetation

The information obtained from NDVI is used to compute a binary mask. Since NDVI indicates how much the near-infrared channel is different from the red one, it can be used to identify vegetation areas. An higher NDVI thus indicates that an area is more probably vegetation. To obtain a binary mask, a threshold for NDVI has to be selected. As proposed in [16], the threshold value is selected using Otsu's method [31]. An example image can be seen in Figure 3.4b.

**Mask cleaning** Since the obtained mask contains some noise, we propose two different strategies to remove it.

**Minimum Near Infra Red Value** The masking procedure is based on the difference between the red and the near-infrared channel, but does not consider their absolute values. Therefore, the mask also includes dark pixels if their NIR channel is slightly above the red one. A simple strategy



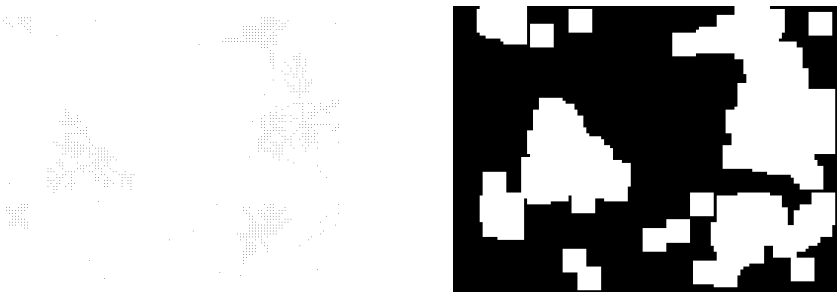
*(a) NDVI* *(b) Uncleaned mask*

*(c) Mask cleaned NIR $\geq$ 74* *(d) Mask cleaned neural network*

*Figure 3.4: Computing the mask for image 027*

to get rid of them is to remove all pixels whose NIR value is below a certain threshold. Experiments (reported in Section 4.2.1) have shown that a good value for the threshold is 74. An example of a cleaned mask can be seen in Figure 3.4c.

**Neural Network** Potena et al. proposed [36] to use a lightweight convolutional neural network to clean the mask. We implemented a similar network: first we clean the mask imposing a more conservative minimum NIR value (using a threshold of 65), then, in order to classify a pixel, we extract a 15x15 tile around it and feed it to a convolutional neural network. Since repeating this process for every pixel would be really demanding, the tiles to classify are extracted so as to be non overlapping and the result is applied to the entire tile. However, this approach remains relatively slow and moreover it does not lead to substantially better results as reported in Section 4.2.1. Therefore, it is not used in the default configuration of the system. The network structure is reported in Figure 2.1. An example of a mask cleaned using this approach can be seen in Figure 3.4d.



*(a) Centers of tiles in black* *(b) Overlapping tiles in white*

*Figure 3.5: Extracting the tiles from image 027*

### Image tiling

The binary mask is then used to extract smaller tiles from the image as proposed by Haug et al. [16]. This approach allows to classify the image without having to explicitly deal with overlapping leaves. Differently from Haug et al., we use a thicker grid, as it produces better results at the expense of a slight increase in computational demands. A grid with a spacing of 10x10 pixels is drawn on the image. Each intersection point that is above soil (i.e. black mask) is discarded, while all the other points are considered as centers of 80x80 tiles. Each tile will be classified separately and the result will be assigned to the central pixel. The effect of this step can be seen in Figure 3.5

### Features extraction

Each tile is used to compute a set of features. As proposed by Hall et al. [13], our system can work with two sets of features. The main one is the same proposed by Haug et al. [16] and reported in Table 3.1, the second one is the subset of those features that are robust to scale variation. Experiments have shown that the smaller set provides better results, as reported in Section 4.2.2. Moreover, the usage of a smaller number of features makes the system faster. The idea is to have a set of features that is fast to compute and is very resilient to condition variation, therefore our system does not contain complex features working on the texture of the leaves like the ones proposed by Lottes et al. [25]. Figure 3.6 contains the morphological objects used to compute the features.

*Table 3.1: Computed features*

| $f_i$ | Description | Scale invariant |
|---|---|---|
| $f_1$ | perimeter (length of contour) | no |
| $f_2$ | area (number of pixels covered by leaf) | no |
| $f_3$ | length of skeleton | no |
| $f_4$ | compactness (area / perimeter$^2$) | yes |
| $f_5$ | solidity (area / area of convex hull) | yes |
| $f_6$ | convexity (perimeter / perimeter of convex hull) | yes |
| $f_7$ | length of skeleton / perimeter | yes |
| $f_8$ | minimum of vegetation pixel intensities | yes |
| $f_9$ | maximum of vegetation pixel intensities | yes |
| $f_{10}$ | range of vegetation pixel intensities | yes |
| $f_{11}$ | mean of vegetation pixel intensities | yes |
| $f_{12}$ | median of vegetation pixel intensities | yes |
| $f_{13}$ | standard deviation of vegetation pixel intensities | yes |
| $f_{14}$ | kurtosis of vegetation pixel intensities | yes |
| $f_{15}$ | skewness of vegetation pixel intensities | yes |

### Classifiers

The computed features are used to train several baseline classifiers. The system can be easily extended to incorporate new classifiers, we have trained a random forest classifier, a support vector machine classifier and a gradient boosted classifier. All the classifiers produce as output the probabilities for the tile of being soil, crop or weed. In fact, even if they are used just on points that are masked as vegetation by the previous steps, the addition of a soil class helps to further reduce the number of misclassified points.

### Averaging probabilities

Every classifier outputs three probabilities (i.e., soil/weed/crop) for every keypoint of the grid. The probabilities of various classifiers are simply averaged at each keypoint. All the keypoints whose biggest probability corresponds to soil are added to the binary mask. The other points keep their weed probability so to discriminate between crop and weed. In Figure 3.7a is visible the output of this process on a sample image when averaging the outputs of the three classifiers.

*(a) NDVI* *(b) Contour and convex hull* *(c) Skeleton*

*Figure 3.6: NDVI index, contour, convex hull and skeleton of a CWFID tile used to compute the features*

### Spatial smoothing

In order to better exploit the local informations, Haug et al. proposed [16] to spatially smooth the probabilities using a Markov Random Field. To keep low the classification time we propose to use a simpler smoothing process. The idea is to remove outliers by exploiting the information provided by nearby keypoints. In fact, given that the tiles are overlapping, a keypoint is represented by more than one tile. The smoothing process we propose works as reported in Algorithm 1. This approach is much faster than the one proposed by Haug et al. [16], although it is not as accurate.

For each keypoint we consider its neighbors and compute the average probabilities among all of them. The probability of each keypoint in the neighborhood is then updated as the average between its current value and the computed average. The whole process is repeated more times. The result of the spatial smoothing process is visible in Figure 3.7b and is mostly noticeable for the lowest and uppermost plants on the right side of the image.

### Interpolation

At this point we produce a pixel-wise classification using both the binary mask and the smoothed probabilities. All the unmasked (i.e., vegetation) points receive the probability of the closest keypoint that has not been classified as soil. At the end, all the unmasked points that have a probability greater than 0.5 are considered as weed points, the others as crop. The effect of the interpolation is visible in Figure 3.7e, while the final classification is reported in Figure 3.7f.

**Algorithm 1** Spatial smoothing of tile probabilities

**Require:** prob_img: an image with the dimension of the original image whose values are -2 on all non keypoints, -1 on keypoints predicted as soil, $[0, 1]$ elsewhere

**Require:** x_cons, y_cons: positive integers representing the spatial extent of the smoothing process

**Require:** repetitions: positive integer representing the number of repetitions of the algorithm

**if** $repetitions = 0$ **then**
  return prob_img
**end if**
**for** (y,x) representing every available keypoint in the image **do**
  $tot \leftarrow 0$
  $num \leftarrow 0$
  **for** each keypoint in $y + y\_cons, x + x\_cons$ **do**
    **if** $keypoint\_prob \neq -2 \wedge keypoint\_prob \neq -1$ **then**
      $tot \leftarrow tot + keypoint\_prob$
      $num \leftarrow num + 1$
    **end if**
  **end for**
  **if** $num > 0$ **then**
    $avg \leftarrow \frac{tot}{num}$
    **for** each keypoint in $y + y\_cons, x + x\_cons$ **do**
      **if** $keypoint\_prob \neq -2 \wedge keypoint\_prob \neq -1$ **then**
        $keypoint\_prob \leftarrow \frac{avg + keypoint\_prob}{2}$
      **end if**
    **end for**
  **end if**
**end for**
return smooth(prob_img, x_cons, y_cons, repetitions - 1)

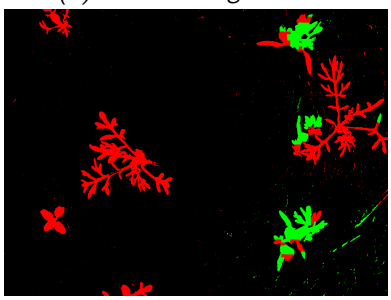
*(a) Aggregated weed probabilities*

*(b) Smoothed weed probabilities*

*(c) Detail of Figure 3.7a*

*(d) Detail of Figure 3.7b*

*(e) NN interpolation*



*(f) Final classification*

*Figure 3.7: Classification of image 027*

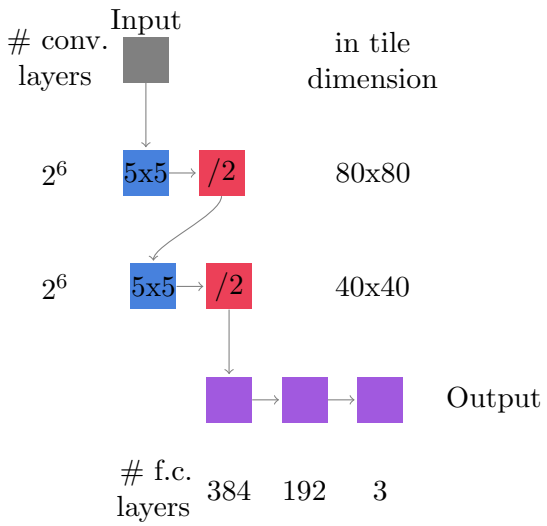



*Figure 3.8: Sliding window architecture. A blue square is a convolutional layer, red means max-pooling, violet is fully connected*

### 3.3.2 Deep Networks

Our system contains several deep neural networks. Their results can be combined with the one produced by the hand crafted features based subsystem or used on their own. We have experimented with four architectures and with some minor modifications, obtaining in total seven different networks.

#### Sliding window convolutional neural network

The first network follows the approach proposed by Potena, Nardi, and Pretto [36], and is composed by 2 $5 \times 5$ convolution+pooling layers followed by a batch normalization layer and three fully connected layers as reported in Figure 2.2. Figure 3.8 depicts the same network using the notation that we will also employ for the other networks. The only difference with the original network is the size of the input patches: we kept it to $80 \times 80$ pixels to better integrate it with the rest of the system. The network takes as input a window and produces a class output which is assigned to the central keypoint. Differently from Potena, Nardi, and Pretto, who are using randomly sampled points to extract the windows, we use the same keypoints that are used to compute the hand crafted features, thus making it easier to combine the output of this network with the output of the more traditional classifiers.

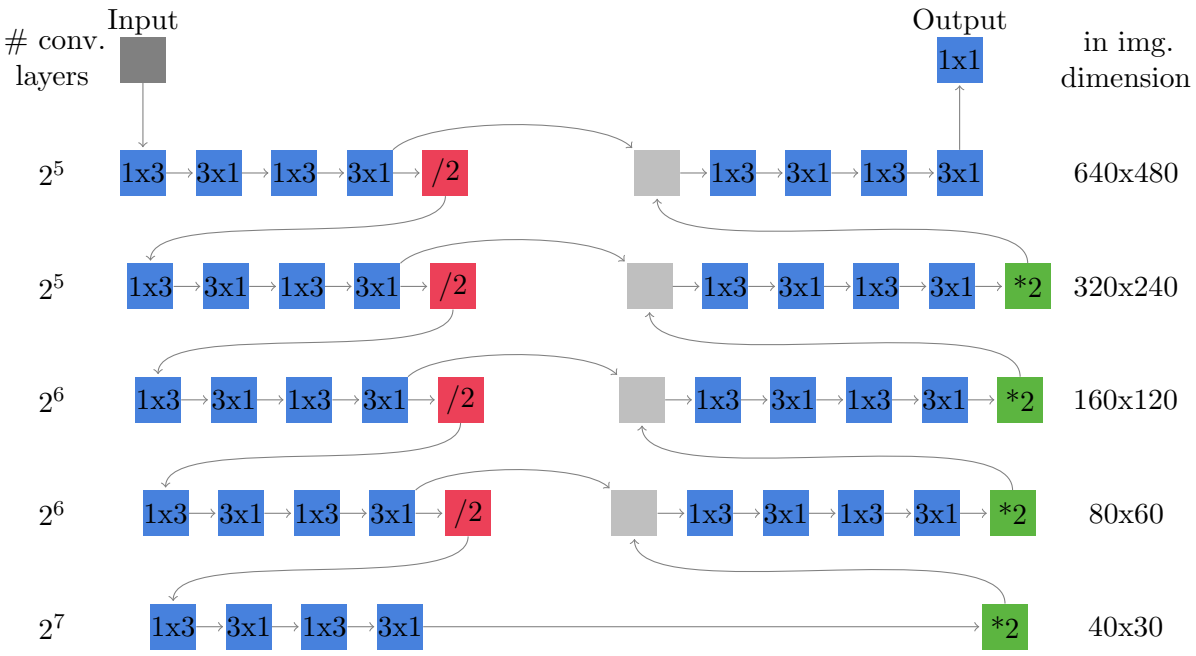



*Figure 3.9: U-Net like architecture. A blue square is a convolutional layer, red means max-pooling, green upconvolutional, grey concatenation*

### U-Net network

The U-Net architecture proposed by Ronneberger, Fischer, and Brox [38] has been implemented with some minor modifications. From the architecture reported in Figure 2.3 we have moved to the one depicted in Figure 3.9.

More specifically, we have reduced the number of convolutional filters to reduce the memory footprint. The padding of the convolutional layers has been modified to maintain the size of the images, given that in our task, differently from the one addresed in [38], we also have to produce precise predictions close to the borders of the images. Moreover, we have factored the $3 \times 3$ convolutions in two subsequent $1 \times 3$ and $3 \times 1$ convolutions as proposed by Szegedy et al. [50] to reduce the number of parameters and thus make the learning process easier.

To further reduce the memory requirements, the images are subsampled to half the size along every dimension before being fed to the network and the output is correspondingly upsampled. Apart from this, the network works on full size images, making the use of keypoints and tiles superfluous. However, if wanted, the network can also produce keypoint probabilities by averaging the predictions of the points in a window drawn around the keypoints. In this way, the results of this network can be smoothed and interpolated like

the ones produced by the hand crafted features based subsystem.

The adopted nonlinearity is ReLU [19] and the weight initialization strategy is He Normal [17].

### Recurrent networks

Basing on the works [52] [53] by Visin et al., we have experimented several recurrent network. From [52] we have taken the ReNet layer, which is composed by 4 recurrent neural networks that sweep the image both vertically and horizontally. In this way, each pixel's activation function depends upon the whole image, instead of the neighboring of the pixel as it happens with convolutional layers. Given that, in its default configuration, the ReNet layer reduces the dimension of the network by a factor of 2, it can be used as a drop-in replacement for the usually adopted convolution+pooling layers. In our experiments, we have tried to use these layers in the contracting path of the U-Net architecture, experimenting with the number of hidden units of the network. For the expanding path of the network we have kept the convolutional layers as, ideally, an activation function with a small spatial extent is better suited for an upsampling job. We will refer to these networks as *U-ReNet* and *U-ReNet2*, where the former indicates the network where the number of hidden units of the ReNet layers is kept fixed while the latter indicates the network with a varied number of hidden units. More specifically, the number of hidden units follows the same scheme adapted for the U-Net network. Figure 3.10 shows the architecture of this network.

For what concerns the ReSeg [53] network, it is composed by three ReNet layers followed by an upsampling layer. Firstly we have experimented the network on its own, then we have tried to preprocess the input data with some pretrained convolutional layers, as suggested in the paper presenting ReSeg. For those convolutional layers, we have used the pre-trained layers of U-Net. The original ReSeg architecture is reported in Figure 2.5, while the architecture with the convolutional layers is reported in Figure 3.11.

As a final experiment, we have tried to ease the job of the upconvolutional stage by using a concatenation layer combining the output of the upconvolutional layer with the input. In this way, the output layer can use the input image to better follow the borders of the image and thus produce more accurate results. The architecture is reported in Figure 3.12

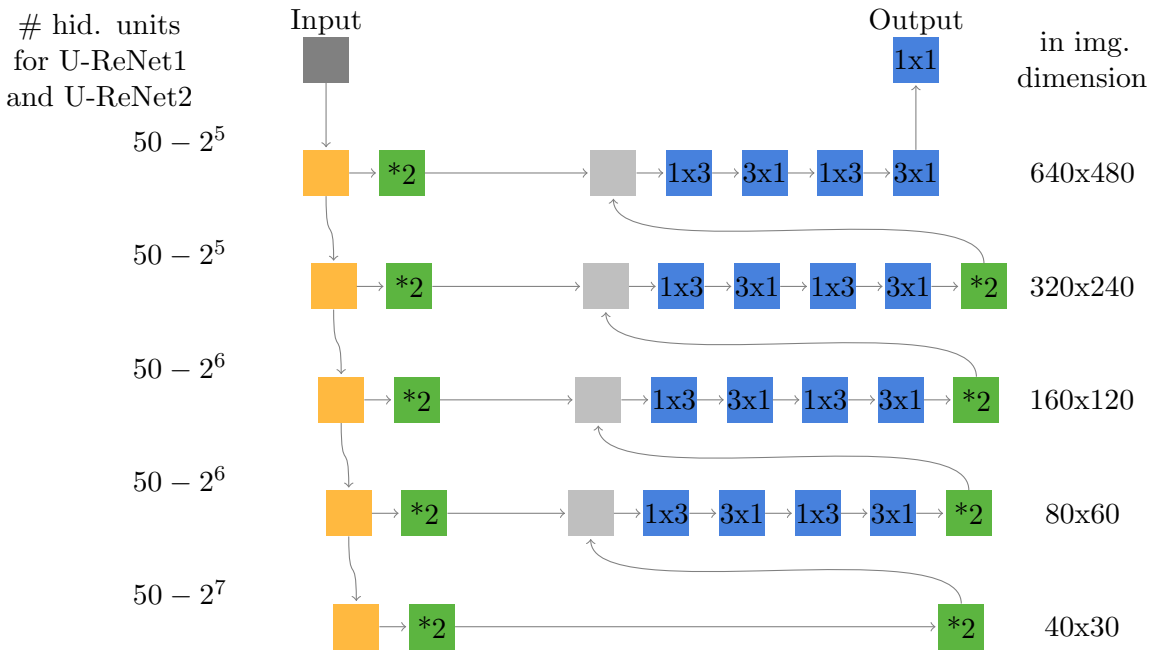



*Figure 3.10: U-ReNet architecture. A blue square is a convolutional layer, yellow represents a ReNet layer, green is upconvolutional and grey concatenation. On the left is reported the number of hidden units for the two networks tested, the number of convolutional layers is the same used in the U-net like network*

## 3.4 Differences for the Broccoli dataset

The Broccoli dataset presents some peculiarities that involved making some modifications to the presented architecture. As a first thing, the Broccoli dataset is composed by RGB images, thus having one more channel than the CWFID images, which are composed by a red and a near-infrared channels. Moreover, the images of the Broccoli dataset do not contain much soil but contain many leaves. Combining this with the fact that no thresholding technique resulted in a good elimination of the little soil present without substantially removing areas of broccoli heads, we decided to treat the problem as a 2 classes segmentation problem: we just separate broccoli heads from the rest of the world (i.e. soil and other vegetation). Therefore we have to use an unmasked picture while extracting the tiles, which leads to a really big dataset in terms of training samples. Since this is not ideal in terms of memory efficiency, we have also tried to use the U-Net network (with a very low threshold probability of 20%) in order to reduce the dataset size. This mimics the idea presented in [36], where a lightweight network is used to preprocess the images and to remove soil. Our network is not lightweight

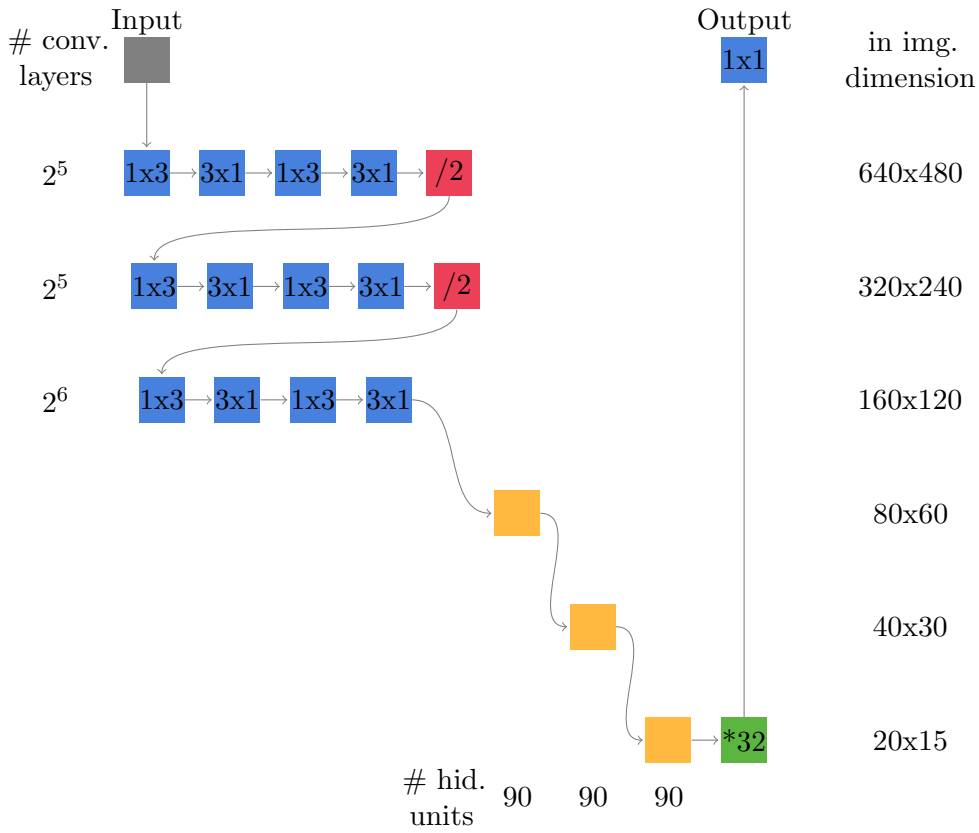



*Figure 3.11: ReSeg + convolution architecture. A blue square is a convolutional layer, yellow represents a ReNet layer, green is upconvolutional and red is max pooling. Considering the image starting from the yellow layers, it depicts the ReSeg network*

and it is not removing soil, but the goal is, in both the cases, to reduce the number of tiles to be classified. Therefore, since we already have a trained network, we can use it in conjunction with a really low threshold to mimic a lightweight network.

Furthermore, the tile-based classifiers have not been trained for the depth-enriched version of the Broccoli dataset due to their really high computational cost.

The greyscale image used to extract the features cannot be computed using NDVI, since we have no near infrared channel. We have therefore adopted the combined index proposed in [12], which is based on three different indexes. More specifically, starting from an RGB images with values in 0-255 ranges, first we compute the normalized values in 0-1 ranges:

$$R_n = \frac{R}{R_{max}}, G_n = \frac{G}{G_{max}}, B_n = \frac{B}{B_{max}}$$

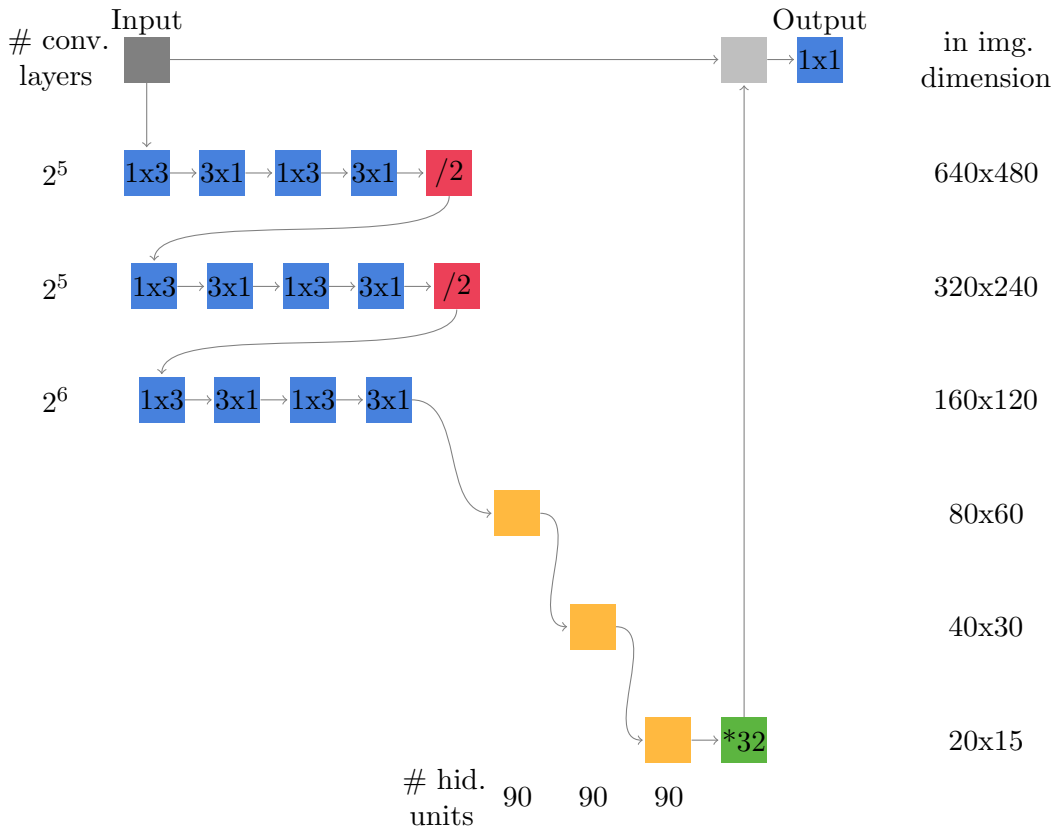



*Figure 3.12: ReSeg + convolution +concatenation architecture. A blue square is a convolutional layer, yellow represents a ReNet layer, green is upconvolutional, grey is the concatenation layer and red is max pooling*

and then normalize again in a relative way:

$$r = \frac{R_n}{R_n + G_n + B_n}, \; g = \frac{G_n}{R_n + G_n + B_n}, \; b = \frac{B_n}{R_n + G_n + B_n}.$$

The indexes that will be combined (excess green, color index of vegetation extraction, vegetative) are computed as [12]:

$$ExG = 2g - r - b,$$

$$CIVE = 0.441r - 0.811g + 0.385b + 18.78745,$$

$$VEG = \frac{g}{r^{\alpha} b^{1-\alpha}},$$

with $\alpha = 0.667$.

And, at last, the indexes are combined as [12]:

$$COM = 0.36ExG + 0.47CIVE + 0.17VEG.$$

This combined index is then used to compute the features of Table 3.1, exactly like the NDVI index is used for CWFID.

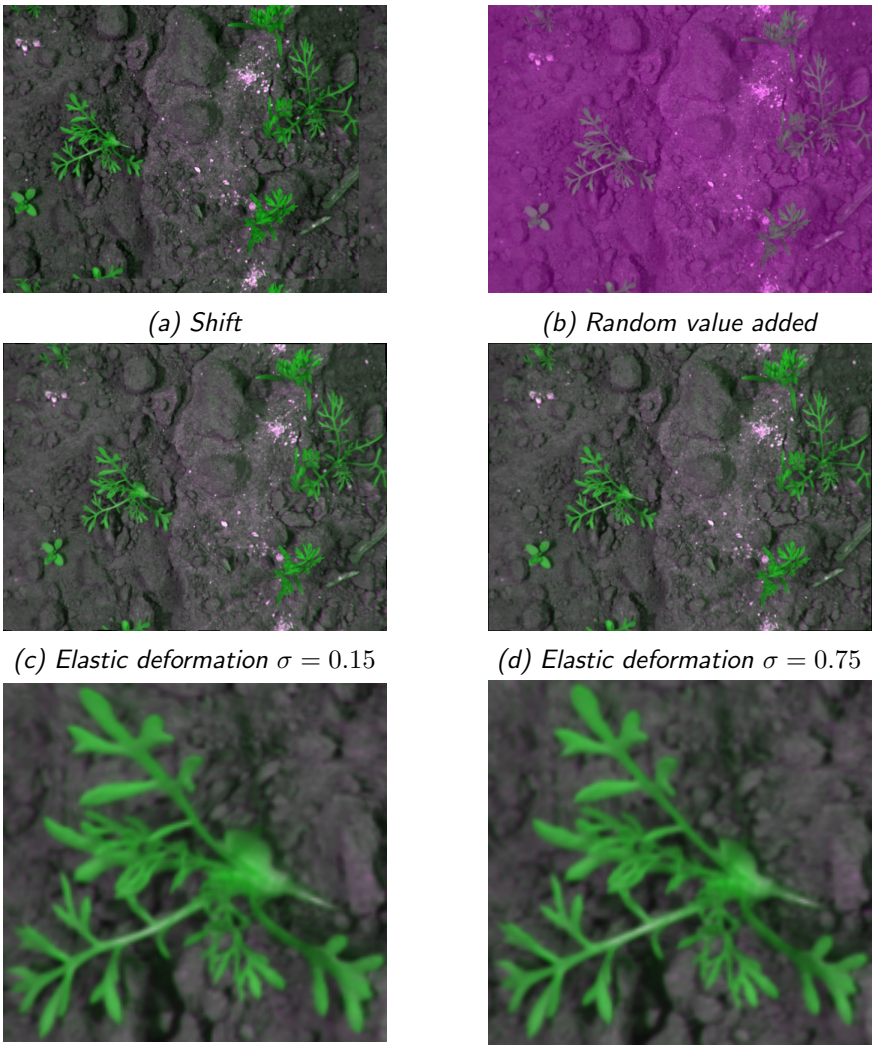
## 3.5 Data augmentation techniques

Given the small number of training data available, data augmentation techniques have been used heavily. More specifically, each image is mirrored horizontally, vertically and diagonally while building the dataset. Moreover, during training time, each image is randomly perturbed in several ways:

1. a random translation is applied in both the x and y directions;
2. a random value is added to all the pixels on all the channels;
3. an elastic transformation is applied;
4. all the networks contain a gaussian noise layer.

In this way, the networks are trained by never seeing the same image more than one time, implicitly augmenting the number of training data. In order to further prevent overfitting, we also apply dropout on the last layer and an $l_2$ regularization term. Experiments shown that the regularization term is not really important to prevent overfitting, whereas the elastic deformation and the dropout are much more useful hyperparameters.

As explained in Section 2.3.4, the effect of the elastic transformation depends on the chosen value for $\sigma$: lower values are more similar to a random field, while bigger values are similar to a translation. We have thus experimented the usage of two possible values for $\sigma$ so to obtain both the effects on all the networks except for the sliding windows one. In fact, since it works with overlapping tiles, the effect of the translation is useless. The effects of the various techniques are reported in Figure 3.13.



*(a) Shift* *(b) Random value added*

*(c) Elastic deformation $\sigma = 0.15$* *(d) Elastic deformation $\sigma = 0.75$*

*(e) Detail of Figure 3.13c* *(f) Detail of Figure 3.13d*

*Figure 3.13: Distortions of image 027*

# Chapter 4

# Performed experiments

*"Sometimes it seems as though each new step towards AI, rather than producing something which everyone agrees is real intelligence, merely reveals what real intelligence is not."*

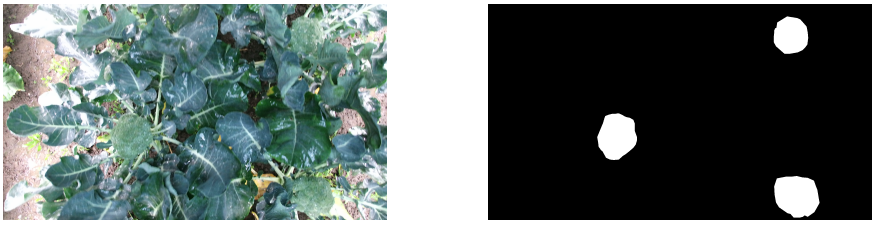
Gödel, Escher, Bach: An Eternal Golden Braid

In this section we report the various steps that we have taken in order to materially implement the proposed system.

## 4.1 Broccoli dataset annotation

From the 3D data provided in the Broccoli dataset we have extracted 2D images suitable for our networks. The dataset also contained a ground truth in a point cloud format. As suggested by the authors of the dataset, we did not convert this information in a 2D segmentation mask, but instead we created from scratch the annotation images.

The final goal of an automated harvesting system is to distinguish harvestable plants, which implies being able to compute the growth status of a plant even when it is occluded. In the case of broccoli, this means being able to compute the dimension of the heads even when they are covered by other leaves. Therefore, we have selected 300 images from the dataset and created their annotation mask using LabelMe [39], trying to annotate as broccoli heads even the parts of the images where we thought a broccoli head was present behind the leaves. A sample annotation is reported in Figure 4.1b.



*(a) Image* *(b) Annotation*

*Figure 4.1: Annotation for Broccoli image 001*

*Table 4.1: Background removal with Otsu's tresholding for the CWFID dataset*

| Split | Jaccard | Recall |
|---|---|---|
| 1 | 0.77625577 | 0.95806455 |
| 2 | 0.77294238 | 0.96049285 |

## 4.2 System training

In this subsection we report the details common to the training of all the classifiers on all the datasets.

### 4.2.1 Background removal

By background removal phase we refer here to the first stage of the features based classification pipeline and not to the final ability of the system to discern soil from vegetation. In fact, later stages of the pipeline can still classify a point as soil. For this stage we have thus selected the Jaccard index as a main metric, but, since at this point we can accept to classify a soil point as vegetation (whereas the contrary would be irreversible), we have also considered the recall, trying to preserve all the vegetation points. This stage has thus to be considered as a really fast way to reduce the workload of the later classifiers, which, being more complex, are better at recognizing soil, but would also be expensive to use on the entire images.

On the CWFID dataset, we have first measured the metrics on the vegetation mask obtained when just using the NDVI index and the Otsu's thresholding. The results are reported in Table 4.1.

In order to produce a cleaner mask, we have tried to impose a minimum value for the near-infrared channel, obtaining the results reported in Table 4.2 for the first split of CWFID. We have thus selected a minimum value of 74. In Table 4.3 are reported the results for the second split, where a similar trend is found, thus confirming the choice for the minimum value.

*Table 4.2: Cleaning the background mask with a minimum NIR value on CWFID split 1*

| Min NIR | Jaccard | Recall |
|---|---|---|
| 0 | 0.77625577 | 0.95806455 |
| 10 | 0.77625577 | 0.95806455 |
| 30 | 0.78089942 | 0.95806455 |
| 50 | 0.83112937 | 0.95806455 |
| 60 | 0.86640958 | 0.95806455 |
| 65 | 0.88361181 | 0.95732255 |
| 70 | 0.88776956 | 0.94872807 |
| 72 | 0.88854491 | 0.94471073 |
| 73 | 0.88885944 | 0.94251554 |
| 74 | **0.88892176** | 0.94027644 |
| 75 | 0.88882208 | 0.93788907 |
| 80 | 0.88594178 | 0.92389516 |
| 90 | 0.86154118 | 0.88167802 |

*Table 4.3: Cleaning the background mask with a minimum NIR value on CWFID split 2*

| Min NIR | Jaccard | Recall |
|---|---|---|
| 0 | 0.77294238 | 0.96049285 |
| 10 | 0.77294238 | 0.96049285 |
| 30 | 0.77746836 | 0.96049285 |
| 50 | 0.82479751 | 0.96049285 |
| 60 | 0.86169192 | 0.96049285 |
| 65 | 0.87954406 | 0.9598068 |
| 70 | 0.88545427 | 0.95216339 |
| 72 | 0.88689477 | 0.948672 |
| 73 | 0.88755181 | 0.94681879 |
| 74 | 0.8880013 | 0.94487607 |
| 75 | 0.88826237 | 0.94281095 |
| 76 | 0.88847851 | 0.940691 |
| 77 | **0.88851354** | 0.93846262 |
| 78 | 0.88846831 | 0.93615789 |
| 79 | 0.88821345 | 0.93366474 |
| 80 | 0.88768638 | 0.93106618 |
| 90 | 0.87110011 | 0.89640984 |

As an alternative cleaning methodology, we have experimented the solution proposed by Potena, Nardi, and Pretto. On the first split of CWFID, we have decided to first clean the mask with a lower minimum near-infrared value (65) so to obtain a good recall and then we have trained the neural network

*Table 4.4: Number of training samples (tiles) for the various datasets*

| Dataset | Number of training samples |
|---|---|
| CWFID 1 | 20 040 |
| CWFID 2 | 40 318 |
| Broccoli | 263 176 |

reported in Figure 2.1. We then tested its performance on the same dataset, obtaining a Jaccard index of 0.88821345 and a recall of 0.93366474. Since those results are not better than the ones obtained with the minimum NIR value, and moreover they drastically augment the computational complexity, we decided to not experiment further with this approach.

For the Broccoli dataset, we did not produce a separate ground truth for the soil. In fact, broccoli images do not contain much soil and thus the reduction in the computational time from its removal would be smaller with regard to the one obtained in CWFID, where the soil composes the vast majority of the images. However, we experimented the use of the U-Net network to remove some tiles from the dataset: by using a very low threshold on the output of U-Net, we are able to keep the vast majority of the image while discarding the leaves points about which the network is absolutely sure that they are not part of a broccoli head. This strategy (with a threshold of 20% that allows to obtain a recall greater than 0.99 with an accuracy of 0.8) has been applied to the 2D Broccoli dataset, while for the dataset with the depth channel we have decided to not implement the tile-based classifiers, given the really bad performances that they exhibited in the 2D Broccoli dataset. This strategy is similar to the one proposed in [36], but, instead of using a lightweight neural network, we use a more complex one imposing a low threshold.

### 4.2.2 Classifiers

Once obtained the masks for the vegetation, we have drawn a sparse grid over the images with a spacing of $10 \times 10$ pixels. Among all the keypoints, we have selected the keypoints above the pixels identified as vegetation and extracted their surrounding tiles. Those tiles have then been used to compute the features reported in Table 3.1, so as to create a training dataset for the classifiers. In Table 4.4 is reported the number of tiles (and therefore of training samples) for the various datasets.

We now explain in detail the steps taken for the training of the various classifiers.

*Table 4.5: Hyperparameters of the random forests*

| Dataset | Parameter | Value | OOB accuracy |
|---|---|---|---|
| CWFID split 1 | Number of trees | 990 | 90.34% |
| | Splitting criterion | gini | |
| | Number of features | sqrt | |
| CWFID split 1 SFO | Number of trees | 690 | 90.64% |
| | Splitting criterion | entropy | |
| | Number of features | sqrt | |
| CWFID split 2 SFO | Number of trees | 830 | 88.80% |
| | Splitting criterion | entropy | |
| | Number of features | $\log_2$ | |
| Broccoli SFO | Number of trees | 350 | 98.50% |
| | Splitting criterion | entropy | |
| | Number of features | $\log_2$ | |

**Random Forests**

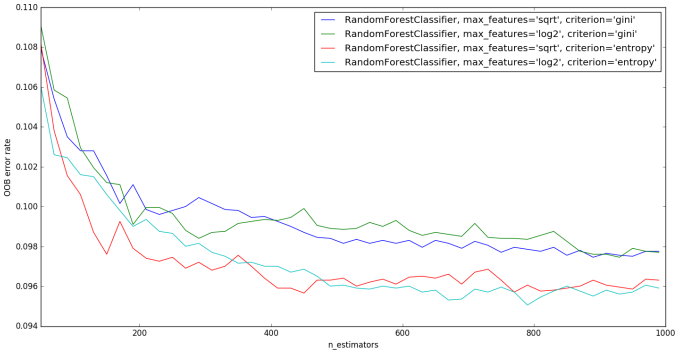
For the hyperparameters selection, we had to decide:

- the number of features considered at each splitting. We just evaluated two choices: the $\log_2$ and the square root of the original number of features;
- the criterion used to compute the information gain obtained from a splitting. We evaluated the gini index and the entropy;
- the number of trees in the forest.

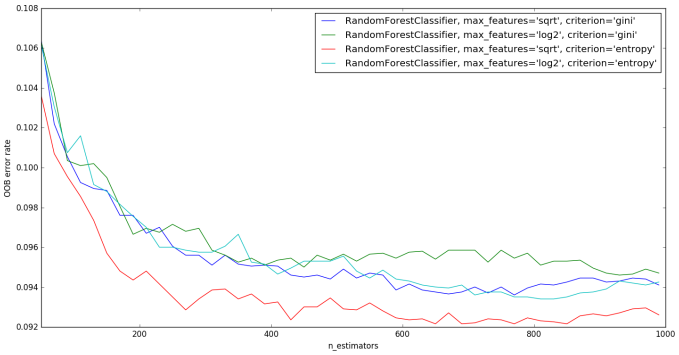
To do that, we built 4 models (with all the possible combinations of splitting criterion and considered features) and progressively added trees to each model. At each step we evaluated the out of bag error rate (i.e. $1 - accuracy$) (explained in Section 2.2.1), obtaining the results reported in Figure 4.2.

From the comparison of Figure 4.2a and Figure 4.2b it can be seen that the usage of a larger set of features does not result in a smaller error. We have therefore decided to use just the scale invariant features for the training of the random forests on the other datasets.

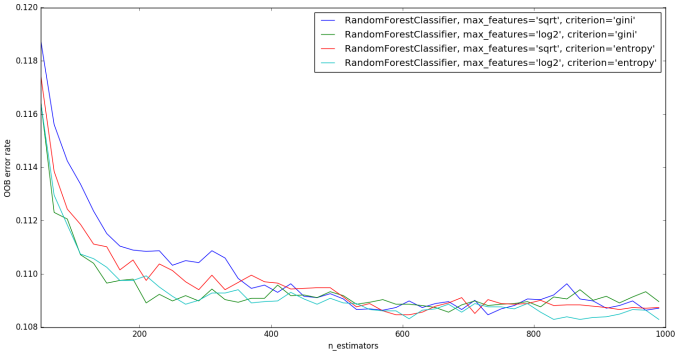
From these results, we selected the hyperparameters yielding the lowest out of bag error and saved the corresponding model. The selected hyperparameters and the corresponding errors are reported in Table 4.5.



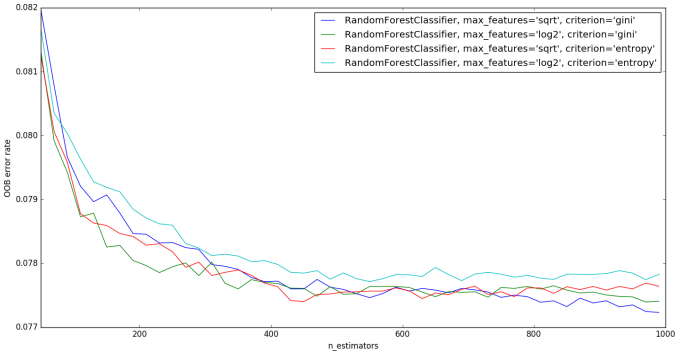

*(a) CWFID split 1 using all features*




*(b) CWFID split 1 using scale free features*




*(c) CWFID split 2 using scale free features*




*(d) Broccoli using scale free features*

*Figure 4.2: Exploration of the random forest hyperparameters*

*Table 4.6: SVM for CWFID split 1 with all the features. Selection of the kernel functions*

| Kernel | average accuracy | accuracy std |
|---|---|---|
| linear | 77.27% | 2.57% |
| polynomial | 77.72% | 3.31% |
| **radial basis** | **78.62%** | **2.71%** |
| sigmoid | 66.87% | 0.01% |

*Table 4.7: SVM for CWFID split 1 with scale invariant features. Selection of the kernel functions*

| Kernel | average accuracy | accuracy std |
|---|---|---|
| linear | 76.89% | 2.55% |
| polynomial | 77.57% | 3.25% |
| **radial basis** | **78.51%** | **2.74%** |
| sigmoid | 66.87% | 0.01% |

**Support vector machines**

The tuning of the SVM hyperparameters consisted in a two phases process. First we selected the kernel function, then we proceeded to tune the other hyperparameters. The research of the optimal hyperparameters has been staged on a progressive scale from coarse to fine. In order to evaluate the goodness of the hyperparameters we have evaluated the accuracy of each model with a 5-fold cross validation schema. Before being fed to the model, the data have been preprocessed so to obtain a zero mean and a unitary variance.

In the following tables (from Table 4.6 to Table 4.9) are reported the results obtained while deciding the kernel functions for the various datasets.

We have thus decided to use radial basis functions while training the models for the CWFID dataset and to just use a linear classifier for the Broccoli dataset. Once decided the kernels, we have tuned the other hyperparameters. For the radial basis function we tuned $C$ and $\gamma$, while for the linear version we tuned both an $l_1$ and an $l_2$ penalization term and tried to see whether we could benefit from the introduction of a class weight in order to counterbalance the unbalanced classed. The tables from 4.10 to 4.13 contain the results of this optimization phase. As happened with the random forests, from the comparison of Table 4.10 and Table 4.11 we can see that the introduction of additional features does not result in a better classification accuracy. Moreover, from the comparison of Table 4.10, Table 4.11 and Table 4.12 we can see that on all the versions of the CWFID

*Table 4.8: SVM for CWFID split 2 with scale invariant features. Selection of the kernel functions*

| Kernel | average accuracy | accuracy std |
|---|---|---|
| linear | 79.00% | 1.93% |
| polynomial | 79.77% | 2.13% |
| **radial basis** | **81.12%** | **2.07%** |
| sigmoid | 68.07% | 0.01% |

*Table 4.9: SVM for Broccoli with scale invariant features. Selection of the kernel functions*

| Kernel | average accuracy | accuracy std |
|---|---|---|
| **linear** | **91.40%** | **0.61%** |
| polynomial | 90.89% | 0.51% |
| radial basis | 91.36% | 0.67% |
| sigmoid | 78.34% | 0.01% |

dataset the classifier reached their optimum with similar hyperparameters, thus indicating that those values are good for the specific problem.

*Table 4.10: Tuning SVM on CWFID 1 all the features*

| C | $\gamma$ | acc. avg | acc. std |
|---|---|---|---|
| 0.1 | 1.0 | 69.13% | 0.97% |
| 0.1 | 0.1 | 77.23% | 2.96% |
| 0.1 | 0.01 | 76.14% | 3.39% |
| 1.0 | 1.0 | 73.10% | 1.04% |
| 1.0 | 0.1 | 78.42% | 2.77% |
| 1.0 | 0.01 | 77.98% | 2.86% |
| 5.0 | 0.006 | 78.75% | 2.82% |
| 5.0 | 0.01 | 78.97% | 2.77% |
| 5.0 | 0.02 | 78.99% | 2.63% |
| 8.0 | 0.008 | 78.86% | 2.65% |
| 8.0 | 0.01 | 78.94% | 2.71% |
| 8.0 | 0.013 | 79.00% | 2.67% |
| 9.0 | 0.011 | 78.99% | 2.61% |
| 9.0 | 0.01 | 78.98% | 2.71% |
| 9.0 | 0.009 | 78.91% | 2.70% |
| 10.0 | 1.0 | 70.82% | 1.91% |
| 10.0 | 0.1 | 77.03% | 2.19% |
| 10.0 | 0.02 | 78.93% | 2.49% |
| 10.0 | 0.013 | 78.94% | 2.60% |
| 10.0 | 0.011 | 78.99% | 2.64% |
| **10.0** | **0.01** | **79.01%** | **2.68%** |
| 10.0 | 0.009 | 78.91% | 2.69% |
| 10.0 | 0.008 | 78.89% | 2.72% |
| 10.0 | 0.006 | 78.86% | 2.68% |
| 10.0 | 0.002 | 78.12% | 2.84% |
| 10.0 | 0.005 | 78.80% | 2.84% |
| 10.0 | 0.02 | 78.74% | 2.41% |
| 10.0 | 0.01 | 78.99% | 2.57% |
| 10.0 | 0.006 | 78.86% | 2.69% |
| 11.0 | 0.011 | 78.98% | 2.62% |
| 11.0 | 0.01 | 78.99% | 2.67% |
| 11.0 | 0.009 | 78.96% | 2.66% |
| 13.0 | 0.008 | 78.90% | 2.66% |
| 13.0 | 0.01 | 79.00% | 2.60% |
| 13.0 | 0.013 | 78.86% | 2.52% |
| 20.0 | 0.002 | 78.43% | 2.85% |
| 20.0 | 0.005 | 78.87% | 2.58% |
| 20.0 | 0.01 | 78.96% | 2.59% |
| 50.0 | 0.002 | 78.77% | 2.76% |
| 50.0 | 0.005 | 78.93% | 2.60% |
| 50.0 | 0.01 | 78.80% | 2.35% |

*Table 4.11: Tuning SVM on CWFID 1 SFO*

| C | $\gamma$ | acc. avg | acc. std |
|---|---|---|---|
| 0.1 | 0.1 | 77.09% | 3.08% |
| 0.1 | 1.0 | 70.24% | 1.39% |
| 0.1 | 10.0 | 66.87% | 0.01% |
| 1.0 | 0.002 | 76.03% | 2.97% |
| 1.0 | 0.01 | 77.75% | 2.81% |
| 1.0 | 0.1 | 78.43% | 2.73% |
| 1.0 | 1.0 | 73.44% | 1.25% |
| 1.0 | 10.0 | 66.87% | 0.01% |
| 5.0 | 0.002 | 77.22% | 2.83% |
| 5.0 | 0.01 | 78.74% | 2.74% |
| 5.0 | 0.1 | 77.68% | 2.25% |
| 10.0 | 0.002 | 77.80% | 2.78% |
| 10.0 | 0.004 | 78.23% | 2.67% |
| 10.0 | 0.01 | 78.88% | 2.68% |
| 10.0 | 0.02 | 78.82% | 2.52% |
| 10.0 | 0.1 | 77.18% | 2.13% |
| 10.0 | 1.0 | 70.83% | 2.13% |
| 10.0 | 10.0 | 66.87% | 0.02% |
| 15.0 | 0.006 | 78.85% | 2.64% |
| 15.0 | 0.01 | 79.00% | 2.63% |
| 15.0 | 0.013 | 78.96% | 2.63% |
| 17.0 | 0.008 | 78.91% | 2.65% |
| **17.0** | **0.01** | **79.08%** | **2.60%** |
| 17.0 | 0.011 | 79.00% | 2.63% |
| 20.0 | 0.004 | 78.53% | 2.64% |
| 20.0 | 0.006 | 78.92% | 2.62% |
| 20.0 | 0.008 | 78.98% | 2.63% |
| 20.0 | 0.01 | 79.03% | 2.61% |
| 20.0 | 0.011 | 78.97% | 2.60% |
| 20.0 | 0.013 | 78.81% | 2.52% |
| 20.0 | 0.02 | 78.58% | 2.45% |
| 25.0 | 0.008 | 79.06% | 2.59% |
| 25.0 | 0.01 | 79.01% | 2.59% |
| 25.0 | 0.011 | 78.83% | 2.56% |
| 30.0 | 0.006 | 78.94% | 2.58% |
| 30.0 | 0.01 | 78.91% | 2.54% |
| 30.0 | 0.013 | 78.77% | 2.44% |
| 50.0 | 0.004 | 78.84% | 2.53% |
| 50.0 | 0.01 | 78.76% | 2.41% |
| 50.0 | 0.02 | 78.45% | 2.40% |

*Table 4.12: Tuning SVM on CWFID 2 SFO*

| C | $\gamma$ | acc. avg | acc. std |
|---|---|---|---|
| 0.1 | 0.1 | 80.49% | 2.02% |
| 0.1 | 1.0 | 75.17% | 0.68% |
| 0.1 | 10.0 | 68.08% | 0.00% |
| 0.5 | 0.005 | 79.68% | 2.00% |
| 0.5 | 0.01 | 80.20% | 2.01% |
| 0.5 | 0.02 | 80.48% | 2.05% |
| 1.0 | 0.002 | 79.26% | 1.96% |
| 1.0 | 0.005 | 80.05% | 2.03% |
| 1.0 | 0.01 | 80.40% | 2.03% |
| 1.0 | 0.02 | 80.63% | 2.06% |
| 1.0 | 0.1 | 81.18% | 2.16% |
| 1.0 | 1.0 | 77.53% | 1.49% |
| 1.0 | 10.0 | 68.06% | 0.01% |
| 5.0 | 0.002 | 80.12% | 1.94% |
| 5.0 | 0.005 | 80.48% | 1.98% |
| 5.0 | 0.01 | 80.74% | 1.94% |
| 5.0 | 0.02 | 81.05% | 1.95% |
| 5.0 | 0.04 | 81.22% | 2.00% |
| 5.0 | 0.1 | 80.79% | 2.19% |
| 7.0 | 0.028 | 81.19% | 1.91% |
| 7.0 | 0.04 | 81.24% | 1.96% |
| 7.0 | 0.066 | 81.06% | 2.11% |
| 10.0 | 0.002 | 80.30% | 2.00% |
| 10.0 | 0.01 | 80.91% | 1.99% |
| 10.0 | 0.02 | 81.19% | 1.93% |
| 10.0 | 0.028 | 81.23% | 1.92% |
| **10.0** | **0.04** | **81.31%** | **1.97%** |
| 10.0 | 0.066 | 81.01% | 2.12% |
| 10.0 | 0.1 | 80.42% | 2.11% |
| 10.0 | 1.0 | 73.82% | 1.20% |
| 10.0 | 10.0 | 68.04% | 0.06% |
| 12.0 | 0.028 | 81.25% | 1.92% |
| 12.0 | 0.04 | 81.27% | 1.97% |
| 12.0 | 0.066 | 81.00% | 2.13% |
| 15.0 | 0.02 | 81.20% | 1.86% |
| 15.0 | 0.04 | 81.22% | 1.95% |
| 15.0 | 0.1 | 80.14% | 2.04% |

*Table 4.13: Tuning SVM on Broccoli SFO*

| Class weight | $l_1$ | $l_2$ | acc. avg | acc. std |
|---|---|---|---|---|
| No | 0.1 | 0.0 | 91.36% | 0.01% |
| **No** | **1.0** | **0.0** | **91.36%** | **0.56%** |
| No | 10.0 | 0.0 | 91.36% | 0.56% |
| No | 100.0 | 0.0 | 91.36% | 0.56% |
| No | 0.0 | 0.1 | 91.36% | 0.56% |
| No | 0.0 | 1.0 | 91.35% | 0.57% |
| No | 0.0 | 10.0 | 91.35% | 0.56% |
| No | 0.0 | 100.0 | 91.35% | 0.56% |
| Yes | 0.1 | 0.0 | 88.09% | 0.43% |
| Yes | 1.0 | 0.0 | 88.11% | 0.42% |
| Yes | 10.0 | 0.0 | 88.12% | 0.42% |
| Yes | 100.0 | 0.0 | 88.12% | 0.48% |
| Yes | 0.0 | 0.1 | 88.17% | 0.42% |
| Yes | 0.0 | 1.0 | 88.20% | 0.42% |
| Yes | 0.0 | 10.0 | 88.20% | 0.43% |
| Yes | 0.0 | 100.0 | 88.20% | 0.42% |

**Gradient Boosted Trees**

For the gradient boosted trees we have tuned the hyperparameters with a sequential approach, using a 5-fold cross validation to measure the f1-score. The followed approach is reported below[1]:

1. Using an high learning rate, select the number of trees.
2. Select the maximum depth and the minimum number of child of the trees.
3. Tune $\gamma$.
4. Recalibrate the number of trees.
5. Tune the subsampling and colsampling hyperparameters.
6. Select an $l_1$ and an $l_2$ regularization terms.
7. Lower the learning rate and recalibrate the number of trees.

For the selection of the number of trees we have followed an iterative approach: starting from an empty forest we added one tree per time until the cross validation score did not improve for 50 rounds.

For the score, we have selected the f1-score instead of accuracy as done for the other classifiers. This is due to the fact that accuracy gave results really similar to f1, except for some configurations where it was really noisy. In the following pages we report the training details for the various datasets.

[1]The code used for the hyperparameters selection has been adapted from [6].

## CWFID split 1 using all the features

Starting with a learning rate of 0.5 we have selected 379 trees, we have then tuned the depth and the minimum child weight:

| Max depth | min child w | F1 avg | F1 std |
|---|---|---|---|
| 1 | 3 | 0.64747 | 0.06858 |
| 1 | 5 | 0.64628 | 0.07049 |
| 1 | 7 | 0.64900 | 0.06798 |
| 1 | 8 | 0.64968 | 0.06934 |
| 1 | 9 | 0.64870 | 0.06869 |
| 2 | 7 | 0.65241 | 0.06760 |
| **2** | **8** | **0.65568** | **0.06645** |
| 2 | 9 | 0.65124 | 0.07024 |
| 2 | 10 | 0.64809 | 0.06900 |
| 3 | 3 | 0.64426 | 0.06672 |
| 3 | 5 | 0.64308 | 0.06949 |
| 3 | 7 | 0.64765 | 0.06834 |
| 3 | 8 | 0.64930 | 0.07324 |
| 3 | 9 | 0.65180 | 0.06741 |
| 3 | 10 | 0.64971 | 0.07378 |
| 4 | 8 | 0.65040 | 0.06930 |
| 4 | 9 | 0.64641 | 0.06935 |
| 4 | 10 | 0.64856 | 0.06927 |
| 5 | 3 | 0.64745 | 0.06739 |
| 5 | 5 | 0.64670 | 0.06401 |
| 5 | 7 | 0.65150 | 0.07140 |
| 5 | 9 | 0.64858 | 0.06987 |

Then we have tuned $\gamma$:

| $\gamma$ | F1 avg | F1 std |
|---|---|---|
| **0** | **0.65568** | **0.06645** |
| 0.1 | 0.65004 | 0.07024 |
| 0.3 | 0.64542 | 0.06846 |
| 0.6 | 0.64410 | 0.07195 |
| 1.2 | 0.64894 | 0.07113 |
| 2.5 | 0.64741 | 0.07131 |
| 5.0 | 0.64514 | 0.07437 |
| 10.0 | 0.64657 | 0.08842 |
| 20.0 | 0.64366 | 0.09013 |

Using the found values, we recalibrated the number of trees to 272 and tuned the subsample and the colsample hyperparameters:

| Colsample | subsample | F1 avg | F1 std |
|---|---|---|---|
| 0.6 | 0.6 | 0.65124 | 0.07104 |
| 0.6 | 0.7 | 0.65014 | 0.06922 |
| 0.6 | 0.8 | 0.64427 | 0.06840 |
| 0.6 | 0.9 | 0.64665 | 0.06801 |
| 0.7 | 0.6 | 0.64558 | 0.07332 |
| 0.7 | 0.7 | 0.64497 | 0.06831 |
| 0.7 | 0.8 | 0.64757 | 0.07087 |
| 0.7 | 0.9 | 0.64822 | 0.06861 |
| **0.75** | **0.75** | **0.65578** | **0.06703** |
| 0.75 | 0.8 | 0.65460 | 0.06495 |
| 0.75 | 0.85 | 0.64954 | 0.06901 |
| 0.8 | 0.6 | 0.64587 | 0.06435 |
| 0.8 | 0.7 | 0.64890 | 0.06884 |
| 0.8 | 0.75 | 0.65578 | 0.06703 |
| 0.8 | 0.8 | 0.65460 | 0.06495 |
| 0.8 | 0.85 | 0.64954 | 0.06901 |
| 0.8 | 0.9 | 0.65057 | 0.07368 |
| 0.85 | 0.75 | 0.64915 | 0.06989 |
| 0.85 | 0.8 | 0.65413 | 0.06623 |
| 0.85 | 0.85 | 0.64945 | 0.06858 |
| 0.9 | 0.6 | 0.64897 | 0.06896 |
| 0.9 | 0.7 | 0.65211 | 0.06782 |
| 0.9 | 0.8 | 0.64881 | 0.06728 |
| 0.9 | 0.9 | 0.65109 | 0.06939 |

At last we tuned the $l_1$ and $l_2$ terms:

| $l_1$ | $l_2$ | F1 avg | F1 std |
|---|---|---|---|
| 0.00001 | 0.00001 | 0.64923 | 0.06815 |
| 0.01 | 0.00001 | 0.64904 | 0.07029 |
| 0.1, | 0.00001 | 0.64663 | 0.06853 |
| 10 | 0.00001 | 0.64777 | 0.07192 |
| 100 | 0.00001 | 0.64650 | 0.07173 |
| 1000 | 0.00001 | 0.57881 | 0.12107 |
| 0.00001 | 0.01 | 0.64801 | 0.06892 |
| 0.01 | 0.01 | 0.65072 | 0.06668 |
| 0.1 | 0.01 | 0.64571 | 0.06714 |
| 10 | 0.01 | 0.64619 | 0.07167 |
| 100 | 0.01 | 0.64691 | 0.07311 |
| 1000 | 0.01 | 0.57881 | 0.12107 |
| 0.00001 | 0.1 | 0.64813 | 0.06364 |
| 0.01 | 0.1 | 0.65111 | 0.06987 |
| 0.1 | 0.1 | 0.65227 | 0.06688 |
| 10 | 0.1 | 0.64817 | 0.07340 |
| 100 | 0.1 | 0.64853 | 0.07287 |
| 1000 | 0.1 | 0.57881 | 0.12107 |
| 0.00001 | 10 | 0.64641 | 0.07499 |
| 0.01 | 10 | 0.64857 | 0.07582 |
| 0.1 | 10 | 0.64633 | 0.07169 |
| 10 | 10 | 0.64745 | 0.07521 |
| 100 | 10 | 0.64947 | 0.07562 |
| 1000 | 10 | 0.57804 | 0.12172 |
| 0.00001 | 100 | 0.62461 | 0.10363 |
| 0.01 | 100 | 0.62461 | 0.10363 |
| 0.1 | 100 | 0.62504 | 0.10400 |
| 10 | 100 | 0.62389 | 0.10497 |
| 100 | 100 | 0.62347 | 0.10392 |
| 1000 | 100 | 0.56540 | 0.12516 |
| 0.00001 | 100000 | 0.08801 | 0.00005 |
| 0.01 | 100000 | 0.08801 | 0.00005 |
| 0.1 | 100000 | 0.08801 | 0.00005 |
| 10 | 100000 | 0.08801 | 0.00005 |
| 100 | 100000 | 0.08801 | 0.00005 |
| 1000 | 100000 | 0.08801 | 0.00005 |
| 0.05 | 0.05 | 0.64533 | 0.07003 |
| 0.1 | 0.05 | 0.64969 | 0.07097 |
| 0.5 | 0.05 | 0.64906 | 0.07099 |
| 0.05 | 0.1 | 0.65010 | 0.06762 |
| 0.1 | 0.1 | 0.65227 | 0.06688 |
| 0.5 | 0.1 | 0.64954 | 0.07248 |
| 0.05 | 0.5 | 0.65109 | 0.07082 |
| 0.1 | 0.5 | 0.65027 | 0.07239 |
| 0.5 | 0.5 | 0.64549 | 0.07156 |
| **0.07** | **0.07** | **0.65234** | **0.06813** |
| 0.1 | 0.07 | 0.64944 | 0.06658 |
| 0.2 | 0.07 | 0.65017 | 0.06671 |
| 0.07 | 0.1 | 0.64927 | 0.07059 |
| 0.1 | 0.1 | 0.65227 | 0.06688 |
| 0.2 | 0.1 | 0.65138 | 0.06737 |
| 0.07 | 0.2 | 0.64396 | 0.07147 |
| 0.1 | 0.2 | 0.64997 | 0.06681 |
| 0.2 | 0.2 | 0.65057 | 0.06585 |

Finally, we have lowered the learning rate to 0.05 and recalibrated the number of trees to 763 estimators.

## CWFID split 1 using just scale invariant features

Starting with a learning rate of 0.5 we have selected 290 trees, we have then tuned the depth and the minimum child weight:

| Max depth | min child w | F1 avg | F1 std |
|---|---|---|---|
| 1 | 3 | 0.64301 | 0.06722 |
| 1 | 5 | 0.64281 | 0.06746 |
| 1 | 7 | 0.64516 | 0.07104 |
| 1 | 9 | 0.64217 | 0.07531 |
| 2 | 9 | 0.64744 | 0.07204 |
| 2 | 11 | 0.64292 | 0.07018 |
| 2 | 13 | 0.64418 | 0.06953 |
| 3 | 3 | 0.63884 | 0.06813 |
| 3 | 5 | 0.64435 | 0.06581 |
| 3 | 7 | 0.64525 | 0.07036 |
| 3 | 8 | 0.64481 | 0.06911 |
| 3 | 9 | 0.64610 | 0.07009 |
| 3 | 10 | 0.64104 | 0.06876 |
| 3 | 11 | 0.64720 | 0.06772 |
| 3 | 13 | 0.64353 | 0.06959 |
| 4 | 7 | 0.64570 | 0.06758 |
| 4 | 8 | 0.64100 | 0.06976 |
| **4** | **9** | **0.64957** | **0.06672** |
| 4 | 10 | 0.64510 | 0.06931 |
| 4 | 11 | 0.64164 | 0.07269 |
| 4 | 13 | 0.64064 | 0.07277 |
| 5 | 3 | 0.63559 | 0.07063 |
| 5 | 5 | 0.63845 | 0.06802 |
| 5 | 7 | 0.64244 | 0.06814 |
| 5 | 8 | 0.64162 | 0.07178 |
| 5 | 9 | 0.64355 | 0.07042 |
| 5 | 10 | 0.64714 | 0.06926 |

Then we have tuned $\gamma$:

| $\gamma$ | F1 avg | F1 std |
|---|---|---|
| **0** | **0.64957** | **0.06672** |
| 0.1 | 0.64934 | 0.07000 |
| 0.3 | 0.64139 | 0.06993 |
| 0.6 | 0.64421 | 0.06922 |
| 1.2 | 0.63599 | 0.07513 |
| 2.5 | 0.64522 | 0.07245 |
| 5.0 | 0.63990 | 0.08091 |
| 10.0 | 0.64129 | 0.08442 |
| 20.0 | 0.63757 | 0.09299 |

Using the found values, we recalibrated the number of trees to 136 and tuned the subsample and colsample hyperparameters:

| Colsample | subsample | F1 avg | F1 std |
|---|---|---|---|
| 0.6 | 0.6 | 0.64116 | 0.06766 |
| 0.6 | 0.7 | 0.63554 | 0.07299 |
| 0.6 | 0.8 | 0.63720 | 0.07255 |
| 0.6 | 0.9 | 0.64034 | 0.06974 |
| 0.7 | 0.6 | 0.63893 | 0.07104 |
| 0.7 | 0.7 | 0.64227 | 0.07060 |
| 0.7 | 0.8 | 0.63986 | 0.07217 |
| 0.7 | 0.9 | 0.63854 | 0.07593 |
| 0.75 | 0.75 | 0.64116 | 0.07321 |
| 0.75 | 0.8 | 0.63986 | 0.07217 |
| 0.75 | 0.85 | 0.64147 | 0.07165 |
| 0.8 | 0.6 | 0.64238 | 0.07133 |
| 0.8 | 0.7 | 0.64435 | 0.06714 |
| 0.8 | 0.75 | 0.64226 | 0.07141 |
| **0.8** | **0.8** | **0.64905** | **0.06642** |
| 0.8 | 0.85 | 0.64104 | 0.07009 |
| 0.8 | 0.9 | 0.64270 | 0.64270 |
| 0.85 | 0.75 | 0.64307 | 0.06879 |
| 0.85 | 0.8 | 0.64277 | 0.07353 |
| 0.85 | 0.85 | 0.64216 | 0.06968 |
| 0.9 | 0.6 | 0.64042 | 0.07220 |
| 0.9 | 0.7 | 0.64330 | 0.07175 |
| 0.9 | 0.8 | 0.64277 | 0.07353 |
| 0.9 | 0.9 | 0.64502 | 0.07025 |

At last we have tuned the $l_1$ and $l_2$ terms:

| $l_1$ | $l_2$ | F1 avg | F1 std |
|---|---|---|---|
| 0.00001 | 0.00001 | 0.63958 | 0.07114 |
| 0.01 | 0.00001 | 0.64339 | 0.06984 |
| 0.1 | 0.00001 | 0.64428 | 0.07228 |
| 10 | 0.00001 | 0.64464 | 0.07126 |
| 100 | 0.00001 | 0.64299 | 0.07651 |
| 1000 | 0.00001 | 0.54692 | 0.13432 |
| 0.00001 | 0.01 | 0.63875 | 0.07111 |
| 0.01 | 0.01 | 0.64632 | 0.07175 |
| 0.1 | 0.01 | 0.64976 | 0.06870 |
| 10 | 0.01 | 0.64497 | 0.07201 |
| 100 | 0.01 | 0.64533 | 0.07707 |
| 1000 | 0.01 | 0.54692 | 0.13432 |
| 0.00001 | 0.1 | 0.64285 | 0.06840 |
| 0.01 | 0.1 | 0.64298 | 0.07132 |
| 0.1 | 0.1 | 0.64961 | 0.06961 |
| 10 | 0.1 | 0.64369 | 0.07323 |
| 100 | 0.1 | 0.64400 | 0.07545 |
| 1000 | 0.1 | 0.54692 | 0.13432 |
| 0.00001 | 10 | 0.64269 | 0.07307 |
| 0.01 | 10 | 0.64494 | 0.07364 |
| 0.1 | 10 | 0.64306 | 0.07608 |
| 10 | 10 | 0.64646 | 0.07296 |
| 100 | 10 | 0.64766 | 0.07512 |
| 1000 | 10 | 0.54585 | 0.13453 |
| 0.00001 | 100 | 0.62013 | 0.09983 |
| 0.01 | 100 | 0.62013 | 0.09983 |
| 0.1 | 100 | 0.62013 | 0.09983 |
| 10 | 100 | 0.62106 | 0.09766 |
| 100 | 100 | 0.62315 | 0.10211 |
| 1000 | 100 | 0.53372 | 0.13731 |
| 0.00001 | 1000 | 0.08801 | 0.00005 |
| 0.01 | 1000 | 0.08801 | 0.00005 |
| 0.1 | 1000 | 0.08801 | 0.00005 |
| 10 | 1000 | 0.08801 | 0.00005 |
| 100 | 1000 | 0.08801 | 0.00005 |
| 1000 | 1000 | 0.08801 | 0.00005 |
| 0.05 | 0.001 | 0.64439 | 0.06985 |
| 0.1 | 0.001 | 0.64663 | 0.07292 |
| 0.5 | 0.001 | 0.64263 | 0.07024 |
| 0.05 | 0.01 | 0.64074 | 0.07092 |
| 0.1 | 0.01 | 0.64976 | 0.06870 |
| 0.5 | 0.01 | 0.64338 | 0.06968 |
| 0.05 | 0.05 | 0.64316 | 0.07121 |
| 0.1 | 0.05 | 0.64364 | 0.07422 |
| 0.5 | 0.05 | 0.64080 | 0.07050 |
| 0.07 | 0.005 | 0.64610 | 0.07196 |
| 0.1 | 0.005 | 0.64352 | 0.06807 |
| 0.2 | 0.005 | 0.64468 | 0.07201 |
| 0.07 | 0.01 | 0.64500 | 0.06995 |
| **0.1** | **0.01** | **0.64976** | **0.06870** |
| 0.2 | 0.01 | 0.64295 | 0.06983 |
| 0.07 | 0.02 | 0.64590 | 0.07056 |
| 0.1 | 0.02 | 0.64493 | 0.06541 |
| 0.2 | 0.02 | 0.64340 | 0.07267 |

Finally, we have lowered the learning rate to 0.05 and recalibrated the number of trees to 809 estimators.

## CWFID split 2 using just scale invariant features

Starting with a learning rate of 0.5, we have selected 412 trees, we have then tuned the depth and the minimum child weight:

| Max depth | min child w | F1 avg | F1 std |
|---|---|---|---|
| 3 | 1 | 0.66304 | 0.05306 |
| 3 | 3 | 0.66228 | 0.05655 |
| 3 | 5 | 0.66214 | 0.05584 |
| 5 | 1 | 0.65925 | 0.05238 |
| 5 | 3 | 0.65693 | 0.05272 |
| 5 | 5 | 0.65799 | 0.05356 |
| 7 | 1 | 0.65922 | 0.05374 |
| 7 | 3 | 0.65953 | 0.05520 |
| 7 | 5 | 0.65802 | 0.05227 |
| 9 | 1 | 0.66219 | 0.05195 |
| 9 | 3 | 0.66065 | 0.05291 |
| 9 | 5 | 0.66065 | 0.05154 |
| 9 | 9 | 0.65971 | 0.05110 |
| **11** | **1** | **0.66370** | **0.05530** |
| 11 | 5 | 0.66194 | 0.05384 |
| 11 | 9 | 0.65894 | 0.05338 |
| 13 | 1 | 0.66157 | 0.05554 |
| 13 | 5 | 0.66366 | 0.05284 |
| 13 | 9 | 0.66011 | 0.05213 |

Then we have tuned $\gamma$:

| $\gamma$ | F1 avg | F1 std |
|---|---|---|
| 0 | 0.66370 | 0.05530 |
| 0.1 | 0.65785 | 0.05308 |
| 0.3 | 0.66126 | 0.05435 |
| 0.6 | 0.66019 | 0.05428 |
| 1.2 | 0.65960 | 0.05427 |
| 2.5 | 0.66133 | 0.05328 |
| 5.0 | 0.66674 | 0.05347 |
| 10.0 | 0.66872 | 0.05802 |
| **20.0** | **0.67255** | **0.05512** |
| 30.0 | 0.67240 | 0.05984 |
| 50.0 | 0.66149 | 0.06154 |
| 70.0 | 0.65226 | 0.06326 |

Using the found values, we recalibrated the number of trees to 150 and tuned the subsample and colsample hyperparameters:

| Colsample | subsample | F1 avg | F1 std |
|---|---|---|---|
| 0.6 | 0.6 | 0.66903 | 0.05882 |
| 0.6 | 0.7 | 0.66963 | 0.05799 |
| 0.6 | 0.8 | 0.67122 | 0.05941 |
| 0.6 | 0.9 | 0.66900 | 0.05998 |
| 0.7 | 0.6 | 0.66703 | 0.05641 |
| 0.7 | 0.7 | 0.67187 | 0.05556 |
| 0.7 | 0.8 | 0.67221 | 0.05782 |
| 0.7 | 0.9 | 0.66896 | 0.05678 |
| 0.8 | 0.6 | 0.66997 | 0.05692 |
| **0.8** | **0.7** | **0.67318** | **0.05591** |
| 0.8 | 0.8 | 0.67159 | 0.05627 |
| 0.8 | 0.9 | 0.67170 | 0.05738 |
| 0.9 | 0.6 | 0.67301 | 0.05741 |
| 0.9 | 0.7 | 0.67057 | 0.05455 |
| 0.9 | 0.8 | 0.67216 | 0.05798 |
| 0.9 | 0.9 | 0.67048 | 0.05493 |

At last we have tuned the $l_1$ and $l_2$ terms:

| $l_1$ | $l_2$ | F1 avg | F1 std |
|---|---|---|---|
| 0.00001 | 0.00001 | 0.66898 | 0.05565 |
| 0.01 | 0.00001 | 0.67083 | 0.05541 |
| 0.1 | 0.00001 | 0.66985 | 0.05985 |
| 10 | 0.00001 | 0.67173 | 0.05889 |
| 100 | 0.00001 | 0.66632 | 0.05964 |
| 100000 | 0.00001 | 0.27002 | 0.00001 |
| 0.00001 | 0.01 | 0.66851 | 0.05773 |
| 0.01 | 0.01 | 0.66938 | 0.05820 |
| 0.1 | 0.01 | 0.66824 | 0.05802 |
| **10** | **0.01** | **0.67193** | **0.05934** |
| 100 | 0.01 | 0.66632 | 0.05964 |
| 100000 | 0.01 | 0.27002 | 0.00001 |
| 0.00001 | 0.1 | 0.66660 | 0.05656 |
| 0.01 | 0.1 | 0.66681 | 0.05911 |
| 0.1 | 0.1 | 0.66723 | 0.05574 |
| 10 | 0.1 | 0.67158 | 0.05899 |
| 100 | 0.1 | 0.66631 | 0.05988 |
| 100000 | 0.1 | 0.27002 | 0.00001 |
| 0.00001 | 10 | 0.66933 | 0.05669 |
| 0.01 | 10 | 0.66867 | 0.05779 |
| 0.1 | 10 | 0.66623 | 0.05845 |
| 10 | 10 | 0.66900 | 0.06098 |
| 100 | 10 | 0.66525 | 0.05881 |
| 100000 | 10 | 0.27002 | 0.00001 |
| 0.00001 | 100 | 0.64934 | 0.05517 |
| 0.01 | 100 | 0.64934 | 0.05517 |
| 0.1 | 100 | 0.64934 | 0.05517 |
| 10 | 100 | 0.65049 | 0.05629 |
| 100 | 100 | 0.65083 | 0.05653 |
| 100000 | 100 | 0.27002 | 0.00001 |
| 0.00001 | 100000 | 0.07951 | 0.00001 |
| 0.01 | 100000 | 0.07951 | 0.00001 |
| 0.1 | 100000 | 0.07951 | 0.00001 |
| 10 | 100000 | 0.07951 | 0.00001 |
| 100 | 100000 | 0.07951 | 0.00001 |
| 100000 | 100000 | 0.07951 | 0.00001 |

Finally, we have lowered the learning rate to 0.05 and recalibrated the number of trees to 761 estimators.

## Broccoli using just scale invariant features

Starting with a learning rate of 0.8, we have selected 140 trees, we have then tuned the depth and the minimum child weight:

| Max depth | min child w | F1 avg | F1 std |
|---|---|---|---|
| 1 | 2 | 0.87184 | 0.00936 |
| 1 | 3 | 0.87184 | 0.00936 |
| **1** | **4** | **0.87186** | **0.00936** |
| 1 | 5 | 0.87186 | 0.00936 |
| 1 | 7 | 0.87186 | 0.00936 |
| 2 | 2 | 0.87080 | 0.01015 |
| 2 | 3 | 0.87120 | 0.01001 |
| 2 | 4 | 0.87083 | 0.00991 |
| 3 | 1 | 0.86874 | 0.00986 |
| 3 | 3 | 0.86880 | 0.00977 |
| 3 | 4 | 0.86892 | 0.00931 |
| 3 | 5 | 0.86936 | 0.00961 |
| 3 | 7 | 0.86907 | 0.00983 |
| 4 | 4 | 0.86679 | 0.00927 |
| 4 | 5 | 0.86607 | 0.00962 |
| 4 | 7 | 0.86672 | 0.00980 |
| 5 | 1 | 0.86244 | 0.00903 |
| 5 | 3 | 0.86257 | 0.00911 |
| 5 | 5 | 0.86335 | 0.00988 |
| 7 | 1 | 0.85713 | 0.00904 |
| 7 | 3 | 0.85657 | 0.00827 |
| 7 | 5 | 0.85766 | 0.00908 |
| 9 | 1 | 0.85297 | 0.00889 |
| 9 | 3 | 0.85333 | 0.00887 |
| 9 | 5 | 0.85324 | 0.00775 |

Then we have tuned $\gamma$:

| $\gamma$ | Average F1 score | F1 score std |
|---|---|---|
| **0** | **0.87186** | **0.00936** |
| 0.1 | 0.87186 | 0.00936 |
| 0.3 | 0.87186 | 0.00936 |
| 0.6 | 0.87186 | 0.00936 |
| 1.2 | 0.87186 | 0.00936 |
| 2.5 | 0.87186 | 0.00936 |
| 5.0 | 0.87186 | 0.00936 |
| 10.0 | 0.87182 | 0.00959 |
| 20.0 | 0.87201 | 0.00905 |
| 30.0 | 0.87199 | 0.00935 |
| 50.0 | 0.87198 | 0.00975 |
| 70.0 | 0.87174 | 0.00923 |

Using the found values, we recalibrated the number of trees to 166 and tuned the subsample and colsample hyperparameters:

| Colsample | subsample | F1 avg | F1 std |
|---|---|---|---|
| 0.6 | 0.6 | 0.87194 | 0.00987 |
| 0.6 | 0.7 | 0.87153 | 0.00976 |
| 0.6 | 0.8 | 0.87170 | 0.00975 |
| 0.6 | 0.9 | 0.87212 | 0.00988 |
| 0.7 | 0.6 | 0.87192 | 0.00953 |
| 0.7 | 0.7 | 0.87178 | 0.01013 |
| 0.7 | 0.8 | 0.87197 | 0.00968 |
| 0.7 | 0.9 | 0.87196 | 0.01013 |
| 0.75 | 0.85 | 0.87187 | 0.00992 |
| 0.75 | 0.9 | 0.87196 | 0.01013 |
| 0.75 | 0.95 | 0.87202 | 0.00960 |
| 0.8 | 0.6 | 0.87172 | 0.00983 |
| 0.8 | 0.7 | 0.87168 | 0.00988 |
| 0.8 | 0.8 | 0.87177 | 0.01030 |
| **0.8** | **0.85** | **0.87223** | **0.01021** |
| 0.8 | 0.9 | 0.87222 | 0.00983 |
| 0.8 | 0.95 | 0.87190 | 0.00985 |
| 0.85 | 0.85 | 0.87189 | 0.01025 |
| 0.85 | 0.9 | 0.87203 | 0.00982 |
| 0.85 | 0.95 | 0.87169 | 0.00978 |
| 0.9 | 0.6 | 0.87190 | 0.00974 |
| 0.9 | 0.7 | 0.87181 | 0.01030 |
| 0.9 | 0.8 | 0.87182 | 0.01000 |
| 0.9 | 0.9 | 0.87203 | 0.00982 |

At last we have tuned the $l_1$ and $l_2$ terms:

| $l_1$ | $l_2$ | F1 avg | F1 std |
|---|---|---|---|
| 0.00001 | 0.00001 | 0.87212 | 0.01025 |
| 0.01 | 0.00001 | 0.87193 | 0.01022 |
| 0.1 | 0.00001 | 0.87225 | 0.01030 |
| 10 | 0.00001 | 0.87193 | 0.01020 |
| 100 | 0.00001 | 0.87251 | 0.00968 |
| 1000 | 0.00001 | 0.87221 | 0.00914 |
| 0.00001 | 0.01 | 0.87205 | 0.00997 |
| 0.01 | 0.01 | 0.87192 | 0.01023 |
| 0.1 | 0.01 | 0.87213 | 0.01025 |
| 10 | 0.01 | 0.87187 | 0.01013 |
| 100 | 0.01 | 0.87230 | 0.00947 |
| 1000 | 0.01 | 0.87221 | 0.00914 |
| 0.00001 | 0.1 | 0.87176 | 0.01038 |
| 0.01 | 0.1 | 0.87182 | 0.01048 |
| 0.1 | 0.1 | 0.87226 | 0.00998 |
| 10 | 0.1 | 0.87236 | 0.01039 |
| 100 | 0.1 | 0.87236 | 0.00966 |
| 1000 | 0.1 | 0.87229 | 0.00917 |
| 0.00001 | 10 | 0.87231 | 0.00985 |
| 0.01 | 10 | 0.87231 | 0.00985 |
| 0.1 | 10 | 0.87231 | 0.00976 |
| 10 | 10 | 0.87227 | 0.00982 |
| 100 | 10 | 0.87228 | 0.00956 |
| 1000 | 10 | 0.87233 | 0.00919 |
| 0.00001 | 100 | 0.87244 | 0.00939 |
| 0.01 | 100 | 0.87244 | 0.00939 |
| 0.1 | 100 | 0.87237 | 0.00937 |
| 10 | 100 | 0.87252 | 0.00935 |
| 100 | 100 | 0.87275 | 0.00961 |
| 1000 | 100 | 0.87232 | 0.00923 |
| 0.00001 | 1000 | 0.43929 | 0.00000 |
| 0.01 | 1000 | 0.43929 | 0.00000 |
| 0.1 | 1000 | 0.43929 | 0.00000 |
| 10 | 1000 | 0.43929 | 0.00000 |
| 100 | 1000 | 0.43929 | 0.00000 |
| 1000 | 1000 | 0.43929 | 0.00000 |
| 50 | 50 | 0.87246 | 0.00955 |
| 100 | 50 | 0.87251 | 0.00962 |
| 500 | 50 | 0.87272 | 0.00960 |
| 50 | 100 | 0.87252 | 0.00945 |
| 100 | 100 | 0.87275 | 0.00961 |
| **500** | **100** | **0.87276** | **0.00961** |
| 50 | 500 | 0.87267 | 0.00927 |
| 100 | 500 | 0.87271 | 0.00965 |
| 500 | 500 | 0.87276 | 0.00946 |

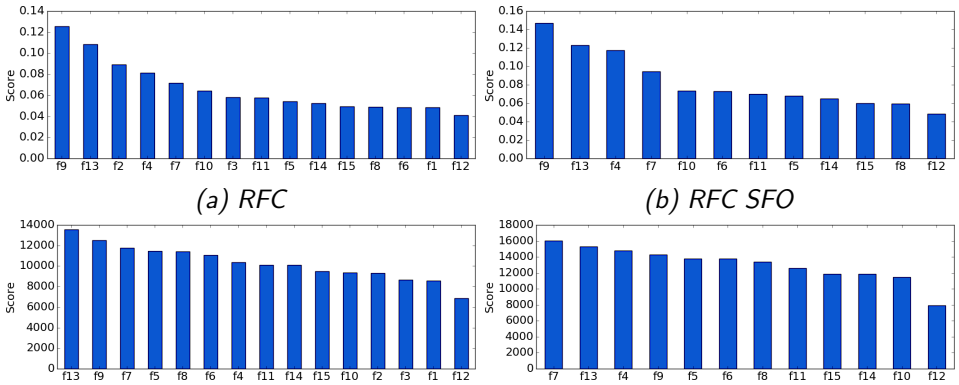
Finally, we have lowered the learning rate to 0.05 and recalibrated the number of trees to 819 estimators.

### Features importance

From the training of the random forests and the gradient boosted trees we have derived the importance of the various features.

As visible[2] in Figure 4.3, the removal of the scale dependent features is justified by their marginal importance. The only relevant feature in fact is $f_2$, which is the area of the region identified as vegetation, but, since this value largely depends on the position of the keypoints, it can be hypothesized that the classifiers are using this feature with a low threshold to easily remove the remaining soil. Conversely, when also considering Figure 4.4 it can be seen how the features $f_9$ and $f_{13}$ are really important for all the classifiers. Interestingly, feature $f_{13}$ becomes fundamental for the Broccoli dataset, as visible in Figure 4.5. Being both those features based on the NDVI value, it could be interesting to see whether a different index could lead to better results. Another really important feature is $f_4$, which is the *compactness* (area / perimeter$^2$) of the tile and thus represent a morphological characteristic.

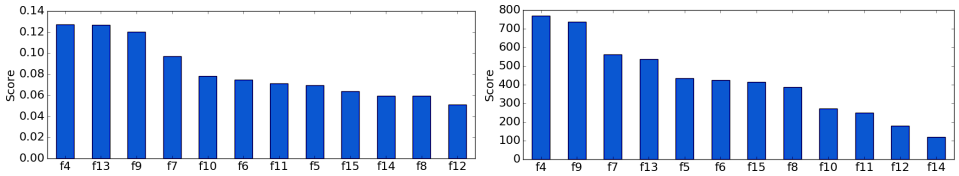
[2]We remind that a list of the features is reported in Table 3.1 on page 33




*(a) RFC* *(b) RFC SFO*

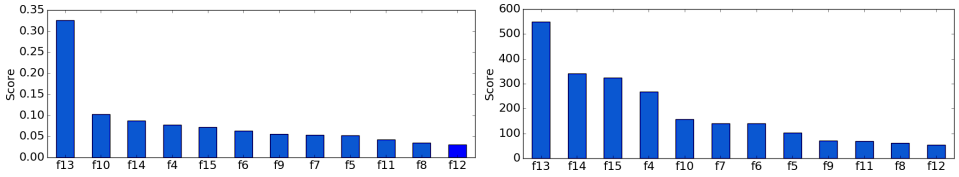
*(c) XGB* *(d) XGB SFO*

*Figure 4.3: Features importance on CWFID split 1*




*(a) RFC SFO* *(b) XGB SFO*

*Figure 4.4: Features importance on CWFID split 2*




*(a) RFC SFO* *(b) XGB SFO*

*Figure 4.5: Features importance on the Broccoli dataset*

### 4.2.3 Neural networks

When training the neural networks we have followed a common approach so to better highlight their differences. Using the training and the validation datasets we have tuned the hyperparameters measuring the log loss and the accuracy. Since we are more interested in having networks that are accurate on the vegetation we also measured the two metrics on the masked vegetation.

The approach we used is the following[3]:

1. Putting all the hyperparameters to zero, we did a single epoch to select a good learning rate. A single epoch is enough to exclude the ranges that lead to a gradient explosion or that are too small and thus do not substantially modify the parameters.

2. Resetting all the parameters of the network, we did another training epoch to select the $l_2$ term, selecting a value bringing to a difference between the loss on the validation and the loss on the training sets without degrading too much the performance.

3. We then performed 3 epochs of training for 20 times, resetting the parameters each time. All the hyperparameters, except for the learning rate and the $l_2$ term, have been sampled randomly on a coarse scale. We then selected some good ranges for all the hyperparameters looking for low validation losses in conjunction with higher training losses.

4. We repeated the last step but for 10 epochs around the found ranges, selecting the set of hyperparameters bringing to the lowest validation loss.

5. We then actually trained the networks until convergence was visually reached, which usually took around 200 epochs, selecting the epoch with the best result on validation as our final model.

The results of the hyperparameters selection are reported in Appendix A. Before presenting the graphs of the training processes, we report a legend explaining the meaning of the various colors in Figure 4.6 and an explanation of the various hyperparameters:

- LR: the learning rate:
- $l_2$: the weight decay multiplier;

[3]The training methodology has been adapted from the material of Stanford's CS231n course [7].

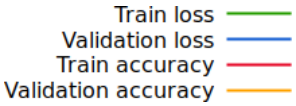



*Figure 4.6: Training graphs legend*

- GS: the $\sigma$ value for the gaussian noise layer of the network;
- OD: dropout value for output layer;
- D1D: dropout value for the first dense layer;
- D2D: dropout value for the second dense layer;
- EL_S: $\sigma$ value for the elastic deformation;
- EL_A: $\alpha$ value for the elastic deformation;
- AR: absolute maximum random value added to all channels;
- SR: maximum value for the random shift.

### Sliding window convolutional neural network

The first network resulted being very sensible to the hyperparameters selection. In fact, we had to train it three times on the second split of CWFID in order to obtain good results.

With the hyperparameters reported in Table 4.14 we have trained the sliding window network, obtaining the results depicted in Figure 4.7. From the observation of the graphs, it can be easily seen the extreme variability exhibited by this network. In fact, on the first split of CWFID the network reaches a good validation accuracy, even if the loss grows dramatically. However, the high instability of the network to hyperparameters variations can be best seen on the second split of CWFID: in the first two attempts, in fact, the network is not learning at all.

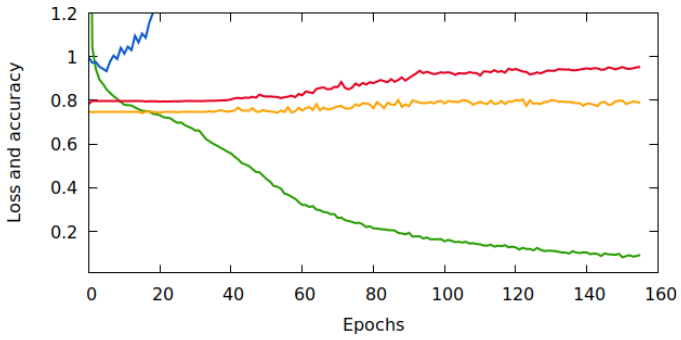
The Broccoli dataset was less problematic, as the network is able to clearly reach an optimum before starting to get worse. It can also be noticed that, albeit the validation loss explodes, the validation accuracy remains high. This could be linked to the fact that the dataset may contain some windows whose content is, in the vast majority, different from the central pixel, which is used to annotate the window. Therefore the network effectively learns to correctly classify those windows, but, since their annotation is wrong, the loss starts to get worse and worse since it is not limited, whereas the

*Table 4.14: Hyperparameters for the sliding window network*

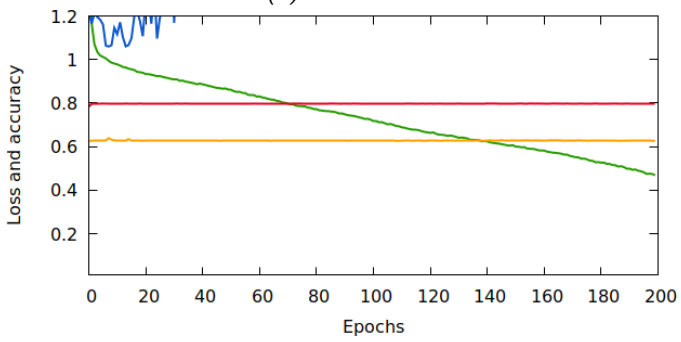
| Dataset | LR | $l_2$ | GS | OD | D1D | D2D | EL_S | EL_A | AR |
|---|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.00050000 | 0.00001000 | 0.06 | 0.76 | 0.18 | 0.37 | 10.99 | 53.80 | 0.48 |
| CWFID2_1 | 0.00002000 | 0.00010000 | 0.12 | 0.42 | 0.44 | 0.47 | 46.82 | 87.44 | 0.74 |
| CWFID2_2 | 0.00046017 | 0.00077041 | 0.28 | 0.75 | 0.46 | 0.18 | 99.08 | 3.02 | 0.85 |
| CWFID2_3 | 0.00002142 | 0.00027067 | 0.00 | 0.40 | 0.37 | 0.83 | 25.76 | 62.10 | 0.10 |
| Broccoli | 0.00002000 | 0.00010000 | 0.03 | 0.57 | 0.74 | 0.19 | 42.21 | 119.33 | 0.12 |

accuracy is bounded. However, for the second CWFID split, the network reaches a good performance just one time on three trials, even when just considering the validation accuracy.

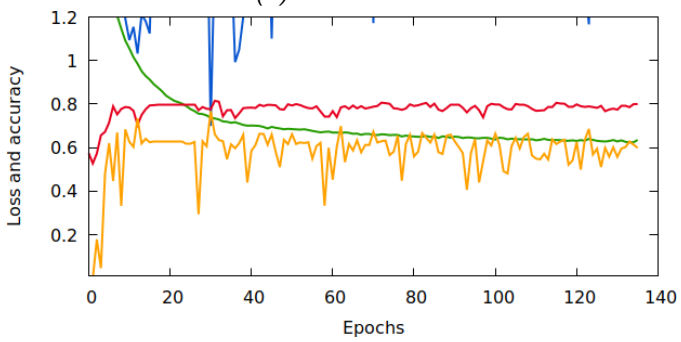
Moreover, this network is working with overlapping tiles, therefore the dataset has a really big dimension in memory. This problem made it impossible for us to train the network on the Broccoli dataset with the depth channel, since the memory required to load the dataset was not available. A solution could be to avoid keeping in memory the entire dataset, but this would slow down the training. The computational cost on the Broccoli dataset and the really high hyperparameters dependence are the main drawbacks of this network.



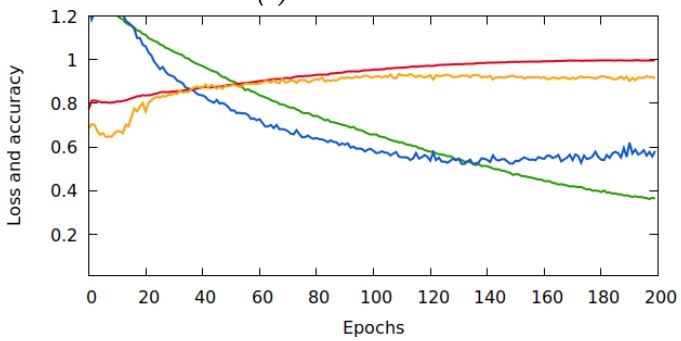

*(a) CWFID 1*



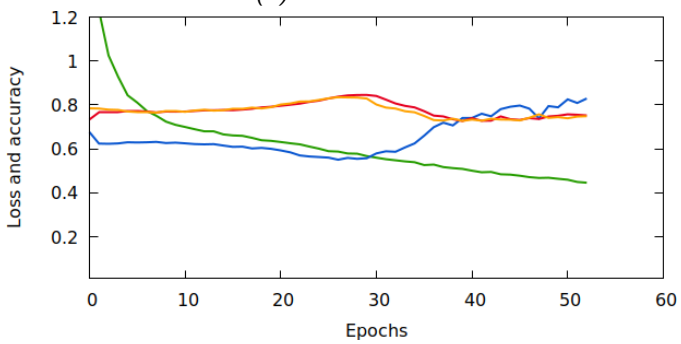

*(b) CWFID 2 1*




*(c) CWFID 2 2*




*(d) CWFID 2 3*




*(e) Broccoli*



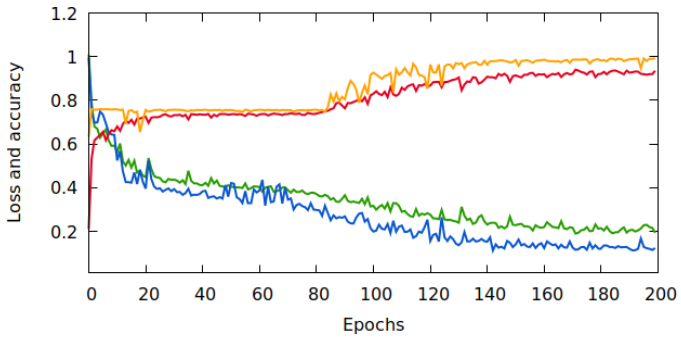
*Figure 4.7: Training of sliding window network*

*Table 4.15: Hyperparameters for U-Net*

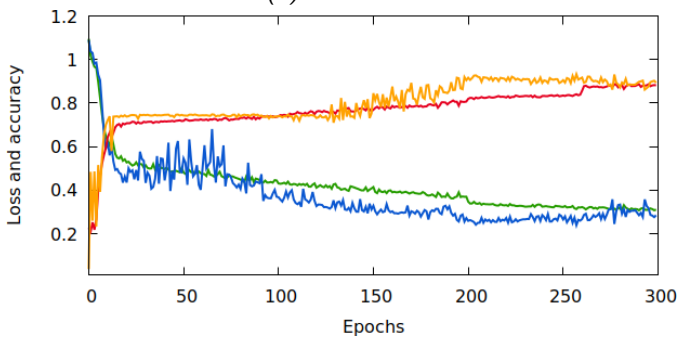
| Dataset | LR | L2 | GS | OD | EL_S | EL_A | AR | SR |
|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.00050000 | 0.00000001 | 0.54 | 0.45 | 15,67 | 47 | 0.08 | 0 |
| CWFID2 1 | 0.00020000 | 0.00001000 | 0.64 | 0.74 | 13,94 | 84,92 | 0.55 | 16 |
| CWFID2 2 | 0.00020000 | 0.00001000 | 0.04 | 0.80 | 19,76 | 87 | 0.37 | 57 |
| Broccoli | 0.00020000 | 0.00010000 | 0.23 | 0.76 | 12,54 | 34 | 0.60 | 47 |
| BroccoliD | 0.00050000 | 0.01000000 | 0.58 | 0.59 | 11,78 | 65,52 | 0.11 | 78 |

### U-Net network

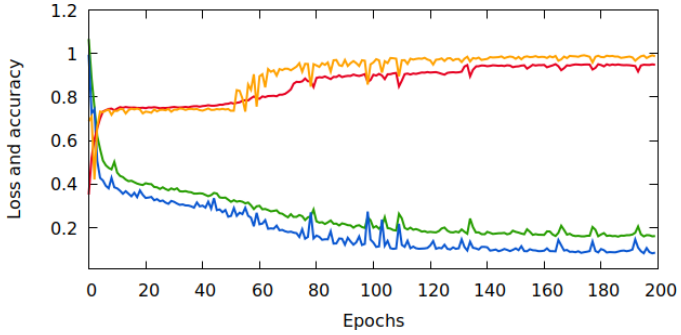
Using the hyperparameters reported in Table 4.15 we have trained the network as detailed in Figure 4.8. Like for the previous network, it can be seen that the Broccoli dataset is the easiest case, while the second split of CWFID is the most problematic. In fact, even this network required to be trained a second time in order to reach an optimum. However, the difference in the two results here is not as dramatic as it was for the previous network. Moreover, it is visible how on the second split of CWFID the network's accuracy reaches a plateau before jumping rapidly on a higher level (around epoch 250 in Figure 4.8b and epoch 70 for Figure 4.8c). This is a peculiarity that will be also present in the other networks, and could be indicating that the U-shaped architecture reacts slowly to the creation of a really useful implicit feature as the information has to be backpropagated through an high number of layers.



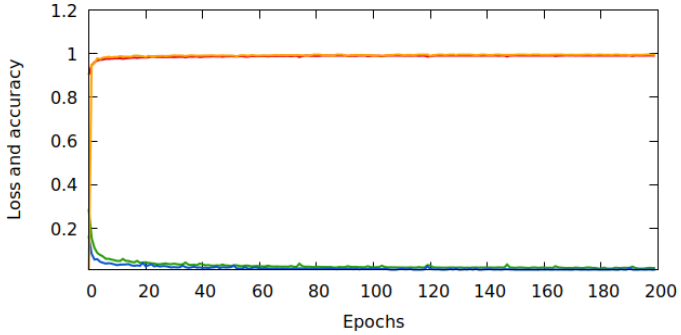

*(a) CWFID 1*



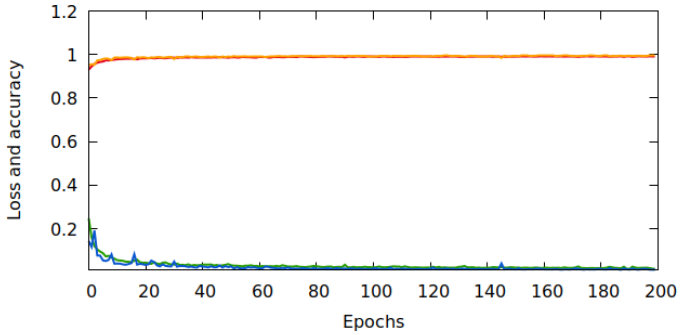

*(b) CWFID 2 1*




*(c) CWFID 2 2*




*(d) Broccoli*




*(e) Broccoli Depth*



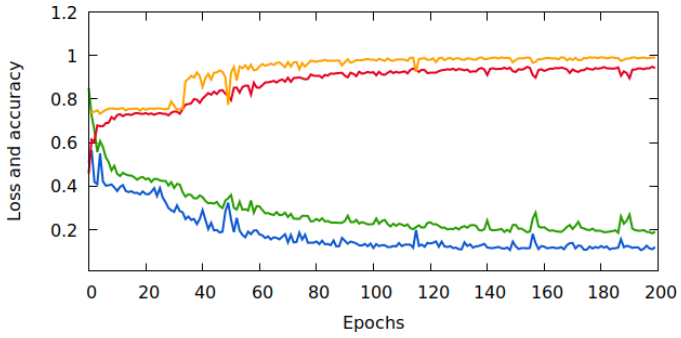
*Figure 4.8: Training of U-Net*

*Table 4.16: Hyperparameters for U-ReNet*

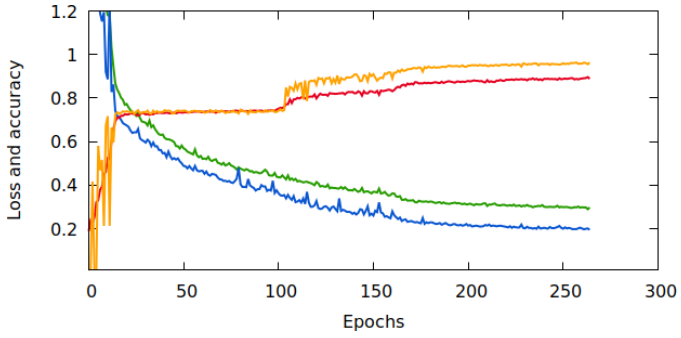
| Dataset | LR | L2 | GS | OD | EL_S | EL_A | AR | SR |
|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.0005 | 0.0000001 | 0.72 | 0.55 | 15,53 | 43 | 0.14 | 49 |
| CWFID2 | 0.0002 | 0.0001 | 0.39 | 0.75 | 14,73 | 75,55 | 0.78 | 88 |
| Broccoli | 0.0002 | 0.001 | 0.40 | 0.52 | 49,88 | 20,26 | 0.88 | 57 |
| BroccoliD | 0.0005 | 0.00000001 | 0.56 | 0.31 | 59 | 14 | 0.57 | 86 |

### U-ReNet network

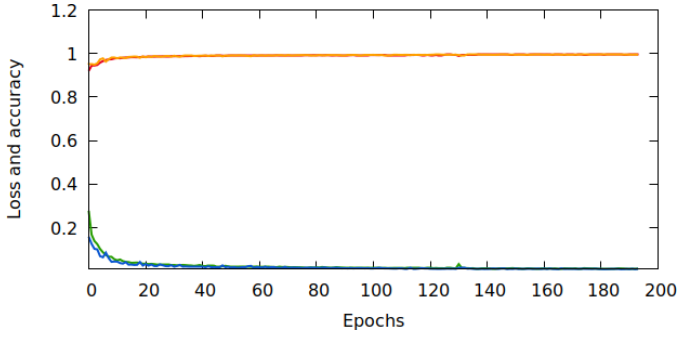
Using the hyperparameters reported in Table 4.16 we have trained the network as detailed in Figure 4.9. By visually comparing this graphs to the U-Net ones we cannot appreciate significant differences. Interestingly, the plateau phase is still present. Differently from U-Net, this network reached good performance at the first training. This suggest a better resilience to hyperparameters variations, but we will better investigate this in later sections. Since the validation accuracies are not substantially different from the ones obtained with U-Net, we can hypothesize that, in our problems, the annotation of a single pixel mostly depends on its neighbors and thus we are not able to take advantage from the high distance dependences found by the ReNet layers.



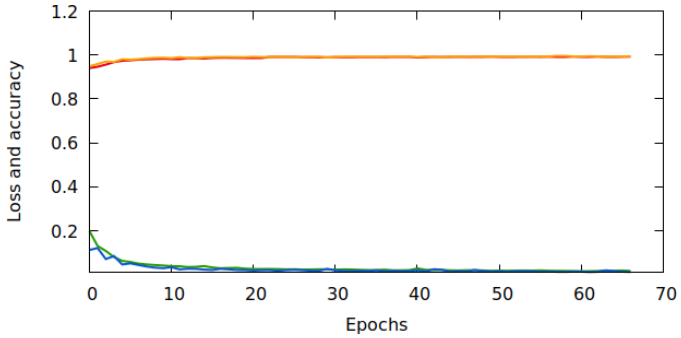

*(a) CWFID 1*




*(b) CWFID 2*




*(c) Broccoli*




*(d) Broccoli Depth*

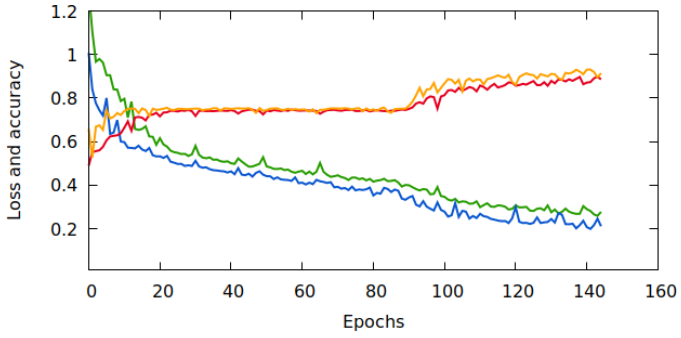
*Figure 4.9: Training of U-ReNet*

*Table 4.17: Hyperparameters for U-ReNet2*

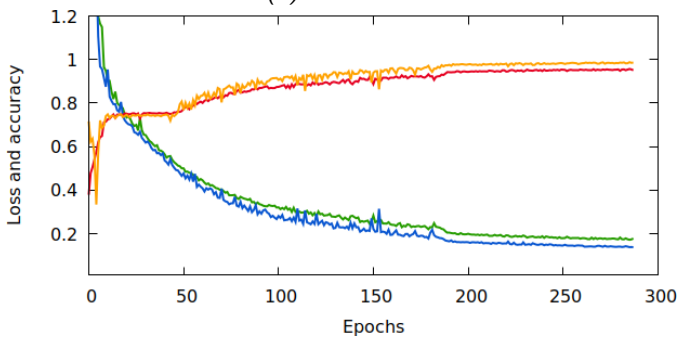
| Dataset | LR | L2 | GS | OD | EL_S | EL_A | AR | SR |
|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.0005 | 0.0000001 | 0.56 | 0.59 | 11,89 | 48,19 | 0.10 | 65 |
| CWFID2 | 0.0002 | 0.0001 | 0.12 | 0.14 | 19,52 | 55,630 | 0.45 | 77 |
| Broccoli | 0.0002 | 0.0001 | 0.31 | 0.40 | 28,89 | 22,54 | 0.03 | 57 |
| BroccoliD | 0.0005 | 0.00000001 | 0.20 | 0.52 | 11,75 | 59,89 | 0.53 | 98 |

### U-ReNet2 network

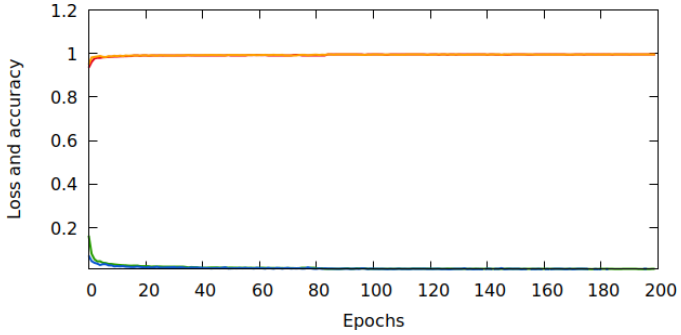
Reminding that this network is equal to the last one except for the number of hidden units, we report in Figure 4.10 the graphs of the training obtained with the hyperparameters depicted in Table 4.17. The results are not substantially different, suggesting that the number of hidden units is not a critical hyperparameter for the network performance. The main difference between U-ReNet and U-ReNet2 resides in the time needed to train the networks, as will be examined in a later section.



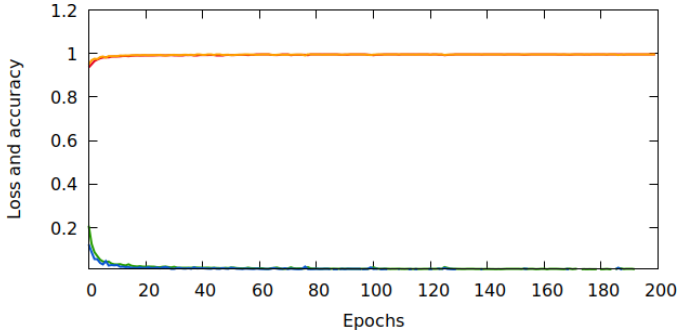

*(a) CWFID 1*




*(b) CWFID 2*




*(c) Broccoli*




*(d) Broccoli Depth*

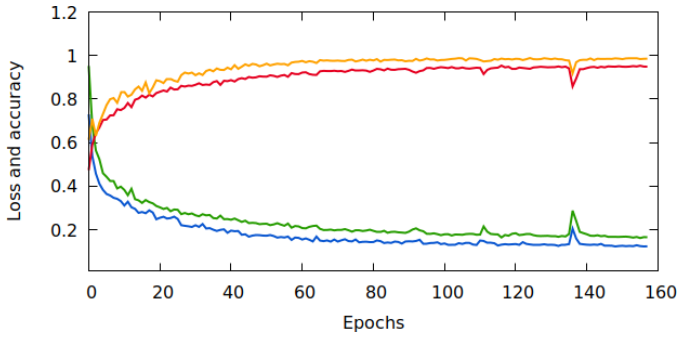
*Figure 4.10: Training of ReNet2*

*Table 4.18: Hyperparameters for reseg*

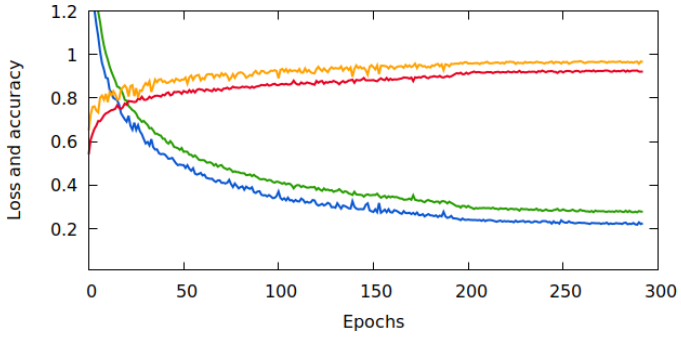
| Dataset | LR | L2 | GS | OD | EL_S | EL_A | AR | SR |
|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.0005 | 0.000001 | 0.28 | 0.27 | 15,77 | 84 | 0.21 | 58 |
| CWFID2 | 0.0002 | 0.0001 | 0.58 | 0.42 | 11,75 | 60,60 | 0.15 | 16 |
| Broccoli | 0.0002 | 0.0001 | 0.49 | 0.75 | 15,52 | 77,50 | 0.10 | 46 |
| BroccoliD | 0.0005 | 0.0000001 | 0.27 | 0.70 | 16,26 | 18,65 | 0.02 | 50 |

### ReSeg network

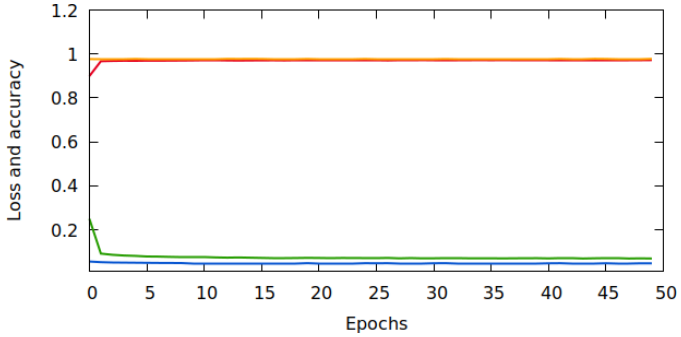
Using the hyperparameters reported in Table 4.18 we have trained this network as detailed in Figure 4.11. Even for this network we see that the Broccoli dataset is the easiest case. Interestingly, the plateau phase is not present, which corroborate our hypothesis that the plateau arises from the U-shaped structure of U-Net and U-ReNet. The main difference between U-ReNet and ReSeg is in the upsampling strategy, which is more complex in U-ReNet. The validation accuracies are not substantially different between the two architectures, therefore we will later analyze better the differences.



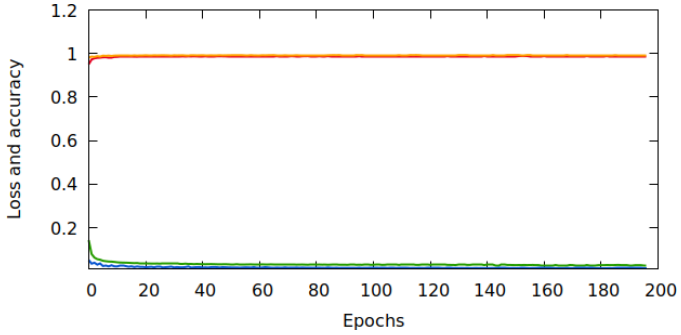

*(a) CWFID 1*




*(b) CWFID 2*




*(c) Broccoli*




*(d) Broccoli Depth*

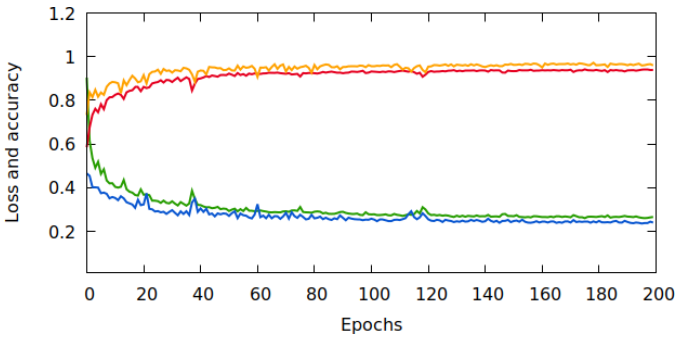
*Figure 4.11: Training of ReSeg*

*Table 4.19: Hyperparameters for ReConv*

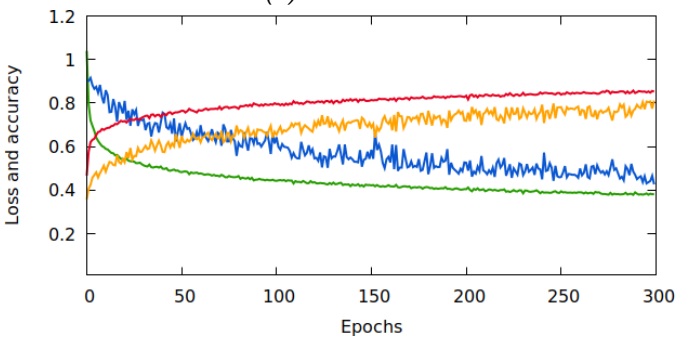
| Dataset | LR | L2 | GS | OD | EL_S | EL_A | AR | SR |
|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.0005 | 0.00000001 | 0.29 | 0.72 | 11,83 | 13,66 | 0.11 | 50 |
| CWFID2 1 | 0.00002 | 0.000001 | 0.57 | 0.71 | 20,67 | 94 | 0.33 | 71 |
| CWFID2 2 | 0.0005 | 0.00000001 | 0.77 | 0.53 | 12,60 | 79,51 | 0.01 | 49 |
| Broccoli | 0.0002 | 0.001 | 0.49 | 0.70 | 43,83 | 42,67 | 0.43 | 95 |
| BroccoliD | 0.0005 | 0.00001 | 0.37 | 0.27 | 16,20 | 45,34 | 0.28 | 82 |

### ReConv network

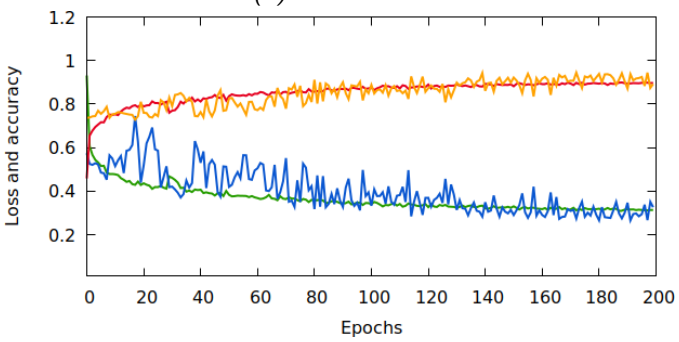
Using the hyperparameters of Table 4.19 we have trained the network obtaining the results of Figure 4.12. We remind that this network works like ReSeg but it receives as input the image as preprocessed by the first 3 layers of U-Net. From the validation datasets we can already see that this configuration is not as good as the previous ones, a sign that the features extracted by the convolutional layers are not good as an input to the ReSeg network. This can probably be explained by the fact that, differently from what happens in [53], our convolutional layers come from an U-shaped network and therefore may not contain sufficient information to reconstruct the image. In other words, since U-Net can retrieve the shape of the image using the concatenations from the contracting to the expanding path, it is possible that the convolutions just learn to extract some implicit features, therefore loosing the ability to reconstruct the original image as needed for image segmentation. On the other hand, this could also be due to the factorization of the convolutions, which therefore demands further analysis.



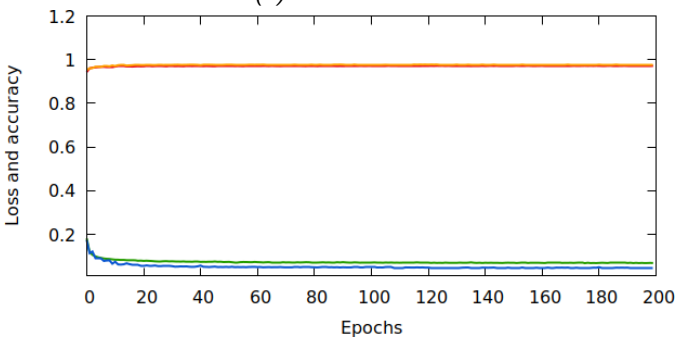

*(a) CWFID 1*



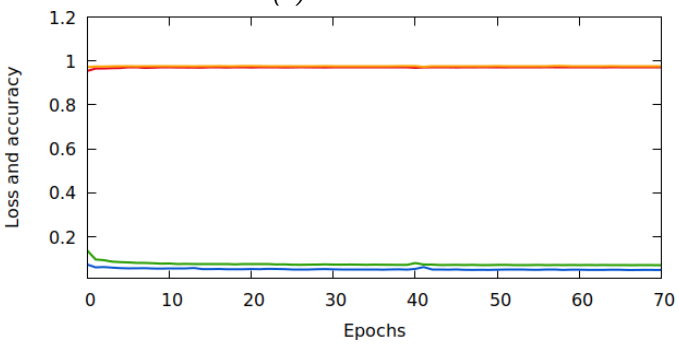

*(b) CWFID 2 1*




*(c) CWFID 2 2*




*(d) Broccoli*



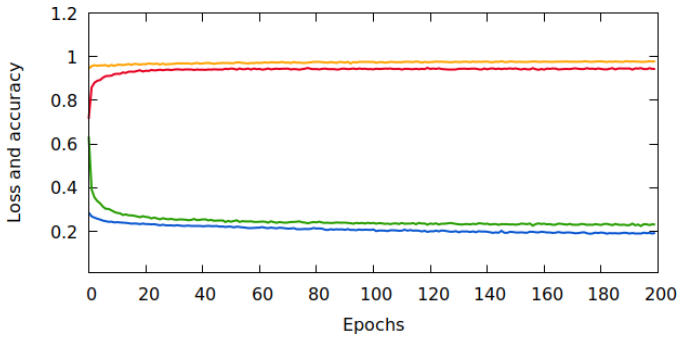

*(e) Broccoli Depth*



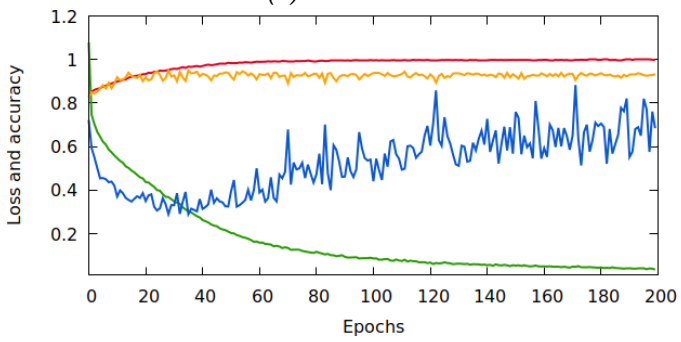
*Figure 4.12: Training of ReConv*

### ReConcat network

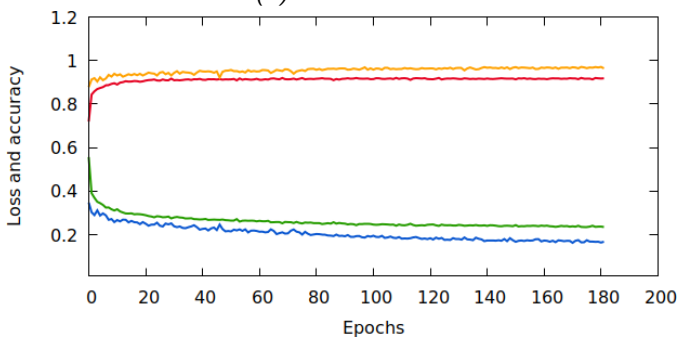
When adding a concatenation with the input on top of ReConv, we started from the parameters of ReConv, using the same hyperparameters. The results of the training processes are visible in Figure 4.13. The difference with ReConv is that this network is able to better follow the input image, resulting in a slightly increase in the segmentation accuracy. Nonetheless, this network still suffers from the poor performance of ReConv.



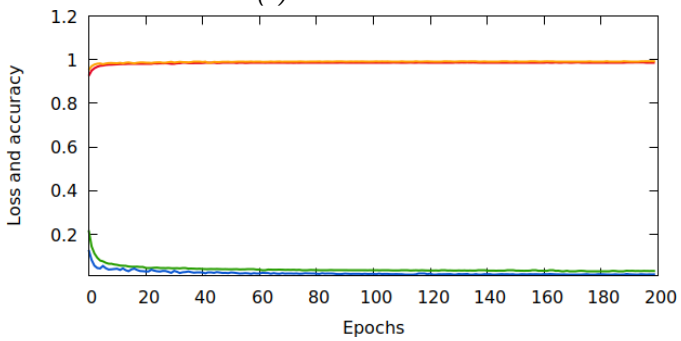

*(a) CWFID 1*



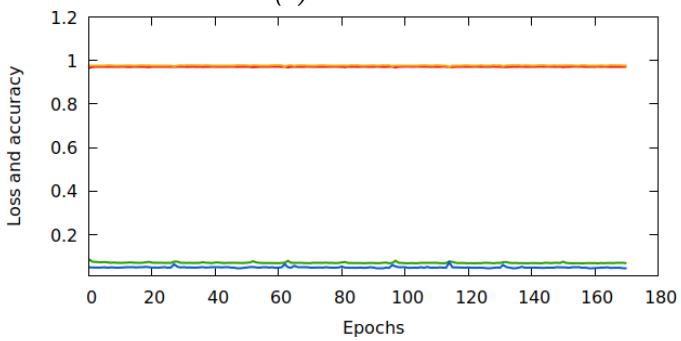

*(b) CWFID 2 1*




*(c) CWFID 2 2*




*(d) Broccoli*




*(e) Broccoli Depth*



*Figure 4.13: Training of ReConcat*

# Chapter 5

# Comparison of results

*"Marvin: That ship hated me.*
*Ford: Ship? What happened to it? Do you know?*
*Marvin: It hated me because I talked to it.*
*Ford: You talked to it? What do you mean you talked to it?*
*Marvin: Simple. I got very bored and depressed, so I went and plugged myself into its external computer feed. I talked to the computer at great length and explained my view of the universe to it.*
*Ford: And what happened?*
*Marvin: It committed suicide."*

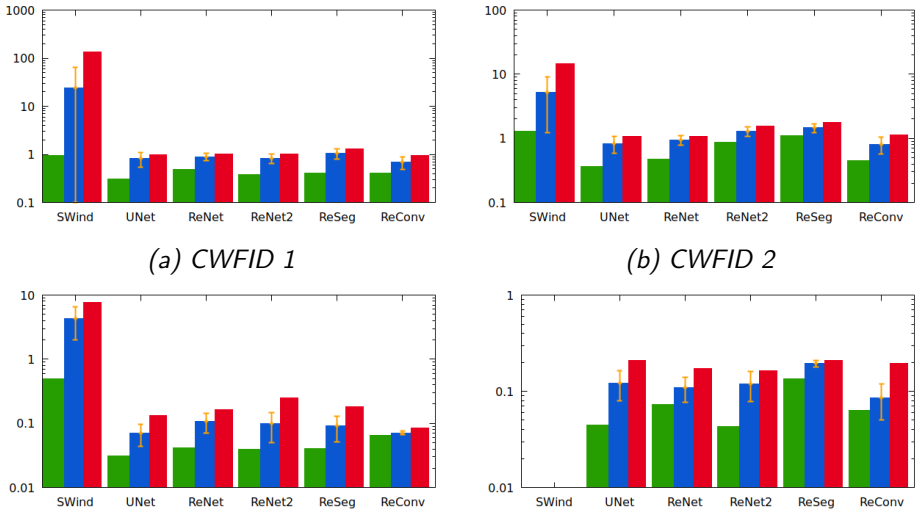
The Hitchhiker's Guide to the Galaxy

In this section we critically analyze, from various perspectives, the classifiers that we have implemented.

## 5.1 Comparison of hyperparameters dependence

The various networks exhibit a different resilience to the selection of the hyperparameters: some of them are really sensible and thus perform well just on a small range of values, whereas other networks have the great advantage of achieving good performances on a broader range of hyperparameters values. To further highlight this, we have used the data reported in appendix A to produce the graphs reported in Figure 5.1 and Figure 5.2, where, for every dataset, we report the minimum, the maximum, the average and the standard deviation of the validation loss that has been obtained by the various networks in the hyperparameters selection stage. As the loss is measured without considering the $l_2$ term, the results are directly comparable.

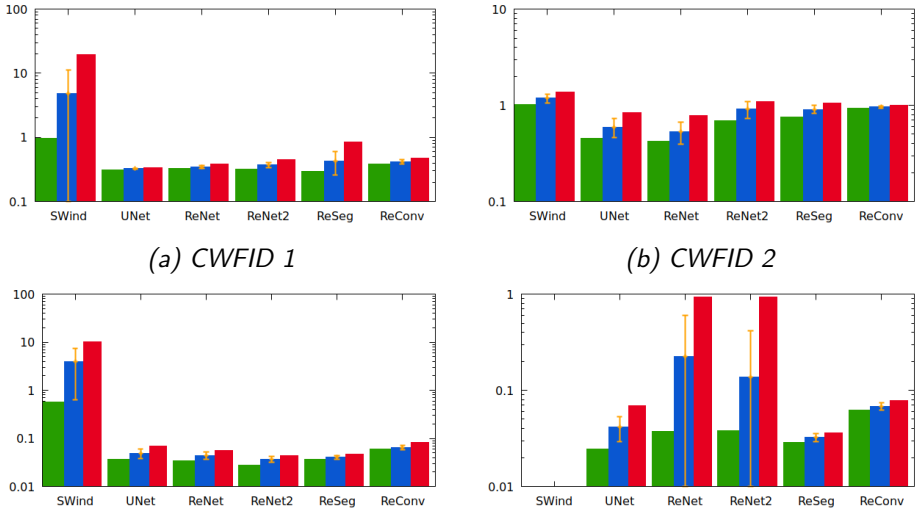
From the analysis of Figure 5.1, it is visible that, in every situation, the sliding window network has the greatest variability, therefore making it




*(a) CWFID 1* *(b) CWFID 2*

*(c) Broccoli* *(d) Broccoli depth*

*Figure 5.1: Validation loss after 3 epochs of training. Minimum, maximum, average and standard deviation of validation loss on logarithmic scale*




*(a) CWFID 1* *(b) CWFID 2*

*(c) Broccoli* *(d) Broccoli depth*

*Figure 5.2: Validation loss after 10 epochs of training. Minimum, maximum, average and standard deviation of validation loss on logarithmic scale*

crucial to select a proper set of hyperparameters. The other networks are more or less equivalent, with U-ReNet and U-Net usually achieving the best results. These results are confirmed in the finer tuning stage, reported in Figure 5.2, with the exception of the BroccoliDepth dataset where U-ReNet

has an higher variance. However, considering that the graphs are plotted on a logarithmic scale, we must notice that this higher variance is not nearly as bad as the one exhibited by the sliding window network.

Nonetheless, we must notice that the sliding window network has the tendency to obtain a really high loss even when the accuracy remains good, as explained in Section 4.2.3.

## 5.2 Feature analysis

As said before, we have decided to implement just a small set of handcrafted features hoping that more complex ones could have been identified by the neural networks. Moreover, one of the things we noticed while examining the importance of the hand crafted features (Section 4.2.2 on page 59) was that the two most important features were based on the colors of the images and the third one was a morphological property. It is therefore interesting to investigate which kind of features are being learned by the various networks.

We have explored two ways to visualize the extracted features:

- the first strategy consists in finding an image that, once fed to the network, produces a segmentation similar to a given one;
- the second strategy consists in visualizing the gradient of a given neuron so to understand what excites it.

### 5.2.1 Minimize output error

The first visualization can be obtained by starting from an empty image and computing the output of the network. The difference between the output and a given segmentation image is considered a loss, which is used to compute the derivative w.r.t. the input image pixels and correspondingly update the image. By repeating this procedure we obtain an image which, according to the network, represents the given segmentation. This strategy is useful to obtain an image that represents the abstract idea of a class according to the specific network. From a practical point of view, the loss has to be modified in order to make the image more pleasurable to the eye. In fact, the plain gradient of the loss would contain much high-frequency information and, therefore, the method would produce an image that exactly matches the given segmentation but is meaningless to the eye. Most of the previous works that focused on this kind of representations were conceived for classification networks, therefore we have tried to adapt them to segmentation networks. In fact, in segmentation problems a loss is specified for every pixel of the

output image and, when back-propagated, it affects multiple input pixels. From a practical point of view, this leads to noisy gradients that must be smoothed.

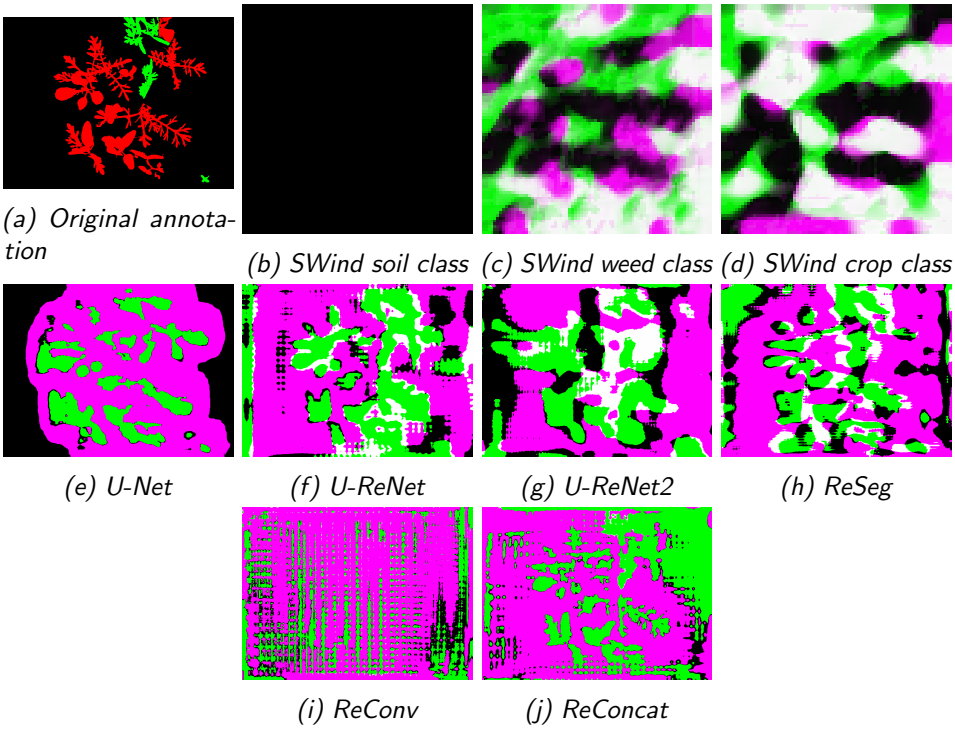
More specifically, following the pseudocode reported in Algorithm 2, we start from an all-zero image and compute its loss and the respective gradient. Then, in order to smooth the gradient, we compute two layers of the laplacian pyramid [1]. We update the gradient by adding the two pyramidal layers, thus giving more importance to the low frequency features and then normalize the gradient to unitary variance. After that, we put to zero all the gradient's points whose absolute value is lower than a small threshold, which we fixed to 0.05 times the mean of the image taken in absolute value. We then update the image using this gradient and the Adam update scheme, with $\beta_1 = 0.2$, $\beta_2 = 0.5$, $LR = 0.1$ and $\epsilon = 10^{-8}$ for 10 epochs. The values have been chosen just by trial and error trying to produce good-looking pictures.

**Algorithm 2** Minimize output error

**Require:** image: the (empty) image that has to be optimized
**Require:** target: the annotation that is used for the optimization of image
**Require:** net: a differentiable function (neural network) that, given an image, produces a target
  **for** epoch in 1...10 **do**
    $output \leftarrow net(image)$
    $loss \leftarrow cross\_entropy\_loss(output, target)$
    $gradient \leftarrow \frac{\partial loss}{\partial image}$
    $l \leftarrow laplacyan\_pyramid(gradient)$
    $gradient \leftarrow gradient + l(1) + l(2)$
    $gradient \leftarrow \frac{gradient}{std(gradient)}$
    $th \leftarrow 0.05 \cdot mean(|gradient|)$
    **for** point in gradient **do**
      **if** $|gradient[point]| < th$ **then**
        $gradient[point] \leftarrow 0$
      **end if**
    **end for**
    $image \leftarrow adam\_update(image, gradient)$
  **end for**

For the sliding window network, this approach can be adapted by using a class instead of a segmentation map and thus optimize a tile instead of an image.

In Figure 5.3 we report the images optimized with the networks trained on the first CWFID split. They are not much interpretable, but still they allow us to make some comparisons. The most important thing is that, more



(a) Original annotation (b) SWind soil class (c) SWind weed class (d) SWind crop class

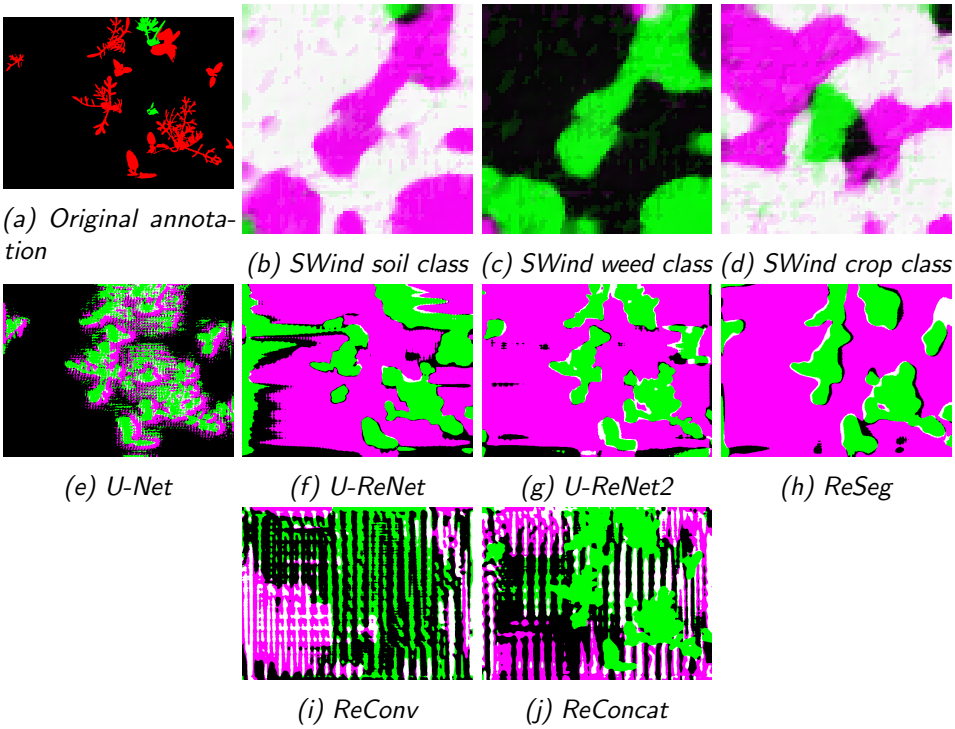
(e) U-Net (f) U-ReNet (g) U-ReNet2 (h) ReSeg

(i) ReConv (j) ReConcat

*Figure 5.3: Optimizing for CWFID annotation 001*

or less, all the networks seem able to recover the structure of the image, except for the ReConv network. This ability is an indication that the network is able to extract enough specialized features to capture the image content. The fact the ReConv is unable to recover the image confirms the results found during the training phase, where we hypothesized that this network is unable to extract sufficiently expressive features. The ReConcat network, conversely, is able to reconstruct the image, but this is probably mostly due to the direct concatenation, which would still imply that the ReConv network does not contain paths that embed the image's informations. For the sliding window network, the difference between the two vegetation classes is not so clear, but it seems to involve both the color and the shape.

The results for the second CWFID split (Figure 5.4) are visually more interesting. First of all, we still see the inability of ReConv to reconstruct the image and the better job done by ReConcat, thus confirming that ReConv does not have an adequate expressive power. Moreover, it is clearly visible the difference between the convolutional and the recurrent networks, where the former is really good at detecting borders trough the color contrast, whereas the latter exploits long distance relationship to detect the shape of the leaf. Interestingly, the recurrent networks seems to be mostly based on



(a) Original annotation (b) SWind soil class (c) SWind weed class (d) SWind crop class

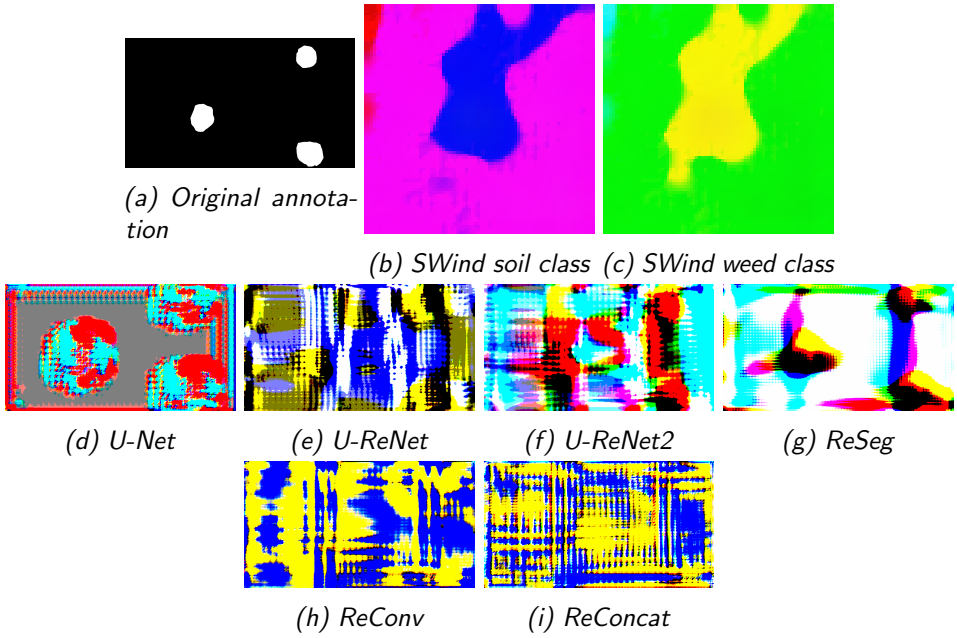
(e) U-Net (f) U-ReNet (g) U-ReNet2 (h) ReSeg

(i) ReConv (j) ReConcat

*Figure 5.4: Optimizing for CWFID annotation 002*

horizontal patterns.

The results for the Broccoli dataset are reported in Figure 5.5. We can observe that the sliding window network is mostly exploiting the color in order to distinguish between the classes. All the other networks seem to be trying to detect round shaped objects in correspondence of the broccoli heads, which is reasonable.



(a) Original annotation (b) SWind soil class (c) SWind weed class

(d) U-Net (e) U-ReNet (f) U-ReNet2 (g) ReSeg

(h) ReConv (i) ReConcat

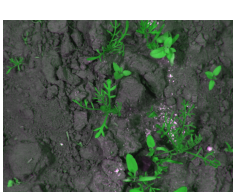
*Figure 5.5: Optimizing for Broccoli annotation 001*

### 5.2.2 Maximize neuron output

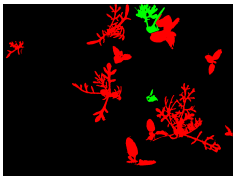
To obtain this visualization we start from an existing image and compute the output up to a certain layer. We then select a channel of the layer and compute its gradient w.r.t. the input image, thus obtaining an image that represent how we should modify the input image in order to increase the output and make stronger the implicit feature detected by the convolutional layer at the selected layer/channel. This visualization helps to understand what is being detected by a neuron, and therefore to see which kind of features are being extracted by the network. The resulting image, in fact, will have an higher value in the regions where the gradient is stronger, therefore identifying which regions mostly affect the implicit feature. Since we have multichannel images, the output will also tell us which channel we have to increase so to boost the feature. In order to produce better looking pictures, we have adopted the guided backrprop [47] variation to compute the gradient of the ReLU units. For graphic reasons we have lightened the images and increased the contrast just to make them more interpretable, since the resulting images are pretty dark.

For space reasons we cannot report all the neurons of all the networks, therefore we just display the positive gradient of some neurons for the networks trained on the second split of CWFID to give an idea of the kind of features extracted by the various networks. All the examples have been made using the annotation of the second image of CWFID, which we report in Figure 5.6.

For the U-Net network, we report in Figure 5.7 some images taken from neurons located in the pooling and upsampling layers. The first layers of the network are used to extract some really simple, color based, features used to discriminate soil and vegetation (Figure 5.7b) and also to detect left and right borders of the leaves (Figure 5.7a and Figure 5.7c). The network progressively learns to extract more complex features, for example at the third pooling layer it is already detecting the whole borders of leaves in a pretty accurate way (Figure 5.7d). At the next layer the network is still detecting borders, but it has also developed enough features to discriminate between different leaves (Figure 5.7e and Figure 5.7f). In the upsampling path the network uses the learned features to compute the final segmentation image by separately identifying the entire plants (Figure 5.7g and Figure 5.7h). We can thus say that this network is mostly reasoning in terms of leaves shape and color, from the analysis of Figure 5.7b we can also hypothesize that the network is extracting a sort of texture of the leaves, but this features is not present in the subsequent layers.

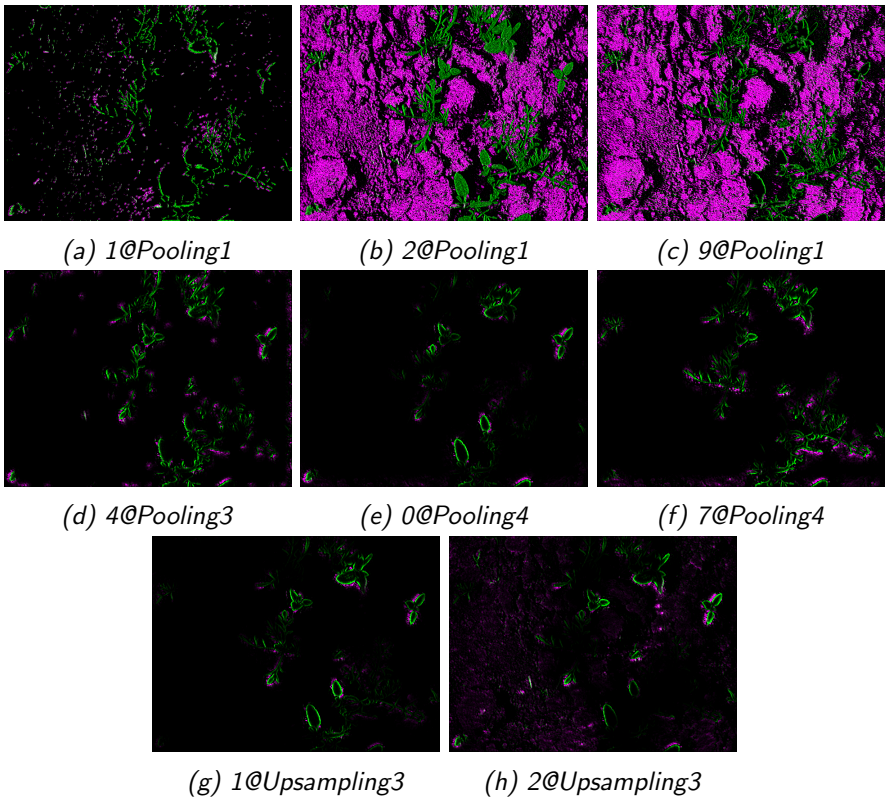


(a) Image 2 of CWFID



(b) Segmentation of CWFID image 2

*Figure 5.6: CWFID image 2: original image and ground truth annotation.*



(a) 1@Pooling1 (b) 2@Pooling1 (c) 9@Pooling1

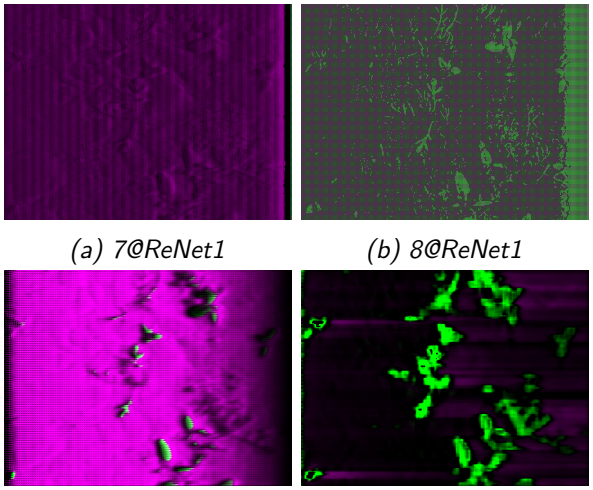
(d) 4@Pooling3 (e) 0@Pooling4 (f) 7@Pooling4

(g) 1@Upsampling3 (h) 2@Upsampling3

*Figure 5.7: U-Net trained on CWFID2. Gradients of some neurons, labels are meant as neuron@layer*

Due to their recurrent nature, the results obtained for the ReNet-based architectures are not so easily interpretable. In fact, the gradient in a pixel is the effect of the activation of many, possibly distant, other pixels and therefore the images are interpretable just on the very first layers where they are mostly exploiting local informations. Nonetheless, we will report some of them.

Some neurons from the U-ReNet network are shown in Figure 5.8. This



(a) 7@ReNet1 (b) 8@ReNet1

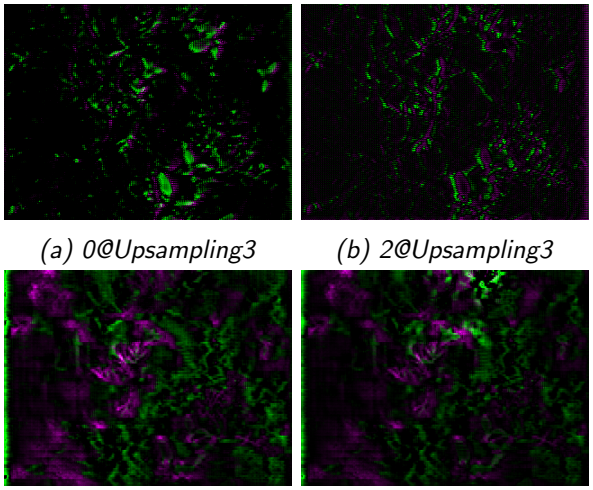
(c) 9@ReNet2 (d) 7@ReNet3

*Figure 5.8: U-ReNet trained on CWFID2. Gradients of some neurons, labels are meant as neuron@layer*

network mostly relies on the borders of the leaves to classify them, in fact, as visible in Figure 5.8a the network is computing a sort of low relief of the image. Figure 5.8b clearly shows the underlying structure of the network, which is based on a set of recurrent neurons sweeping the image both horizontally and vertically: the presence of neighboring pixels with contrasting colors suggests a feature that detects borders. In Figure 5.8c and Figure 5.8d we see that the network is also utilizing colors to detect the plants.

In Figure 5.9 we display some neurons taken from later layers of U-ReNet2. In Figure 5.9a and Figure 5.9b it is visible that the network is exploiting the color contrast in order to detect the borders of the leaves. Figure 5.9c and Figure 5.9d have been taken from the output layer and thus represent how we should change the image in order to make the network more sure about the final segmentation. The images are not very clear, but it is interesting to notice that the main difference between the images is located in the area where there should be a weed. This indicates that, in order to make that area more weed-like, the network would like to see a change *in the same area*. This is an indication that the network is effectively recognizing the plant and not just the surrounding context like it would happen if the network was overfitting (consider that the given annotation was part of the training dataset).

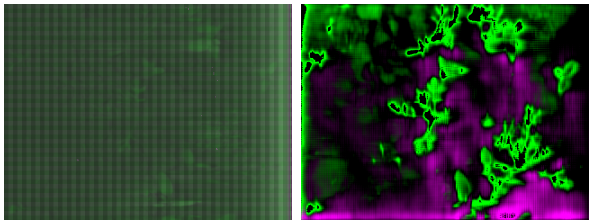
The ReSeg-based networks are even less interpretable than the U-ReNet ones, so we show just a couple of neurons in Figure 5.10, so to notice that they maintain the vertical/horizontal structure that arise from the ReNet



(a) 0@Upsampling3 (b) 2@Upsampling3

(c) 1@Output (d) 2@Output

*Figure 5.9: U-ReNet2 trained on CWFID2. Gradients of some neurons, labels are meant as neuron@layer*



(a) 0@ReNet1 (b) 1@ReNet2

*Figure 5.10: ReSeg trained on CWFID2. Gradients of some neurons, labels are meant as neuron@layer*

layers.

*Table 5.1: Number of parameters*

| Network | CWFID | Broccoli | BroccoliD |
|---|---|---|---|
| SWind | 10 011 267 | 10 012 674 | - |
| U-Net | **503267** | **503330** | **503426** |
| U-ReNet | 1 224 611 | 1 225 778 | 1 226 978 |
| U-ReNet2 | 1 218 083 | 1 218 818 | 1 219 586 |
| ReSeg | 1 212 963 | 1 214 362 | 1 216 042 |
| ReConv | 2 201 459 | 2 201 194 | 2 201 290 |
| ReConcat | 2 201 465 | 2 201 200 | 2 201 298 |

## 5.3 Number of parameters

An important aspect of the various networks is the number of parameters. Ideally we would like to have a network that reaches optimal segmentation accuracies with a few number of parameters, since this would mean a lower training and prediction time. Moreover it would ease the training process by making the network less prone to overfitting. The number of parameters for the various networks is reported in Table 5.1, from its analysis we can see that the simplest network is the U-Net one, whereas the sliding window network, albeit its simple structure, is, by far, the most complex one. This is in accordance with the fact that, for this network, we did not perform any convolution factorization. Moreover, the fully connected layers involve a really big number of parameters.

## 5.4 Comparison of training time

In Table 5.2 we report the time needed to train each network on the various datasets. The train has been performed on the Galileo system from cineca, equipped with a NVidia Tesla K80 gpu. Since some networks required to be trained multiple times, we will report here just the time employed for the train that yield the best result in terms of validation loss. Moreover, we consider the train time just up to the epoch that we selected as the best considering its validation loss.

Given that the selection of the optimal epoch is a noisy process, in Table 5.3 we also report the average time per epoch of each configuration.

By analyzing the tables we can say that the sliding window network and the U-Net architecture are the fastest, while the ReNet-based networks pay the fact of not being easily parallelizable: in fact, as outlined in [52], the recurrent networks have a sequential nature and thus we cannot fully take

*Table 5.2: Training time in hours*

| Network | CWFID1 | CWFID2 | Broccoli | BroccoliD |
|---|---|---|---|---|
| SWind | **3.5** | **10.4** | **2.0** | - |
| U-Net | 5.0 | 15.1 | 11.3 | 15.6 |
| U-ReNet | 13.7 | 39.0 | 23.6 | **9.34** |
| U-ReNet2 | 13.5 | 37.0 | 22.5 | 21.8 |
| ReSeg | 11.2 | 37.4 | 19.5 | 20.6 |
| ReConv | 8.19 | 17.0 | 15.1 | 4.9 |
| ReConcat | 7.46 | 16.4 | 3.24 | 11.0 |

*Table 5.3: Average training time per epoch in seconds*

| Network | CWFID1 | CWFID2 | Broccoli | BroccoliD |
|---|---|---|---|---|
| SWind | **98** | **273** | 268 | - |
| U-Net | 126 | **273** | **205** | 298 |
| U-ReNet | 262 | 530 | 440 | 542 |
| U-ReNet2 | 249 | 470 | 404 | 392 |
| ReSeg | 270 | 459 | 355 | 392 |
| ReConv | 150 | 309 | 271 | 264 |
| ReConcat | 151 | 345 | 233 | **245** |

advantage of the GPU parallelism. When comparing the sliding window network with the U-Net architecture, we must consider their different nature: the former works on tiles, while the latter receives the full images. Therefore, U-Net will take the same time on all the images, whereas the sliding window approach requires to have a smart way to preprocess the images and select just the relevant tiles. For example, in the Broccoli dataset we do not remove the background and, therefore, for each image we have a lot of overlapping tiles, which requires the network to repeat the same computations over and over again, thus slowing down the training process.

Moreover, loading all the tiles of an image can be demanding in terms of memory and thus require a powerful GPU. We can thus conclude that the sliding window approach is convenient when an easy and fast method to clean the images is available, whereas it is better to use U-Net when the images cannot be cleaned and all pixels must be processed. For what concerns the recurrent networks, we can say that ReConv is the fastest network, which is expected given that is has less layers. Moreover, U-ReNet2 is faster than U-ReNet, which, again, is in accordance with the fact that it has less hidden units in the first layers (where the image is bigger) and more neurons in the

*Table 5.4: Average image classification time on Cineca server in seconds*

| Network | CWFID1 | CWFID2 | Broccoli | BroccoliD |
|---|---|---|---|---|
| RFC | 3.53 | 3.46 | 58.57 | - |
| SVM | 3.46 | 3.97 | 59.71 | - |
| XGB | 3.24 | 3.16 | 60.28 | - |
| SWind alone | 3.49 | 4.77 | 14.36 | - |
| SWind U-Net | - | - | 2.94 | - |
| U-Net | 0.54 | 0.73 | 2.07 | 1.25 |
| U-ReNet | 0.82 | 1.22 | 2.16 | 1.33 |
| U-ReNet2 | 0.84 | 0.81 | 1.96 | 1.31 |
| ReSeg | 0.68 | 0.83 | **1.85** | 1.23 |
| ReConv | 0.56 | **0.63** | 1.89 | **1.03** |
| ReConcat | **0.52** | 0.68 | 1.88 | 1.08 |

*Table 5.5: Average image classification time on laptop in seconds*

| Network | CWFID1 | CWFID2 | Broccoli | BroccoliD |
|---|---|---|---|---|
| RFC | 5.13 | 5.18 | 97.48 | - |
| SVM | 5.38 | 5.86 | 96.50 | - |
| XGB | 5.23 | 4.88 | 99.75 | - |
| SWind alone | 4.12 | 4.33 | 77.65 | - |
| SWind U-Net | - | - | 9.69 | - |
| U-Net | 1.25 | 1.36 | 2.94 | 2.03 |
| U-ReNet | 2.15 | 2.28 | 3.68 | 2.63 |
| U-ReNet2 | 1.83 | 1.76 | 3.32 | 2.36 |
| ReSeg | 1.94 | 1.91 | 3.36 | 2.46 |
| ReConv | **1.19** | **1.17** | 2.91 | 1.99 |
| ReConcat | 1.45 | 1.27 | **2.91** | **1.97** |

last layers, where the image size has already been reduced.

## 5.5 Comparison of classification time

Another interesting measure is the time needed to classify an image with each classifier, as this affects the real-time applicability of the system. As this process, in principle, has to be carried out in field, we report in Table 5.4 the time needed by the same system we used for training and also, in Table 5.5, the time needed by an average laptop equipped with an NVidia GeForce 740M as it is more similar to an hypothetic real-life in-field device. The classification time does not include the time needed to compile the networks.

Interestingly, the baseline classifiers are not faster than the networks on CWFID and they are dramatically slower on the Broccoli dataset. This is due to the fact that the tiles are hugely overlapping, which, even if it is an advantage in terms of accuracy, is also really painful in terms of computational performance. On the CWFID dataset this disadvantage is compensated by the fact that we have a fast way to remove the soil and thus evaluate just a small subset of the available tiles. The sliding window network is affected by the same problem, however, since it runs on a GPU, the massive parallelization alleviates the problem. It is also interesting to notice that, on the Broccoli dataset, it is faster to run the U-Net network and use its result to produce a mask image and thus reduce the number of windows to be classified by the sliding window approach. Among the other networks, the ReConv and ReConcat architecture are the fastest. This is explained by the fact that they have the most shallow structure and, moreover, the recurrent layers work on a preprocessed image, which is therefore smaller. However, as we will see later, those two architectures do not reach optimal segmentation accuracies, and therefore we have to also analyze the other ones. U-Net is, as expected, faster than U-ReNet architectures and it is also nearly as fast as the ReConv networks. We can thus conclude that, time-wise, U-Net architecture is the best solution.

## 5.6 Comparison of segmentation performance

Once obtained the various classifiers, we have evaluated their performance on the test set. This evaluation is performed at pixel level on full-sized images. Therefore it involves an upsampling process for the networks and an interpolation for the baseline classifiers as explained in Section 3.3. The smoothing is still not applied.

In Table 5.6 we report the results obtained on the first split of CWFID, where there is not a clear winner. The sliding window network is the most balanced of all, since it reaches good performances on all the metrics. Conversely, the U-Net is the worst one as it is biased toward the weed class and annotates as weed more than it should.

In Table 5.7 we report the results obtained on the second split of CWFID. The increased size of the training dataset gives an advantage to all the classifiers, but the neural networks receive a greater boost and outperform the baseline classifiers. Here we can say that U-Net is the most suited network for the job. It still has a tendency to be biased toward the weed class but, apart from precision, it greatly outperforms all the concurrents in every metric. It is worth mentioning the fact that U-ReNet reaches acceptable

*Table 5.6: Segmentation results on CWFID1*

| Classifier | Accuracy | Precision | Recall | F1 | Jaccard |
|---|---|---|---|---|---|
| [15] | 0.859 | 0.796 | 0.808 | 0.802 | - |
| RFC | 0.843 | 0.848 | 0.888 | 0.849 | 0.866 |
| SVM | **0.861** | 0.846 | 0.916 | 0.860 | 0.866 |
| XGB | 0.851 | 0.838 | 0.920 | 0.859 | 0.866 |
| SWind | 0.857 | 0.830 | 0.950 | **0.865** | **0.866** |
| U-Net | 0.773 | 0.775 | 0.929 | 0.821 | 0.866 |
| U-ReNet | 0.808 | 0.801 | **0.954** | 0.846 | 0.866 |
| U-ReNet2 | 0.809 | 0.815 | 0.925 | 0.839 | 0.865 |
| ReSeg | 0.843 | **0.855** | 0.895 | 0.851 | 0.865 |
| ReConv | 0.817 | 0.820 | 0.927 | 0.843 | 0.865 |
| ReConcat | 0.838 | 0.822 | 0.944 | 0.856 | 0.866 |

*Table 5.7: Segmentation results on CWFID2*

| Network | Accuracy | Precision | Recall | F1 | Jaccard |
|---|---|---|---|---|---|
| RFC | 0.855 | 0.886 | 0.888 | 0.881 | 0.879 |
| SVM | 0.870 | 0.887 | 0.911 | 0.894 | 0.879 |
| XGB | 0.852 | 0.884 | 0.890 | 0.880 | **0.879** |
| SWind | 0.885 | **0.925** | 0.897 | 0.903 | 0.879 |
| U-Net | **0.897** | 0.886 | **0.972** | **0.922** | 0.879 |
| U-ReNet | 0.863 | 0.873 | 0.942 | 0.898 | 0.879 |
| U-ReNet2 | 0.873 | 0.895 | 0.915 | 0.897 | 0.879 |
| ReSeg | 0.854 | 0.860 | 0.946 | 0.894 | 0.879 |
| ReConv | 0.737 | 0.770 | 0.883 | 0.809 | 0.879 |
| ReConcat | 0.743 | 0.769 | 0.846 | 0.799 | 0.879 |

level of performance on both the splits, which means that it suffers the lack of training data less than U-Net. In conclusion we can say that, depending on the size of the available training data, the sliding window network and U-Net should be the preferred architectures for CWFID.

When analyzing the results for the Broccoli dataset, reported in Table 5.8, the small gap between the convolutional and the recurrent networks is no longer present. In fact, U-ReNet2 seems the best architecture, achieving good results on all the metrics. This may be an indication that, for this job, the ability of the recurrent networks to capture long distance dependences is useful, thus making the ReNet based architecture the best choice. Another relevant observation regards the sliding window network: as said, we have tested it both when used alone and when staged after the U-Net network

*Table 5.8: Segmentation results on Broccoli*

| Network | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| [22] [1] | - | 0.952 | - | - |
| RFC | 0.952 | 0.562 | 0.345 | 0.414 |
| SVM | 0.946 | 0.477 | 0.388 | 0.417 |
| XGB | 0.882 | 0.212 | 0.466 | 0.288 |
| SWind alone | 0.633 | 0.106 | 0.833 | 0.187 |
| SWind U-Net | 0.986 | **0.959** | 0.755 | 0.841 |
| U-Net | 0.988 | 0.924 | 0.842 | 0.878 |
| U-ReNet | 0.984 | 0.870 | 0.813 | 0.835 |
| U-ReNet2 | **0.989** | 0.895 | **0.889** | **0.888** |
| ReSeg | 0.986 | 0.882 | 0.834 | 0.854 |
| ReConv | 0.974 | 0.783 | 0.683 | 0.724 |
| ReConcat | 0.973 | 0.785 | 0.649 | 0.701 |

*Table 5.9: Segmentation results on BroccoliD*

| Network | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| U-Net | 0.983 | 0.877 | 0.786 | 0.819 |
| U-ReNet | 0.980 | 0.911 | 0.687 | 0.776 |
| U-ReNet2 | 0.985 | **0.918** | 0.785 | 0.842 |
| ReSeg | **0.987** | 0.902 | **0.833** | **0.864** |
| ReConv | 0.975 | 0.794 | 0.693 | 0.735 |
| ReConcat | 0.975 | 0.787 | 0.708 | 0.741 |

so to have a mask and reduce the number of windows to be classified. It is visible that, when used alone, the network does not absolutely reach a good performance level. Therefore, the results obtained by the sliding window network on preprocessed images can be mostly attributed to the U-Net network. We can thus conclude that U-ReNet2 vastly outperforms the sliding window approach.

In Table 5.9 we report the results obtained on the Broccoli dataset enriched with the depth channel. The first relevant comment is that those results are worse than the ones obtained without the depth channel. This is probably due to two problems:

- the depth channel is really noisy;
- our architectures mix the depth with the color channel.

[1] We remind that those results have been obtained using cloudpoint data and different ground truths

The second issue is the most relevant. In fact, we have stacked the depth image as a fourth channel and thus it was involved in all the neural computations. Therefore, it is hard for the networks to extract some depth-related features and use them. Moreover, the color-based features also receive an additional channel that slows them down and adds noise to the process. Therefore, we cannot evaluate the gain given by the depth information, but we can treat these results as an additional experiment that confirms the results found when using the 2D images only. With this in mind, we can observe that the ReSeg network obtains the best results and that U-ReNet2 is also very good. We can therefore conclude that, accuracy wise, the ReNet-based architectures are the best for the Broccoli dataset.

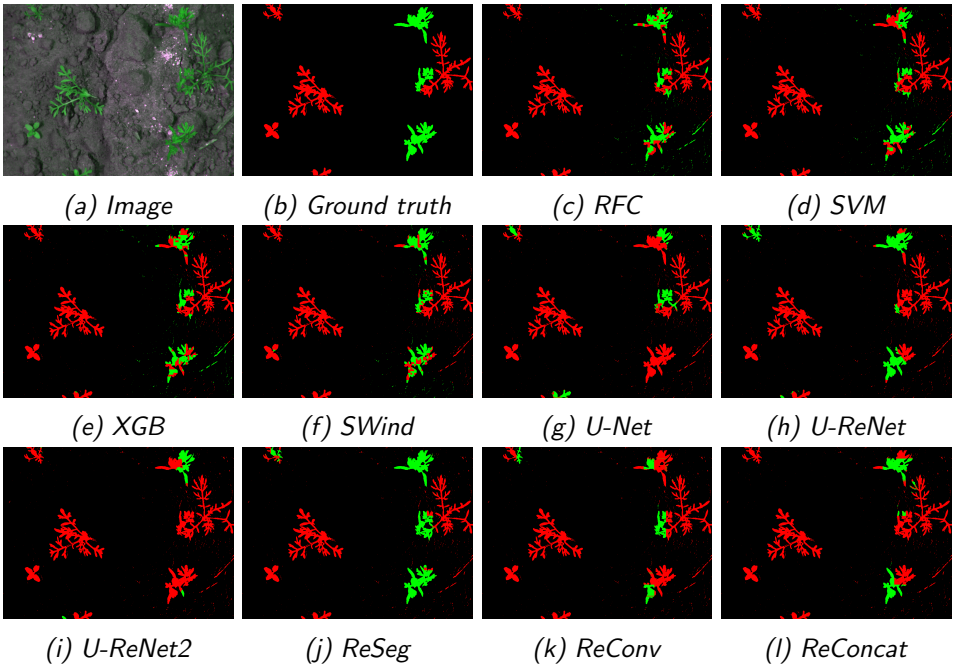
### 5.6.1 Qualitative comparison of segmentations

The segmented images can also be evaluated from a qualitative point of view. In fact, the same metrical result can be obtained in several ways. For instance, imagine we have a CWFID image containing 4 plants, none of which is weed. Imagine also that we have two hypothetic classifiers: the first one labels as crop three plants and as weed the fourth one, whereas the second classifiers labels all the plants as crop, except for some randomly placed pixels that are labeled as weed. The first classifier would lead to the destruction of a crop plant, whereas the error of the second classifier could be easily recovered by filtering on the dimension of the predicted weed. The same problem is present in the Broccoli dataset: our final goal, in fact, is to predict the growth stage of the broccoli heads and, therefore, it is important to produce an annotation that can be used to measure their size. This means that a good annotation is not an annotation that reaches high scores on all the metrics, but an annotation that "looks nice" and closely matches the borders of the heads. Moreover, it is important to evaluate how the various networks react to the occluded part of the leaves.

Albeit this difference is not captured by the adopted metrics, we can qualitatively evaluate the segmented images in order to see if the various classifiers are more prone to specific kinds of error.

The classifiers trained on the first split of CWFID do not present substantial differences. We can see (images from 5.11c to 5.11f) that the classifiers that are based on tiles have a tendency to produce highly fragmented annotation, which is the practical reason that justifies the smoothing process.

It is also interesting to notice that the ReSeg based networks (images from 5.11j to 5.11l) produce annotations whose boundaries are straight lines that follows an horizontal or vertical pattern. This is justified by the nature



(a) Image (b) Ground truth (c) RFC (d) SVM

(e) XGB (f) SWind (g) U-Net (h) U-ReNet

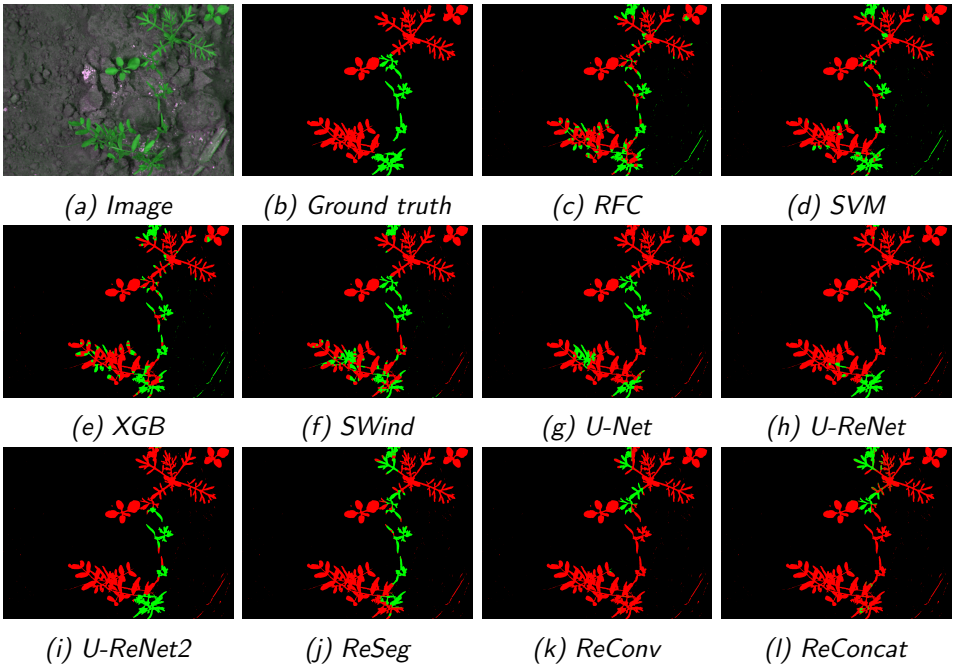
(i) U-ReNet2 (j) ReSeg (k) ReConv (l) ReConcat

*Figure 5.11: Classification of CWFID image 027*

of the underlying recurrent networks and by the upsampling process. In fact, the output of these networks is heavily upsampled in a single passage, and thus it can't be smooth. This suggests that the upsampling process adopted in the U-ReNet architecture is useful to produce better looking annotations, whereas the concatenation trick implemented in the ReConcat network is not enough.

From the analysis of the annotations produced by the classifiers trained on the second split of CWFID (Figure 5.12), we observe a similar pattern: the tile based classifiers are really noisy and the ReSeg based networks produce non smooth boundaries. We can thus conclude that, from qualitative observations, the U-shaped networks produce the best results on the CWFID images.

The outputs for the Broccoli dataset (Figure 5.13) are quite different from the CWFID ones. As a first thing, it is clearly visible that the tile based classifiers achieve really bad performances, as resulted from the analysis of the metrics. The sliding window network actually reaches a good precision, but from visual inspection it is visible that the annotations have a really low quality. This problem can arise from the fact that, differently from CWFID, here it is important to classify the *whole* plant, producing nicely looking masks that can be used to compute the size of the broccoli heads. Therefore, if a classifier receives just a small window, it cannot exploit the surrounding



(a) Image (b) Ground truth (c) RFC (d) SVM

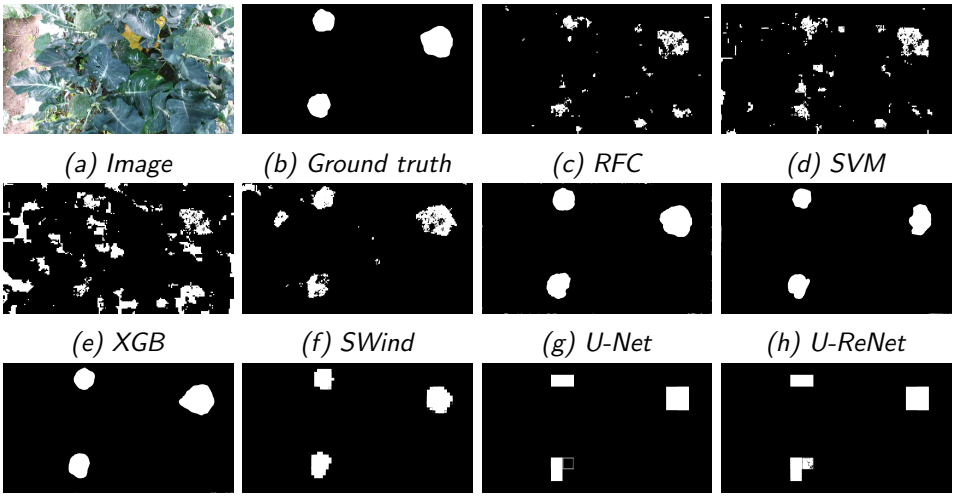
(e) XGB (f) SWind (g) U-Net (h) U-ReNet

(i) U-ReNet2 (j) ReSeg (k) ReConv (l) ReConcat

*Figure 5.12: Classification of CWFID image 032*

context to ease the decision process. In other terms, when classifying a window that has been extracted at the center of a broccoli head, it just looks like a big green square, which is not dramatically different from a big leaf. Therefore all the features that exploit the contour cannot be used and the classification task is more demanding since it can only exploit color informations. This is in accordance with the feature importance analysis that we reported in Section 4.2.2, where it resulted that, for the Broccoli dataset, the classifiers were mostly based on a color feature. Conversely, the other networks can easily exploit the local context and therefore produce way better segmentations. Following these reasoning, we can also justify the fact that the ReNet-based architectures achieved better scores on the metrics. In fact, due to their ability of identifying long distance dependences, they can exploit this context more easily than the convolutional networks. Nonetheless, the U shaped architecture of U-Net allows to obtain very good results and it becomes the best solution when paired with the ReNet layer as done in U-ReNet.

As it happened in CWFID, the ReSeg architecture, due to its upsampling process, produces annotations with non smooth boundaries, but here the problem is more serious since a smooth boundary would make it easier to compute the size of the crop. When moving to the ReConv network the result

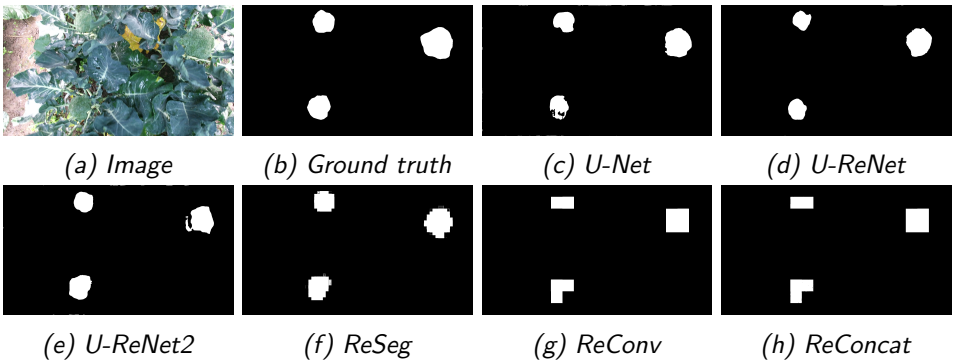


(a) Image (b) Ground truth (c) RFC (d) SVM

(e) XGB (f) SWind (g) U-Net (h) U-ReNet

(i) U-ReNet2 (j) ReSeg (k) ReConv (l) ReConcat

*Figure 5.13: Classification of Broccoli image 245*



(a) Image (b) Ground truth (c) U-Net (d) U-ReNet

(e) U-ReNet2 (f) ReSeg (g) ReConv (h) ReConcat

*Figure 5.14: Classification of Broccoli image 245 also using depth channel*

becomes really bad, indicating that, for the Broccoli dataset, the features extracted by the U-Net network are really unsuited as a preprocessing step for the ReSeg network.

A similar result is obtained on the depth enriched version of the Broccoli dataset. We can thus conclude that, from a qualitative point of view, U-ReNet produces the best results on both the datasets.

*Table 5.10: Smoothing parameters*

| Dataset | Classifier | X | Y | R |
|---|---|---|---|---|
| CWFID1 | RFC | 2 | 2 | 1 |
| | SVM | 3 | 4 | 5 |
| | XGB | 2 | 2 | 1 |
| | SWind | 2 | 3 | 5 |
| CWFID2 | RFC | 2 | 2 | 1 |
| | SVM | 4 | 3 | 5 |
| | XGB | 4 | 3 | 5 |
| | SWind | 2 | 2 | 1 |

*Table 5.11: Average image classification time on Cineca server in seconds when using spatial smoothing*

| Network | CWFID1 | CWFID2 |
|---|---|---|
| RFC | 3.30 | 3.34 |
| SVM | 3.41 | 3.57 |
| XGB | 3.74 | 3.06 |
| SWind alone | 2.85 | 5.47 |

## 5.7 Smoothing tile probabilities

The performance of the tile based classifiers can be increased by applying a spatial smoothing. The effect of this step is to reduce the high frequency variations in the predictions of the classifiers, thus producing better segmentations. We have tuned the parameters of Algorithm 1 for each classifier on both the CWFID splits, looking for the values bringing to the best F1 score on train images. We have selected the F1 score since it is a good balance between precision and recall. The optimal values are reported in Table 5.10, where X and Y represent the number of tile considered by the smoothing algorithm and R is the number of repetitions. This process has not been performed for the Broccoli dataset, where the results obtained by tile-based classifiers are really bad and far from being recoverable with a smoothing.

With the found parameters, we have re-evaluated the tile-based classifiers on the CWFID dataset.

The first thing that we need to evaluate is the increase in the computational complexity that comes from the smoothing stage. In fact, preliminary experiments shown that the smoothing process proposed in [16] is too slow for real-time applications. From the comparison of Table 5.11 with Table 5.4 on

*Table 5.12: Segmentation results on CWFID1 using spatial smoothing*

| Classifier | Accuracy | Precision | Recall | F1 | Jaccard |
|---|---|---|---|---|---|
| RFC | 0.861 | 0.857 | 0.909 | 0.864 | 0.866 |
| SVM | **0.894** | **0.895** | 0.936 | **0.908** | 0.866 |
| XGB | 0.868 | 0.847 | 0.936 | 0.871 | 0.866 |
| SWind | 0.875 | 0.843 | **0.970** | 0.881 | **0.866** |

*Table 5.13: Segmentation results on CWFID2 using spatial smoothing*

| Classifier | Accuracy | Precision | Recall | F1 | Jaccard |
|---|---|---|---|---|---|
| RFC | 0.871 | 0.897 | 0.900 | 0.893 | 0.879 |
| SVM | 0.894 | 0.904 | **0.930** | 0.910 | 0.879 |
| XGB | 0.893 | 0.913 | 0.923 | 0.906 | **0.879** |
| SWind | **0.895** | **0.934** | 0.906 | **0.911** | 0.879 |

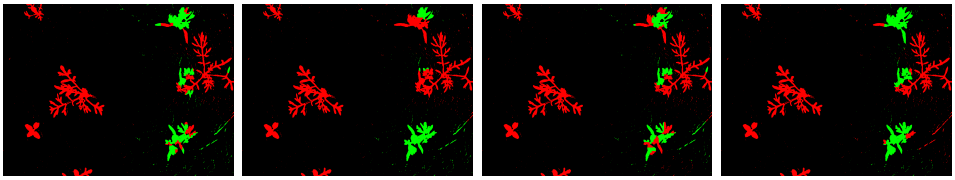
page 90 it is visible that the additional smoothing stage performed with our algorithm does not affect the computational time in an appreciable way, since the visible variations are more likely to be arising from random fluctuations depending on the server workload.

From the comparison of Table 5.12 and Table 5.6 on page 92 we can see how effective is the smoothing process, granting, on average, an improvement of 2.15% in terms of accuracy [2]. With this additional step, the support vector machine becomes the best classifier on the first split of CWFID.

Similarly to what observed for the first CWFID split, the comparison of Table 5.12 and Table 5.6 on page 92 results in a considerable increase in all the metrics [3]. However, none of the tile-based classifiers outperforms the U-Net architecture. In summary, the smoothing improves the performance even on the second split, but the neural networks still reach better performances.

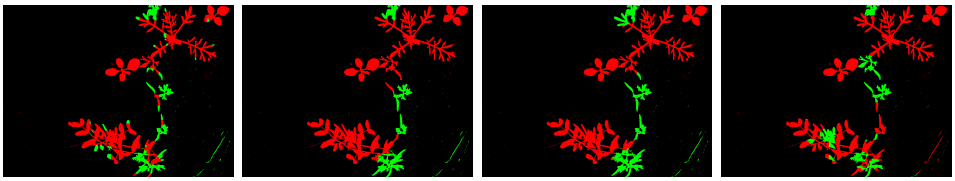
The last comparison regards the visual differences in the produced segmentations. As visible in Figure 5.15 and Figure 5.16, the smoothing process reduces the high frequency variations in the predictions. However, it is also visible that this process is just useful to remove outliers and increase the metrics, but it is not increasing the real quality of the recommendations.

[2]The smoothing process proposed in [16] achieve an accuracy improvement of 2.3%
[3]The average accuracy improvement on CWFID2 is 3%



*(a) RFC smooth* *(b) SVM smooth* *(c) XGB smooth* *(d) SWind smooth*

*Figure 5.15: Classification of CWFID image 027 with smoothing*



*(a) RFC smooth* *(b) SVM smooth* *(c) XGB smooth* *(d) SWind smooth*

*Figure 5.16: Classification of CWFID image 032 with smoothing*

## 5.8 Summary of comparison

After having analyzed all the various aspects of the networks, we can now directly compare them so to summarize the peculiarities of each network.

### 5.8.1 Feature based classifiers VS neural networks

This work is concerned with the comparison of neural network architectures for agricultural image segmentation tasks. However, the first comparison must be done with the feature-based classifiers that we implemented as baselines. In fact, those classifiers are able to achieve really good performances, especially on the small datasets, where neural networks suffer the most. However, we must also notice that, even on the smaller CWFID split with just 20 training images, the neural networks achieve a level of performance comparable to the one obtained by the feature-based classifiers. This is especially true for the sliding window network, which is able to fully take advantage from the overlapping of the tiles.

Moreover, on the second CWFID split, which is still a very small dataset, the feature-based classifiers do not substantially improve their performances, whereas the neural networks are able to fully take advantage from the additional data and constantly outperform the feature-based classifiers, even if the training data is limited to just 40 images. This shows that, through a severe use of data augmentation techniques, deep neural networks can be successfully trained even on small datasets.

### 5.8.2 Sliding window approach VS full-image segmentation

Once decided to use a neural network, one must decide the kind of strategy to adopt for segmentation. The sliding window approach usually reaches good segmentation accuracies, which is mostly due to its implicit augmentation of the available data. In fact, being based on overlapping tiles, the same pixel is used in several computations, and thus the network can be trained more easily without incurring in severe overfitting. However, relying on overlapping data has also negative consequences. First of all, it requires a way to efficiently get rid of unnecessary tiles, so to reduce both the training dataset dimension and the classification time. This in turn requires to either have a segmentation mask for the soil, which can be obtained only with costly handwork, or to have a vegetative index that can be efficiently thresholded.

If a segmentation mask for the soil is not available, the network still reaches good accuracies, albeit this comes at the expense of a dramatic increase in both memory and temporal requirements. Another disadvantage that arises from the tiled nature of this classifier is its inability to vary the size of the local context used to classify a point. In fact, the tile dimension is a really important hyperparameter and choosing a too small value can lead to bad results, as happened on the Broccoli dataset.

Summarizing, the sliding window approach is a good choice to segment small plants when just few images are available, their vast majority is composed by soil and there is an efficient way to remove it. Those limitations may seem severe, but this is precisely the case of CWFID1 and therefore ideally corresponds to a real-life application.

### 5.8.3 Convolutional VS recurrent networks

One of the things we investigated is whether this specific domain can benefit from the usage of recurrent neural architectures. The ReNet layer, in fact, can be used as a drop-in replacement for the usually adopted convolution + pooling layers, with the advantage of being able to identify long distance relationship. However, the segmentation of an agricultural image may not require those relationship, making the usage of ReNet a disadvantage. In fact, it must also be underlined that the recurrent layers have a greater temporal complexity since they cannot be parallelized. Moreover, the U-shaped architecture of U-Net, in principle, allows to easily exploit both short and long distance relationship.

When comparing the results of U-Net to the ones obtained by the ReNet based network, we must distinguish between the two datasets that we employed. In fact, on CWFID2 U-Net resulted being the best architecture,

while U-ReNet was the best for the Broccoli dataset both in metrical and qualitative terms. On CWFID1 the best network resulted being the sliding window one, but U-ReNet reached an acceptable level of performance, whereas U-Net resulted being the worst one. As we said, for the Broccoli dataset it is very important to precisely identify the whole head so to be able to measure its dimension.

We can therefore conclude that the recurrent layers are better suited for the Broccoli dataset, since they can easily detect the correct shape of the plants, whereas, for U-Net, it is a bit harder. Moreover, U-ReNet reached acceptable level of performance on every dataset. Therefore it should be considered as a good standard solution on every dataset.

### 5.8.4 U-ReNet VS ReSeg

After having proposed the ReNet layer and the ReNet network [52] for image classification, the same authors proposed ReSeg [53], which basically stages an upsampling layer on top of 3 ReNet layers to perform image segmentation. This upsampling is a really demanding task, since it is required to reconstruct a full scale segmentation image from a severely downsampled image. On the other hand, the equivalent problem for the convolutional networks has been solved in many ways, one of which is the U-Net architecture. We have thus proposed the U-ReNet architecture, where the ReNet layer is used in an U-shaped architecture, and, therefore, we are interested in comparing this upsampling strategy to the simpler one adopted in ReSeg.

From qualitative analysis, it resulted evident that the U-ReNet architecture performs a better job in the upsampling path, producing smoother boundaries. Moreover, this advantage does not come at the expense of a considerably greater computational complexity. In fact, the greater time employed by the U-ReNet architecture to produce an annotation probably comes, in its greatest part, from the fact that it has more ReNet layers w.r.t. ReSeg. This ability to produce smoother boundaries can be important in tasks where it is needed to accurately determine the shapes of the objects, such as in the Broccoli dataset, and it also allows to reach better metrical results on CWFID.

### 5.8.5 U-ReNet VS U-ReNet2

We have implemented two versions of U-ReNet: in the first one we have used the same number of hidden units for all the layers of the network, whereas in the second one we have adopted the strategy that is usually employed with convolutional networks, i.e. increase the number of neurons in deeper layers.

It is interesting to see whether this strategy also works with the ReNet layers, since the two networks have a similar number of parameters, but distributed in a really different way across the layers.

Experimental results confirmed the applicability of this strategy to recurrent networks. In fact, the two networks always obtained comparable results, but ReNet2 resulted being faster both during training and prediction. It is therefore recommended to always adapt this scaling strategy.

### 5.8.6 ReSeg VS ReConv

The authors of ReSeg showed that its results can be improved by stacking it on top of pre trained convolutional layers. This was not the case in our experiments, which can be explained in two ways:

- as a first hypothesis, it could be that our convolutional network does not extract sufficiently abstract features that can be used by the ReNet layers;
- in alternative, it could be that the factorization of the convolutions makes our features unsuitable for the ReNet layers.

The first hypothesis is the least interesting, since it implies that the advantage from using convolutional layers just depends on the specific task and thus one just needs to try it. Conversely, the second hypothesis is more intriguing and absolutely demands further research.

### 5.8.7 ReConv VS ReConcat

The main weakness of ReConv resides in the upsampling strategy that from a really small image has to go back to the original size. We experimented whether the addition of a concatenation layer that unites the input image with the upsampled output before computing the actual output of the network can help to ease this process.

Given the bad results obtained by ReConv, it is difficult to say whether this strategy was really helpful and, therefore, it requires further analysis.

# Chapter 6

# Conclusions and future developments

*"Mia: Don't you hate that?*
*Vincent: What?*
*Mia: Uncomfortable silences. Why do we feel it's necessary to yak about bullshit in order to be comfortable?*
*Vincent: I don't know. That's a good question.*
*Mia: That's when you know you've found somebody special. When you can just shut the fuck up for a minute and comfortably enjoy the silence."*

Pulp Fiction

In the present work we have compared several neural networks in the task of agricultural image segmentation. We have considered both the problem of weed detection and automated harvesting. For the first problem we have used CWFID [15], while for the latter we extracted 300 2D images from the Broccoli dataset [22] and manually annotated each one of them.

In order to have a baseline to compare against, we implemented a pipeline similar to the one proposed in [16], incrementing the number of classifiers and proposing a faster smoothing algorithm.

We have compared the sliding window approach to the direct, full-size image segmentation implementing networks as the one proposed in [36] and U-Net [37]. We have compared convolutional networks to recurrent ones, using the ReNet layer [52] and the ReSeg network [53]. We have proposed a novel architecture that mixes the U-shaped structure of U-Net with the ReNet layer in order to see whether it could achieve better upsampling results than ReSeg. For the ReSeg network, we have also tested it when stacked on top of some convolutional layers taken from U-Net. In order to produce more accurate segmentations, we have also tested whether concatenating

the input image with the upsampled annotation before computing the real output eases the upsampling process.

## 6.1 Summary of obtained results

Experimental results shown that, using just 20 training images, the baselines achieve slightly better results than the neural networks. Conversely, with 40 images the neural networks are already able to achieve consistently better results, while the baseline classifiers are unable to sufficiently exploit the additional data. Moreover, the neural networks are able to produce segmentations in a considerably shorter time.

The sliding window approach resulted convenient in a really specific situation: the recognition of small weeds when few training images are available and there is an efficient way to preprocess the images so to remove soil. In fact, being tile-based it is not really good at correctly detecting the real shape of broccoli heads, which is important to recognize their growth stage. The reached performance also depends on the size the tiles, which is an important hyperparameter. Moreover, when run on the entire images, the sliding window approach becomes extremely slow due to the severe overlapping of tiles. Like the baseline classifiers, it did not achieve a consistent performance increase from the additional training data.

The recurrent networks resulted being slower than the convolutional ones. Moreover, their ability to identify long distance relationship did not result in consistently greater accuracies, since the U-Net architecture can also exploit them. It is however to notice that our U-ReNet architecture achieved better results in the reconstruction of broccoli heads shapes, which also translated in higher metrical results. It is also to notice that U-ReNet reached good results on both the splits of CWFID, thus resulting a well balanced architecture.

Our U-ReNet architecture outperformed the ReSeg one, producing more accurate results both from a metrical and a qualitative point of view. Moreover, we have seen that, by varying the number of hidden units in the recurrent layers, the U-ReNet architecture can be made faster.

The combined use of U-Net and ReSeg did not result in positive results, and the reason needs to be further investigated.

We can thus conclude that, for agricultural images segmentation tasks with deep neural networks, U-Net and U-ReNet are the most promising architectures, whereas the sliding window approach should be preferred on small datasets for weed detection.

## 6.2 Future developments

From a practical point of view, this work needs to be extended so to be able to distinguish different kind of weeds and to estimate the growth status of the crops. This will require to extend the manual annotation to include this new data. For the weed recognition problem, this will be done on a new bigger dataset that is currently being published.

During the networks evaluation we have tried to visualize the extracted features, which proved to be hard on the recurrent networks. This was due to the fact that the gradient of a pixel is related to the activation of many, possibly distant, neurons. Future experiments should thus try to modify the backpropagation of recurrent layers so to produce better looking gradients and make easier the interpretation.

Moreover, from the qualitative analysis of the produced segmentation images, we have noticed that the same metrical result can be obtained in really different ways, it should therefore be interesting to investigate whether a different metric would solve this problem.

We have also tried to embed the depth channel, obtaining negative results. This is probably due to the fact that our architectures just stacked the depth information as a new image channel. Future extensions should try to implement an architecture where the depth channel is kept separate and see whether this allows to obtain an increase in the accuracy.

Moreover, our experiments can be summarized by saying that, for agricultural segmentation tasks, the best suited architectures are U-Net and U-ReNet. It would therefore be interesting to create a novel architecture that mixes the two approaches. More specifically, we are thinking about an U-shaped network where, at each level, are present both convolutional and recurrent neurons. By building two parallel networks, one working on color images and one on the depth channel, and combining them when producing the final annotation, it should be possible to obtain optimal results on every task, while also partially maintaining the speed of the convolutional layers.

Another interesting development regards the Broccoli dataset. Its images, in fact, are regularly overlapping. This information could be exploited by using several images to detect the same broccoli head. A similar approach has been implemented in [22], by discarding points that are detected as broccoli heads in a low number of frames. The same approach can be easily applied to our networks, but more complex ones could also be interesting. For example, multiple frames could be simultaneously given to a single network to produce the annotation for the central frame, thus effectively evaluating the same pixels from different perspectives.

# Bibliography


[1] Edward H Adelson et al. "Pyramid methods in image processing". In: *RCA engineer* 29.6 (1984), pp. 33–41.

[2] TW Berge, AH Aastveit, and H Fykse. "Evaluation of an algorithm for automatic detection of broad-leaved weeds in spring cereals". In: *Precision Agriculture* 9.6 (2008), pp. 391–405.

[3] Christopher M Bishop. "Pattern recognition". In: *Machine Learning* 128 (2006).

[4] Ralph B Brown and Scott D Noble. "Site-specific weed management: sensing requirements-what do we need to see?" In: *Weed Science* 53.2 (2005), pp. 252–258.

[5] Beibei Cheng and Eric T Matson. "A Feature-Based Machine Learning Agent for Automatic Rice and Weed Discrimination". In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2015, pp. 517–527.

[6] *Complete Guide to Parameter Tuning in XGBoost*. URL: `https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/`.

[7] *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: `http://cs231n.stanford.edu/`.

[8] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[9] Filip Feyaerts and Luc Van Gool. "Multi-spectral vision system for weed detection". In: *Pattern Recognition Letters* 22.6 (2001), pp. 667–674.

[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[11] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[12] José Miguel Guerrero et al. “Support vector machines for crop/weeds identification in maize fields”. In: *Expert Systems with Applications* 39.12 (2012), pp. 11149–11155.

[13] David Hall et al. “Evaluation of features for leaf classification in challenging conditions”. In: *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE. 2015, pp. 797–804.

[14] Bharath Hariharan et al. “Hypercolumns for object segmentation and fine-grained localization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 447–456.

[15] Sebastian Haug and Jörn Ostermann. “A crop/weed field image dataset for the evaluation of computer vision based precision agriculture tasks”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 105–116.

[16] Sebastian Haug et al. “Plant classification system for crop/weed discrimination without segmentation”. In: *IEEE Winter Conference on Applications of Computer Vision*. IEEE. 2014, pp. 1142–1149.

[17] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[18] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[21] Neeraj Kumar et al. “Leafsnap: A computer vision system for automatic plant species identification”. In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 502–516.

[22] Keerthy Kusumam et al. “Can you pick a broccoli? 3D-vision based detection and localisation of broccoli heads in the field”. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 646–651.

[23] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[24] Won Suk Lee, DC Slaughter, and DK Giles. “Robotic weed control system for tomatoes”. In: *Precision Agriculture* 1.1 (1999), pp. 95–113.

[25] P Lottes et al. “An Effective Classification System for Separating Sugar Beets and Weeds for Precision Farming Applications”. In: *Proceedings of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2016.

[26] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[27] GE Meyer et al. “Textural imaging and discriminant analysis for distiguishing weeds for spot spraying”. In: *Transactions of the ASAE* 41.4 (1998), p. 1189.

[28] Tom M Mitchell et al. *Machine learning*. 1997.

[29] Michael Nielsen et al. “Detecting leaf features for automatic weed control using trinocular stereo vision.” In: *Proceedings of the 7th International Conference on Precision Agriculture and Other Precision Resources Management, Hyatt Regency, Minneapolis, MN, USA, 25-28 July, 2004*. Precision Agriculture Center, University of Minnesota, Department of Soil, Water and Climate. 2004, pp. 1016–1031.

[30] Timo Ojala and Matti Pietikäinen. “Unsupervised texture segmentation using feature distributions”. In: *Pattern Recognition* 32.3 (1999), pp. 477–486.

[31] Nobuyuki Otsu. “A threshold selection method from gray-level histograms”. In: *Automatica* 11.285-296 (1975), pp. 23–27.

[32] John Platt Patrice Y. Simard Dave Steinkraus. “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis”. In: Institute of Electrical and Electronics Engineers, Inc., 2003. URL: `https://www.microsoft.com/en-us/research/publication/best-`

practices-for-convolutional-neural-networks-applied-to-visual-document-analysis/.

[33] Alexis Piron, F van Der Heijden, and Marie-France Destain. "Weed detection in 3D images". In: *Precision agriculture* 12.5 (2011), pp. 607–622.

[34] Alexis Piron et al. "Selection of the most efficient wavelength bands for discriminating weeds from crop". In: *Computers and Electronics in Agriculture* 62.2 (2008), pp. 141–148.

[35] John Platt et al. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74.

[36] C. Potena, D. Nardi, and A. Pretto. "Fast and Accurate Crop and Weed Identification with Summarized Train Sets for Precision Agriculture". In: *Proc. of 14th International Conference on Intelligent Autonomous Systems (IAS-14)*. 2016.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *MICCAI*. 2015.

[39] Bryan C Russell et al. "LabelMe: a database and web-based tool for image annotation". In: *International journal of computer vision* 77.1-3 (2008), pp. 157–173.

[40] AJ Sachez and John A Marchant. "Fusing 3D information for crop/weeds classification". In: *Pattern Recognition, 2000. Proceedings. 15th International Conference on*. Vol. 4. IEEE. 2000, pp. 295–298.

[41] IM Scotford and PCH Miller. "Applications of spectral reflectance techniques in northern European cereal production: a review". In: *Biosystems engineering* 90.3 (2005), pp. 235–250.

[42] Scott A Shearer and RG Holmes. "Plant identification using color co-occurrence matrices". In: *Transactions of the ASAE* 33.6 (1990), pp. 1237–1244.

[43] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *CoRR* abs/1312.6034 (2013). URL: http://arxiv.org/abs/1312.6034.

[44] DC Slaughter, DK Giles, and D Downey. "Autonomous robotic weed control systems: A review". In: *Computers and electronics in agriculture* 61.1 (2008), pp. 63–78.

[45] Oskar Söderkvist. "Computer vision classification of leaves from swedish trees". In: (2001).

[46] Henning Tangen Søgaard. "Weed classification by active shape models". In: *Biosystems engineering* 91.3 (2005), pp. 271–281.

[47] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: *CoRR* abs/1412.6806 (2014). URL: http://arxiv.org/abs/1412.6806.

[48] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[49] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[50] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). URL: http://arxiv.org/abs/1512.00567.

[51] JF Thompson, JV Stafford, and PCH Miller. "Potential for automatic weed detection and selective herbicide application". In: *Crop Protection* 10.4 (1991), pp. 254–259.

[52] Francesco Visin et al. "ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks". In: *CoRR* abs/1505.00393 (2015). URL: http://arxiv.org/abs/1505.00393.

[53] Francesco Visin et al. "ReSeg: A Recurrent Neural Network for Object Segmentation". In: *CoRR* abs/1511.07053 (2015). URL: http://arxiv.org/abs/1511.07053.

[54] Stephen Gang Wu et al. "A leaf recognition algorithm for plant classification using probabilistic neural network". In: *2007 IEEE international symposium on signal processing and information technology*. IEEE. 2007, pp. 11–16.

[55] Jason Yosinski et al. “Understanding neural networks through deep visualization”. In: *arXiv preprint arXiv:1506.06579* (2015).

[56] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

# Appendix A

# Tuning of network's hyperparameters

In the following pages we report the results of the hyperparameter selection stage for the neural networks. In all the following tables the word *loss* is meant as *loss* - $l_2$ *loss* so that it is possible to evaluate the true loss of the network without caring for the weight decay term.

## A.1 Learning rate selection

We started by doing a single epoch to select the learning rate:

| Network | Dataset | LR | Train loss | Val loss |
|---|---|---|---|---|
| SWind | CWFID1 | 1e-8 | 3.272 | 3.506 |
| | | 1e-7 | 3.121 | 3.174 |
| | | 1e-6 | 2.041 | 1.278 |
| | | 1e-5 | 0.942 | 1.054 |
| | | 1e-4 | 0.682 | 0.978 |
| | | **1e-3** | **0.639** | **0.907** |
| | | 1e-2 | 3.792 | 1.210 |
| | CWFID2 | 1e-7 | 1.643 | 1.512 |
| | | 1e-6 | 1.006 | 0.875 |
| | | 1e-5 | 0.695 | 0.486 |
| | | **1e-4** | **0.583** | **0.390** |
| | | 1e-3 | 0.599 | 0.463 |
| | | 1e-2 | 2.459 | 1.007 |
| | | 1e-1 | 90.321 | 101.071 |
| | Broccoli | 1e-7 | 1.686 | 1.597 |
| | | 1e-6 | 0.977 | 0.618 |
| | | 1e-5 | 0.479 | 0.361 |
| | | **1e-4** | **0.354** | **0.311** |
| | | 1e-3 | 0.379 | 0.381 |
| | | 1e-2 | 3.621 | 0.631 |
| | | 1e-1 | 60.153 | 0.623 |
| UNet | CWFID1 | 1e-8 | 1.072 | 1.071 |
| | | 1e-7 | 1.072 | 1.071 |
| | | 1e-6 | 1.069 | 1.065 |
| | | 1e-5 | 1.012 | 0.999 |
| | | 1e-4 | 0.790 | 0.390 |
| | | **1e-3** | **0.684** | **0.345** |
| | | 1e-2 | 7.997 | 9.210 |
| | CWFID2 | 1e-7 | 1.114 | 1.106 |
| | | 1e-6 | 1.107 | 1.094 |
| | | 1e-5 | 0.905 | 0.687 |
| | | 1e-4 | 0.591 | 0.443 |
| | | **1e-3** | **0.451** | **0.413** |
| | | 1e-2 | 8.447 | 9.210 |
| | | 1e-1 | nan | nan |
| | | 1e0 | nan | nan |
| | Broccoli | 1e-7 | 1.042 | 1.029 |
| | | 1e-6 | 0.932 | 0.852 |
| | | 1e-5 | 0.532 | 0.245 |
| | | 1e-4 | 0.292 | 0.112 |
| | | **1e-3** | **0.143** | **0.093** |
| | | 1e-2 | 0.931 | 0.931 |
| | | 1e-1 | 0.933 | 0.931 |
| | | 1e0 | nan | nan |
| | BroccoliD | 1e-8 | 0.422 | 0.417 |
| | | 1e-7 | 0.420 | 0.414 |
| | | 1e-6 | 0.401 | 0.372 |
| | | 1e-5 | 0.224 | 0.173 |
| | | 1e-4 | 0.129 | 0.041 |
| | | **1e-3** | **0.091** | **0.051** |
| | | 1e-2 | 0.915 | 0.931 |
| | | 1e-1 | 0.929 | 0.931 |
| | | 1e0 | nan | nan |
| U-ReNet | CWFID1 | 1e-8 | 1.076 | 1.072 |
| | | 1e-7 | 1.075 | 1.071 |
| | | 1e-6 | 1.070 | 1.060 |
| | | 1e-5 | 0.991 | 0.823 |
| | | 1e-4 | 0.496 | 0.429 |
| | | **1e-3** | **0.443** | **0.369** |
| | | 1e-2 | 8.506 | 9.231 |
| | | 1e-1 | 9.007 | 9.210 |
| | | 1e0 | 11.265 | 11.626 |
| | CWFID2 | 1e-7 | 1.096 | 1.097 |
| | | 1e-6 | 1.091 | 1.088 |
| | | 1e-5 | 0.984 | 0.954 |
| | | 1e-4 | 0.477 | 0.444 |
| | | **1e-3** | **0.387** | **0.372** |
| | | 1e-2 | 10.794 | 12.924 |
| | | 1e-1 | 11.357 | 12.935 |
| | | 1e0 | 9.164 | 9.210 |
| | Broccoli | 1e-7 | 0.648 | 0.646 |
| | | 1e-6 | 0.630 | 0.601 |
| | | 1e-5 | 0.334 | 0.183 |
| | | 1e-4 | 0.147 | 0.081 |
| | | **1e-3** | **0.099** | **0.036** |
| | | 1e-2 | 0.850 | 0.931 |
| | | 1e-1 | 1.028 | 1.030 |
| | | 1e0 | 0.929 | 0.931 |
| | BroccoliD | 1e-8 | 0.693 | 0.693 |
| | | 1e-7 | 0.692 | 0.690 |
| | | 1e-6 | 0.674 | 0.655 |
| | | 1e-5 | 0.372 | 0.187 |
| | | 1e-4 | 0.180 | 0.085 |
| | | **1e-3** | **0.114** | **0.042** |
| | | 1e-2 | 0.922 | 0.931 |
| | | 1e-1 | 17.264 | 17.490 |
| | | 1e0 | 0.931 | 0.931 |
| U-ReNet2 | CWFID1 | 1e-8 | 1.104 | 1.104 |
| | | 1e-7 | 1.104 | 1.104 |
| | | 1e-6 | 1.103 | 1.102 |
| | | 1e-5 | 1.089 | 1.048 |
| | | 1e-4 | 0.580 | 0.430 |
| | | **1e-3** | **0.482** | **0.347** |
| | | 1e-2 | 0.925 | 0.411 |
| | | 1e-1 | 10.817 | 11.626 |
| | CWFID2 | 1e-7 | 1.091 | 1.099 |
| | | 1e-6 | 1.085 | 1.094 |
| | | 1e-5 | 1.010 | 1.086 |
| | | 1e-4 | 0.591 | 0.471 |
| | | **1e-3** | **0.424** | **0.455** |
| | | 1e-2 | 8.754 | 9.210 |
| | | 1e-1 | 11.370 | 12.935 |
| | | 1e0 | 11.438 | 12.935 |
| | Broccoli | 1e-7 | 0.665 | 0.663 |
| | | 1e-6 | 0.649 | 0.621 |
| | | 1e-5 | 0.304 | 0.148 |
| | | 1e-4 | 0.151 | 0.075 |
| | | **1e-3** | **0.101** | **0.038** |
| | | 1e-2 | 0.919 | 0.931 |
| | | 1e-1 | 17.267 | 17.490 |
| | | 1e0 | 0.931 | 0.931 |
| | BroccoliD | 1e-8 | 0.688 | 0.686 |
| | | 1e-7 | 0.686 | 0.684 |
| | | 1e-6 | 0.674 | 0.657 |
| | | 1e-5 | 0.372 | 0.189 |
| | | 1e-4 | 0.180 | 0.092 |
| | | **1e-3** | **0.115** | **0.049** |
| | | 1e-2 | 15.368 | 17.490 |
| | | 1e-1 | 17.148 | 17.490 |
| | | 1e0 | 0.931 | 0.931 |

| | | | | |
|---|---|---|---|---|
| ReSeg | CWFID1 | 1e-8 | 1.055 | 1.061 |
| | | 1e-7 | 1.053 | 1.057 |
| | | 1e-6 | 1.035 | 1.022 |
| | | 1e-5 | 0.904 | 0.781 |
| | | 1e-4 | 0.549 | 0.430 |
| | | 1e-3 | 0.463 | 0.514 |
| | | **1e-2** | **0.601** | **0.392** |
| | | 1e-1 | 10.804 | 11.626 |
| | CWFID2 | 1e-7 | 1.135 | 1.135 |
| | | 1e-6 | 1.082 | 1.062 |
| | | 1e-5 | 0.803 | 0.698 |
| | | 1e-4 | 0.484 | 0.474 |
| | | **1e-3** | **0.412** | **0.328** |
| | | 1e-2 | 0.551 | 0.346 |
| | | 1e-1 | 11.281 | 12.935 |
| | | 1e0 | nan | nan |
| | Broccoli | 1e-7 | 0.771 | 0.753 |
| | | 1e-6 | 0.693 | 0.612 |
| | | 1e-5 | 0.380 | 0.211 |
| | | 1e-4 | 0.139 | 0.079 |
| | | **1e-3** | **0.071** | **0.041** |
| | | 1e-2 | 0.227 | 0.163 |
| | | 1e-1 | nan | nan |
| | | 1e0 | nan | nan |
| | BroccoliD | 1e-8 | 1.393 | 1.385 |
| | | 1e-7 | 1.369 | 1.338 |
| | | 1e-6 | 1.154 | 0.940 |
| | | 1e-5 | 0.456 | 0.245 |
| | | 1e-4 | 0.220 | 0.139 |
| | | 1e-3 | 0.106 | 0.054 |
| | | **1e-2** | **0.088** | **0.039** |
| | | 1e-1 | 0.981 | 0.931 |
| | | 1e0 | nan | nan |
| ReConv | CWFID1 | 1e-8 | 1.615 | 1.559 |
| | | 1e-7 | 1.603 | 1.535 |
| | | 1e-6 | 1.492 | 1.325 |
| | | 1e-5 | 0.944 | 0.700 |
| | | 1e-4 | 0.648 | 0.478 |
| | | **1e-3** | **0.577** | **0.478** |
| | | 1e-2 | 1.220 | 0.984 |
| | | 1e-1 | 10.919 | 11.626 |
| | CWFID2 | 1e-8 | 1.131 | 1.069 |
| | | 1e-7 | 1.116 | 1.045 |
| | | 1e-6 | 1.005 | 0.923 |
| | | 1e-5 | 0.719 | 0.709 |
| | | 1e-4 | 0.532 | 0.468 |
| | | **1e-3** | **0.505** | **0.467** |
| | | 1e-2 | 1.166 | 1.554 |
| | | 1e-1 | 9.056 | 9.210 |
| | | 1e0 | 9.110 | 9.210 |
| | Broccoli | 1e-7 | 1.373 | 1.335 |
| | | 1e-6 | 0.924 | 0.594 |
| | | 1e-5 | 0.323 | 0.197 |
| | | 1e-4 | 0.177 | 0.113 |
| | | **1e-3** | **0.102** | **0.074** |
| | | 1e-2 | 0.212 | 0.182 |
| | | 1e-1 | 0.953 | 0.931 |
| | | 1e0 | 0.938 | 0.931 |
| | BroccoliD | 1e-8 | 1.297 | 1.328 |
| | | 1e-7 | 1.253 | 1.239 |
| | | 1e-6 | 0.914 | 0.659 |
| | | 1e-5 | 0.335 | 0.211 |
| | | 1e-4 | 0.187 | 0.123 |
| | | **1e-3** | **0.105** | **0.080** |
| | | 1e-2 | 0.179 | 0.102 |
| | | 1e-1 | 0.928 | 0.931 |
| | | 1e0 | 1.062 | 0.931 |

Basically the learning rate is 1e-3 everywhere, except for a couple of configurations. For the configurations having their best at 1e-4 (like the sliding window network) we kept it, as a lower learning rate does not prevent us from reaching an optimum, even if at the expense of a slower convergence. The configurations with an optimum in 1e-2 (like ReSeg) are more problematic as a too high learning rate can damage the learning process more easily. We have kept the optimum values everywhere, but, as we will se later, when further exploring the hyperparameters we have decided to lower the 1e-2 to 1e-3.

## A.2 $l_2$ term selection

After having selected the learning rate, we reset the network to perform a single epoch for the $l_2$ term. The weight decay term is not really useful to control overfitting, but instead it is used as it helps to avoid instability in the last training epochs [53].

| Network | Dataset | $l_2$ | Train loss | Val loss |
|---|---|---|---|---|
| Deep1 | CWFID1 | 1e-8 | 0.784 | 1.087 |
| | | 1e-7 | 0.675 | 1.273 |
| | | 1e-6 | 0.676 | 0.921 |
| | | **1e-5** | **0.689** | **0.899** |
| | | 1e-4 | 0.720 | 1.106 |
| | | 1e-3 | 0.726 | 1.086 |
| | | 1e-2 | 0.691 | 1.004 |
| | | 1e-1 | 0.727 | 0.954 |
| | | 1e0 | 0.799 | 0.890 |
| | CWFID2 | 1e-7 | 0.638 | 0.433 |
| | | 1e-6 | 0.643 | 0.426 |
| | | 1e-5 | 0.652 | 0.427 |
| | | **1e-4** | **0.784** | **0.557** |
| | | 1e-3 | 2.096 | 1.902 |
| | | 1e-2 | 14.781 | 14.027 |
| | | 1e-1 | 118.262 | 95.161 |
| | | 1e0 | 901.069 | 649.761 |
| | Broccoli | 1e-7 | 0.357 | 0.314 |
| | | 1e-6 | 0.349 | 0.310 |
| | | 1e-5 | 0.366 | 0.335 |
| | | **1e-4** | **0.498** | **0.469** |
| | | 1e-3 | 1.805 | 1.756 |
| | | 1e-2 | 14.007 | 12.899 |
| | | 1e-1 | 99.520 | 67.554 |
| | | 1e0 | 731.302 | 479.974 |
| Deep2 | CWFID1 | **1e-8** | **0.482** | **0.328** |
| | | 1e-7 | 0.583 | 0.335 |
| | | 1e-6 | 0.683 | 0.333 |
| | | 1e-5 | 0.534 | 0.342 |
| | | 1e-4 | 0.618 | 0.356 |
| | | 1e-3 | 0.617 | 0.360 |
| | | 1e-2 | 0.604 | 0.347 |
| | | 1e-1 | 0.935 | 0.759 |
| | | 1e0 | 1.065 | 1.056 |
| | CWFID2 | 1e-7 | 0.403 | 0.387 |
| | | 1e-6 | 0.477 | 0.406 |
| | | **1e-5** | **0.528** | **0.406** |
| | | 1e-4 | 0.855 | 0.696 |
| | | 1e-3 | 3.301 | 2.465 |
| | | 1e-2 | 19.220 | 9.894 |
| | | 1e-1 | 116.132 | 26.605 |
| | | 1e0 | 946.900 | 151.710 |
| | Broccoli | 1e-7 | 0.106 | 0.024 |
| | | 1e-6 | 0.112 | 0.025 |
| | | 1e-5 | 0.122 | 0.030 |
| | | **1e-4** | **0.123** | **0.030** |
| | | 1e-3 | 0.126 | 0.037 |
| | | 1e-2 | 0.130 | 0.047 |
| | | 1e-1 | 0.127 | 0.033 |
| | | 1e0 | 0.131 | 0.035 |
| | BroccoliD | 1e-8 | 0.102 | 0.043 |
| | | 1e-7 | 0.118 | 0.037 |
| | | 1e-6 | 0.112 | 0.057 |
| | | 1e-5 | 0.108 | 0.039 |
| | | 1e-4 | 0.132 | 0.110 |
| | | 1e-3 | 0.120 | 0.099 |
| | | **1e-2** | **0.116** | **0.034** |
| | | 1e-1 | 0.115 | 0.069 |
| | | 1e0 | 0.111 | 0.037 |
| U-ReNet | CWFID1 | 1e-8 | 0.436 | 0.343 |
| | | **1e-7** | **0.486** | **0.335** |
| | | 1e-6 | 0.530 | 0.354 |
| | | 1e-5 | 0.496 | 0.350 |
| | | 1e-4 | 0.508 | 0.337 |
| | | 1e-3 | 0.499 | 0.341 |
| | | 1e-2 | 0.633 | 0.378 |
| | | 1e-1 | 1.014 | 0.985 |
| | | 1e0 | 1.063 | 1.052 |
| | CWFID2 | 1e-7 | 0.424 | 0.394 |
| | | 1e-6 | 0.497 | 0.385 |
| | | 1e-5 | 0.597 | 0.524 |
| | | **1e-4** | **1.255** | **0.931** |
| | | 1e-3 | 5.101 | 2.490 |
| | | 1e-2 | 28.719 | 6.325 |
| | | 1e-1 | 231.670 | 30.104 |
| | | 1e0 | 2148.754 | 219.669 |
| | Broccoli | 1e-7 | 0.114 | 0.043 |
| | | 1e-6 | 0.123 | 0.042 |
| | | 1e-5 | 0.122 | 0.059 |
| | | 1e-4 | 0.133 | 0.042 |
| | | **1e-3** | **0.121** | **0.039** |
| | | 1e-2 | 0.144 | 0.057 |
| | | 1e-1 | 0.119 | 0.044 |
| | | 1e0 | 0.131 | 0.063 |
| | BroccoliD | **1e-8** | **0.105** | **0.040** |
| | | 1e-7 | 0.137 | 0.085 |
| | | 1e-6 | 0.111 | 0.052 |
| | | 1e-5 | 0.108 | 0.045 |
| | | 1e-4 | 0.125 | 0.076 |
| | | 1e-3 | 0.118 | 0.070 |
| | | 1e-2 | 0.113 | 0.053 |
| | | 1e-1 | 0.100 | 0.043 |
| | | 1e0 | 0.117 | 0.071 |
| U-ReNet2 | CWFID1 | 1e-8 | 0.523 | 0.366 |
| | | **1e-7** | **0.663** | **0.350** |
| | | 1e-6 | 0.622 | 0.372 |
| | | 1e-5 | 0.650 | 0.357 |
| | | 1e-4 | 0.691 | 0.379 |
| | | 1e-3 | 0.664 | 0.357 |
| | | 1e-2 | 0.712 | 0.375 |
| | | 1e-1 | 0.990 | 0.936 |
| | | 1e0 | 1.070 | 1.062 |
| | CWFID2 | 1e-7 | 0.462 | 0.424 |
| | | 1e-6 | 0.499 | 0.432 |
| | | 1e-5 | 0.623 | 0.505 |
| | | **1e-4** | **1.242** | **0.903** |
| | | 1e-3 | 4.901 | 2.410 |
| | | 1e-2 | 27.675 | 5.835 |
| | | 1e-1 | 222.747 | 27.270 |
| | | 1e0 | 2079.762 | 209.561 |

| | | | | |
|---|---|---|---|---|
| U-ReNet2 | Broccoli | 1e-7 | 0.122 | 0.061 |
| | | 1e-6 | 0.146 | 0.059 |
| | | 1e-5 | 0.159 | 0.072 |
| | | **1e-4** | **0.152** | **0.045** |
| | | 1e-3 | 0.152 | 0.066 |
| | | 1e-2 | 0.158 | 0.062 |
| | | 1e-1 | 0.143 | 0.064 |
| | | 1e0 | 0.156 | 0.086 |
| | BroccoliD | **1e-8** | **0.099** | **0.042** |
| | | 1e-7 | 0.111 | 0.055 |
| | | 1e-6 | 0.109 | 0.064 |
| | | 1e-5 | 0.124 | 0.069 |
| | | 1e-4 | 0.119 | 0.078 |
| | | 1e-3 | 0.117 | 0.074 |
| | | 1e-2 | 0.119 | 0.060 |
| | | 1e-1 | 0.114 | 0.069 |
| | | 1e0 | 0.113 | 0.093 |
| ReSeg | CWFID1 | 1e-8 | 0.630 | 0.518 |
| | | 1e-7 | 0.563 | 0.659 |
| | | **1e-6** | **0.467** | **0.380** |
| | | 1e-5 | 0.502 | 0.406 |
| | | 1e-4 | 0.486 | 0.597 |
| | | 1e-3 | 0.532 | 0.505 |
| | | 1e-2 | 0.550 | 0.454 |
| | | 1e-1 | 0.868 | 0.982 |
| | CWFID2 | 1e-7 | 0.395 | 0.334 |
| | | 1e-6 | 0.407 | 0.344 |
| | | 1e-5 | 0.519 | 0.488 |
| | | **1e-4** | **1.308** | **1.077** |
| | | 1e-3 | 6.327 | 3.617 |
| | | 1e-2 | 33.796 | 7.206 |
| | | 1e-1 | 254.592 | 28.851 |
| | | 1e0 | 2384.357 | 227.877 |
| | Broccoli | 1e-7 | 0.077 | 0.037 |
| | | 1e-6 | 0.073 | 0.044 |
| | | 1e-5 | 0.074 | 0.041 |
| | | **1e-4** | **0.072** | **0.037** |
| | | 1e-3 | 0.072 | 0.038 |
| | | 1e-2 | 0.074 | 0.040 |
| | | 1e-1 | 0.075 | 0.048 |
| | | 1e0 | 0.074 | 0.048 |
| | BroccoliD | 1e-8 | 0.192 | 0.087 |
| | | **1e-7** | **0.109** | **0.044** |
| | | 1e-6 | 0.210 | 0.183 |
| | | 1e-5 | 0.206 | 0.191 |
| | | 1e-4 | 0.221 | 0.198 |
| | | 1e-3 | 0.207 | 0.198 |
| | | 1e-2 | 0.206 | 0.191 |
| | | 1e-1 | 0.214 | 0.194 |
| | | 1e0 | 0.210 | 0.186 |
| ReConv | CWFID1 | **1e-8** | **0.527** | **0.409** |
| | | 1e-7 | 0.524 | 0.515 |
| | | 1e-6 | 0.509 | 0.438 |
| | | 1e-5 | 0.522 | 0.415 |
| | | 1e-4 | 0.550 | 0.414 |
| | | 1e-3 | 0.566 | 0.384 |
| | | 1e-2 | 0.577 | 0.480 |
| | | 1e-1 | 0.732 | 0.721 |
| | | 1e0 | 0.979 | 1.050 |
| | CWFID2 | **1e-8** | **0.515** | **0.431** |
| | | 1e-7 | 0.528 | 0.505 |
| | | 1e-6 | 0.576 | 0.534 |
| | | 1e-5 | 0.752 | 0.709 |
| | | 1e-4 | 2.574 | 2.307 |
| | | 1e-3 | 12.810 | 7.786 |
| | | 1e-2 | 46.822 | 8.059 |
| | | 1e-1 | 230.377 | 4.685 |
| | | 1e0 | 2047.240 | 19.388 |
| ReConv | Broccoli | 1e-7 | 0.084 | 0.081 |
| | | 1e-6 | 0.084 | 0.071 |
| | | 1e-5 | 0.091 | 0.071 |
| | | 1e-4 | 0.084 | 0.070 |
| | | **1e-3** | **0.088** | **0.067** |
| | | 1e-2 | 0.083 | 0.080 |
| | | 1e-1 | 0.084 | 0.072 |
| | | 1e0 | 0.082 | 0.070 |
| | BroccoliD | 1e-8 | 0.085 | 0.072 |
| | | 1e-7 | 0.084 | 0.070 |
| | | 1e-6 | 0.081 | 0.065 |
| | | **1e-5** | **0.084** | **0.064** |
| | | 1e-4 | 0.081 | 0.075 |
| | | 1e-3 | 0.081 | 0.064 |
| | | 1e-2 | 0.089 | 0.065 |
| | | 1e-1 | 0.088 | 0.072 |

As visible, the weight decay term does not have a great influence on the overfitting and lower values are generally better.

## A.3 Coarse hyperparameter tuning

Using the selected values for the learning rate and the $l_2$ term, we have performed 3 epochs of training choosing all the hyperparameters at random on a broad scale. Since the sliding window network has a set of hyperparameters different from the other networks, we report it separately.

| Dataset | GS | OD | D1D | D2D | EL_S | EL_A | AR | Train loss | Val loss |
|---|---|---|---|---|---|---|---|---|---|
| CWFID1 | 0.69 | 0.86 | 0.30 | 0.19 | 35.01 | 31.84 | 0.08 | 0.851 | 7.940 |
| | 0.35 | 0.81 | 0.35 | 0.48 | 99.35 | 55.65 | 0.12 | 0.841 | 34.890 |
| | **0.08** | **0.88** | **0.36** | **0.38** | **63.75** | **81.10** | **0.04** | **0.814** | **1.032** |
| | **0.30** | **0.93** | **0.27** | **0.25** | **7.54** | **41.82** | **0.69** | **0.919** | **0.963** |
| | 0.60 | 0.64 | 0.54 | 0.36 | 3.95 | 47.49 | 0.48 | 0.847 | 16.613 |
| | 0.65 | 0.28 | 0.51 | 0.29 | 20.77 | 81.45 | 0.61 | 0.852 | 16.329 |
| | 0.71 | 0.61 | 0.48 | 0.14 | 10.21 | 46.58 | 0.10 | 0.810 | 4.747 |
| | 0.75 | 0.46 | 0.10 | 0.79 | 99.64 | 52.14 | 0.21 | 0.847 | 137.385 |
| | **0.24** | **0.58** | **0.13** | **0.42** | **36.60** | **94.12** | **0.82** | **0.901** | **0.974** |
| | 0.83 | 0.31 | 0.81 | 0.38 | 45.07 | 64.70 | 0.00 | 0.824 | 18.374 |
| | 0.42 | 0.20 | 0.50 | 0.73 | 18.47 | 56.52 | 0.21 | 0.845 | 58.103 |
| | 0.95 | 0.73 | 0.52 | 0.02 | 19.75 | 5.80 | 0.95 | 0.895 | 24.401 |
| | 0.46 | 0.61 | 0.55 | 0.53 | 78.49 | 20.61 | 0.14 | 0.845 | 34.013 |
| | 0.24 | 0.89 | 0.03 | 0.39 | 61.78 | 64.19 | 0.06 | 0.845 | 16.106 |
| | 0.41 | 0.13 | 0.46 | 0.42 | 51.73 | 10.71 | 0.51 | 0.847 | 14.929 |
| | **0.32** | **0.07** | **0.13** | **0.23** | **18.31** | **57.41** | **0.72** | **0.850** | **0.942** |
| | 0.67 | 0.65 | 0.34 | 0.88 | 90.73 | 55.82 | 0.59 | 0.851 | 117.816 |
| | 0.39 | 0.05 | 0.08 | 0.07 | 25.84 | 42.53 | 0.11 | 0.702 | 3.132 |
| | 0.51 | 0.57 | 0.18 | 0.46 | 73.33 | 8.59 | 0.82 | 0.883 | 2.477 |
| | 0.11 | 0.86 | 0.05 | 0.96 | 10.49 | 81.28 | 0.03 | 0.860 | 105.262 |
| CWFID2 | **0.24** | **0.37** | **0.24** | **0.74** | **5.75** | **81.88** | **0.90** | **1.066** | **2.309** |
| | 0.59 | 0.32 | 0.57 | 0.42 | 11.60 | 27.74 | 0.13 | 1.020 | 3.404 |
| | 0.43 | 0.75 | 0.53 | 0.82 | 20.20 | 83.52 | 0.06 | 1.145 | 3.962 |
| | 0.60 | 0.65 | 0.20 | 0.78 | 66.07 | 98.74 | 0.91 | 1.130 | 2.735 |
| | 0.38 | 0.27 | 0.76 | 0.44 | 60.25 | 93.23 | 0.85 | 1.174 | 2.350 |
| | **0.13** | **0.06** | **0.58** | **0.66** | **50.74** | **93.56** | **0.34** | **1.046** | **1.495** |
| | 0.26 | 0.44 | 0.19 | 0.55 | 4.18 | 18.28 | 0.23 | 0.988 | 3.861 |
| | 0.68 | 0.07 | 0.87 | 0.92 | 51.83 | 55.68 | 0.49 | 1.274 | 4.475 |
| | 0.69 | 0.70 | 0.90 | 0.97 | 86.74 | 20.84 | 0.00 | 2.102 | 3.359 |
| | 0.15 | 0.58 | 0.83 | 0.48 | 98.92 | 55.36 | 0.34 | 2.379 | 2.482 |
| | 0.51 | 0.22 | 0.82 | 0.79 | 98.61 | 45.56 | 0.98 | 2.370 | 2.624 |
| | 0.98 | 0.86 | 0.20 | 0.68 | 80.88 | 0.39 | 0.35 | 1.224 | 4.375 |
| | 0.64 | 0.59 | 0.76 | 0.53 | 58.98 | 81.65 | 0.15 | 1.592 | 5.547 |
| | 0.27 | 0.52 | 0.13 | 0.19 | 29.44 | 49.92 | 0.09 | 1.008 | 4.778 |
| | 0.86 | 0.12 | 0.20 | 0.57 | 94.02 | 22.67 | 0.86 | 1.067 | 3.006 |
| | 0.14 | 0.65 | 0.59 | 0.35 | 59.34 | 29.64 | 0.05 | 1.117 | 2.860 |
| | 0.38 | 0.16 | 0.98 | 0.05 | 91.69 | 10.87 | 0.27 | 2.096 | 3.687 |
| | 0.57 | 0.35 | 0.49 | 0.36 | 9.48 | 85.94 | 0.63 | 1.133 | 2.618 |
| | **0.10** | **0.87** | **0.55** | **0.14** | **14.59** | **15.92** | **0.49** | **1.237** | **1.111** |
| | **0.12** | **0.00** | **0.87** | **0.05** | **97.56** | **3.44** | **0.59** | **1.294** | **1.093** |
| Broccoli | **0.08** | **0.16** | **0.77** | **0.04** | **30.17** | **71.05** | **0.52** | **0.803** | **0.811** |
| | 0.16 | 0.04 | 0.49 | 0.35 | 76.90 | 21.28 | 0.64 | 0.788 | 1.682 |
| | 0.81 | 0.42 | 0.20 | 0.08 | 58.45 | 31.43 | 0.79 | 0.782 | 3.874 |
| | 0.84 | 0.36 | 0.34 | 0.43 | 0.81 | 78.76 | 0.37 | 0.781 | 7.616 |
| | 0.22 | 0.27 | 0.04 | 0.42 | 97.45 | 68.57 | 0.86 | 0.774 | 3.616 |
| | 0.19 | 0.69 | 0.42 | 0.28 | 67.77 | 8.96 | 0.74 | 0.819 | 1.564 |
| | 0.63 | 0.20 | 0.67 | 0.27 | 53.18 | 16.90 | 0.05 | 0.670 | 6.547 |
| | 0.23 | 0.28 | 0.79 | 0.46 | 19.09 | 22.75 | 0.84 | 0.872 | 1.856 |
| | 0.72 | 0.89 | 0.75 | 0.65 | 24.16 | 78.38 | 0.82 | 0.834 | 4.683 |
| | 0.43 | 0.80 | 0.53 | 0.85 | 51.07 | 23.86 | 0.19 | 0.825 | 6.617 |
| | 0.56 | 0.79 | 0.18 | 0.79 | 84.20 | 45.25 | 0.43 | 0.814 | 6.462 |
| | 0.73 | 0.24 | 0.42 | 0.04 | 29.88 | 31.87 | 0.88 | 0.794 | 5.151 |
| | **0.10** | **0.54** | **0.66** | **0.15** | **29.69** | **95.14** | **0.03** | **0.616** | **1.226** |
| | **0.02** | **0.38** | **0.36** | **0.04** | **24.06** | **68.45** | **0.06** | **0.507** | **0.501** |
| | 0.15 | 0.05 | 0.67 | 0.31 | 87.92 | 3.46 | 0.17 | 0.732 | 4.942 |
| | 0.28 | 0.69 | 0.21 | 0.09 | 65.56 | 42.17 | 0.82 | 0.792 | 4.264 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Broccoli | 0.27 | 0.87 | 0.51 | 0.61 | 99.84 | 28.37 | 0.69 | 0.825 | 5.948 |
| | 0.61 | 0.37 | 0.42 | 0.80 | 56.58 | 76.07 | 0.21 | 0.802 | 5.297 |
| | 0.39 | 0.82 | 0.20 | 0.16 | 25.14 | 68.68 | 0.40 | 0.797 | 7.553 |
| | 0.71 | 0.79 | 0.41 | 0.70 | 62.35 | 47.59 | 0.45 | 0.812 | 6.053 |

All the other networks are on the following table:

| Network | Dataset | GS | OD | EL_S | EL_A | AR | SR | Train loss | Val loss |
|---|---|---|---|---|---|---|---|---|---|
| UNet | CWFID1 | 0.61 | 0.36 | 48.57 | 97.36 | 0.44 | 55 | 0.886 | 0.795 |
| | | 0.34 | 0.83 | 81.24 | 91.51 | 0.45 | 16 | 0.900 | 0.850 |
| | | **0.63** | **0.29** | **89.79** | **52.25** | **0.03** | **44** | **0.321** | **0.324** |
| | | 0.04 | 0.43 | 13.44 | 59.07 | 0.86 | 64 | 0.962 | 0.986 |
| | | 0.31 | 0.44 | 5.12 | 50.57 | 0.65 | 15 | 0.935 | 0.932 |
| | | 0.49 | 0.79 | 92.85 | 47.20 | 0.55 | 21 | 0.938 | 0.987 |
| | | 0.82 | 0.19 | 60.56 | 79.30 | 0.82 | 87 | 0.958 | 0.940 |
| | | 0.65 | 0.61 | 82.04 | 27.72 | 0.56 | 28 | 0.938 | 0.887 |
| | | 0.81 | 0.47 | 85.53 | 50.77 | 0.55 | 26 | 0.957 | 0.953 |
| | | 0.77 | 0.75 | 10.82 | 53.10 | 0.70 | 5 | 0.939 | 0.948 |
| | | **0.89** | **0.03** | **62.75** | **48.64** | **0.02** | **6** | **0.318** | **0.313** |
| | | 0.70 | 0.00 | 94.43 | 24.09 | 0.70 | 13 | 0.948 | 0.945 |
| | | **0.14** | **0.10** | **32.49** | **3.75** | **0.08** | **40** | **0.418** | **0.361** |
| | | **0.20** | **0.69** | **85.00** | **98.95** | **0.13** | **21** | **0.625** | **0.359** |
| | | **0.38** | **0.08** | **88.24** | **87.87** | **0.02** | **35** | **0.323** | **0.325** |
| | | 0.88 | 0.35 | 0.61 | 81.98 | 0.48 | 62 | 0.798 | 1.010 |
| | | 0.75 | 0.26 | 77.06 | 82.04 | 0.71 | 58 | 0.952 | 0.924 |
| | | 0.51 | 0.55 | 43.20 | 49.61 | 0.41 | 22 | 0.900 | 0.935 |
| | | 0.26 | 0.18 | 71.54 | 91.21 | 0.25 | 47 | 0.814 | 0.768 |
| | | 0.70 | 0.14 | 92.08 | 9.91 | 0.97 | 48 | 0.980 | 1.012 |
| | CWFID2 | 0.82 | 0.65 | 55.21 | 80.36 | 0.71 | 37 | 0.972 | 1.031 |
| | | 0.40 | 0.31 | 27.43 | 13.50 | 0.20 | 32 | 0.470 | 0.463 |
| | | 0.60 | 0.19 | 53.28 | 40.90 | 0.21 | 53 | 0.617 | 0.505 |
| | | 0.38 | 0.19 | 74.63 | 24.43 | 0.82 | 18 | 0.991 | 1.074 |
| | | **0.15** | **0.66** | **13.81** | **15.64** | **0.43** | **38** | **0.883** | **0.748** |
| | | 0.06 | 0.01 | 59.98 | 53.01 | 0.01 | 75 | 0.346 | 0.364 |
| | | **0.08** | **0.82** | **25.44** | **97.80** | **0.69** | **64** | **0.969** | **1.026** |
| | | 0.22 | 0.79 | 85.52 | 25.42 | 0.51 | 28 | 0.914 | 1.007 |
| | | 0.25 | 0.31 | 81.36 | 93.05 | 0.54 | 17 | 0.940 | 0.893 |
| | | **0.64** | **0.88** | **94.69** | **48.41** | **0.77** | **63** | **0.985** | **1.025** |
| | | **0.24** | **0.16** | **81.89** | **25.18** | **0.43** | **17** | **0.922** | **0.886** |
| | | **0.58** | **0.38** | **87.93** | **17.94** | **0.75** | **26** | **0.987** | **1.038** |
| | | 0.96 | 0.27 | 7.12 | 44.25 | 0.45 | 41 | 0.914 | 0.977 |
| | | 0.46 | 0.98 | 81.37 | 70.52 | 0.26 | 82 | 0.676 | 0.551 |
| | | 0.93 | 0.41 | 7.22 | 61.46 | 0.11 | 60 | 0.407 | 0.478 |
| | | 0.92 | 0.16 | 97.66 | 54.72 | 0.22 | 83 | 0.708 | 0.616 |
| | | 0.16 | 0.23 | 21.32 | 34.90 | 0.78 | 44 | 0.973 | 1.066 |
| | | 0.84 | 0.48 | 64.21 | 57.63 | 0.63 | 70 | 0.942 | 1.002 |
| | | 0.19 | 0.42 | 77.18 | 41.75 | 0.50 | 19 | 0.929 | 1.024 |
| | | 0.75 | 0.23 | 61.41 | 34.05 | 0.92 | 89 | 0.996 | 1.061 |
| | Broccoli | 0.64 | 0.85 | 65.78 | 20.79 | 0.51 | 6 | 0.048 | 0.051 |
| | | 0.30 | 0.65 | 10.21 | 41.01 | 0.12 | 81 | 0.052 | 0.040 |
| | | **0.06** | **0.52** | **28.27** | **63.74** | **0.39** | **98** | **0.082** | **0.061** |
| | | 0.17 | 0.87 | 51.73 | 83.96 | 0.76 | 51 | 0.109 | 0.092 |
| | | **0.09** | **0.41** | **13.82** | **36.76** | **0.45** | **42** | **0.090** | **0.064** |
| | | 0.65 | 0.28 | 15.15 | 78.35 | 0.59 | 13 | 0.123 | 0.067 |
| | | **0.39** | **0.38** | **76.35** | **4.41** | **0.64** | **34** | **0.089** | **0.049** |
| | | **0.47** | **0.46** | **47.69** | **71.15** | **0.58** | **43** | **0.096** | **0.057** |
| | | 0.36 | 0.29 | 35.53 | 21.28 | 0.26 | 41 | 0.072 | 0.102 |
| | | 0.83 | 0.23 | 75.52 | 70.77 | 0.26 | 27 | 0.082 | 0.067 |
| | | 0.29 | 0.19 | 28.85 | 23.20 | 0.84 | 78 | 0.139 | 0.103 |
| | | 0.03 | 0.67 | 45.33 | 40.84 | 0.90 | 51 | 0.128 | 0.075 |
| | | 0.46 | 0.33 | 96.37 | 84.06 | 0.17 | 17 | 0.068 | 0.051 |
| | | 0.56 | 0.35 | 68.22 | 30.21 | 0.12 | 69 | 0.063 | 0.060 |
| | | 0.69 | 0.33 | 39.24 | 25.83 | 0.68 | 38 | 0.122 | 0.133 |
| | | 0.42 | 0.74 | 78.00 | 11.09 | 0.52 | 92 | 0.110 | 0.090 |
| | | 0.86 | 0.07 | 26.32 | 72.59 | 0.21 | 96 | 0.094 | 0.105 |
| | | 0.60 | 0.13 | 49.31 | 88.81 | 0.07 | 27 | 0.065 | 0.043 |
| | | **0.53** | **0.80** | **23.47** | **39.24** | **0.70** | **22** | **0.111** | **0.055** |
| | | 0.69 | 0.54 | 14.99 | 34.89 | 0.04 | 27 | 0.051 | 0.031 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0.81** | **0.01** | **4.19** | **48.69** | **0.15** | **68** | **0.062** | **0.042** |
| | | **0.62** | **0.44** | **82.75** | **70.39** | **0.52** | **18** | **0.061** | **0.052** |
| | | 0.74 | 0.55 | 38.96 | 53.23 | 0.66 | 2 | 0.071 | 0.066 |
| | | 0.13 | 0.98 | 56.70 | 27.60 | 0.76 | 39 | 0.150 | 0.141 |
| | | 0.07 | 0.21 | 73.11 | 67.63 | 0.24 | 50 | 0.119 | 0.103 |
| | | 0.49 | 0.88 | 85.60 | 48.79 | 0.31 | 89 | 0.122 | 0.104 |
| | | 0.70 | 0.08 | 93.34 | 17.03 | 0.61 | 18 | 0.120 | 0.126 |
| | | 0.08 | 0.16 | 84.08 | 54.84 | 0.35 | 72 | 0.129 | 0.139 |
| | | 0.35 | 0.17 | 92.73 | 52.92 | 0.33 | 19 | 0.075 | 0.074 |
| UNet | BroccoliD | **0.24** | **0.92** | **7.03** | **91.88** | **0.93** | **64** | **0.148** | **0.132** |
| | | 0.64 | 0.62 | 4.23 | 69.72 | 0.61 | 56 | 0.154 | 0.139 |
| | | 0.73 | 0.89 | 93.24 | 50.54 | 0.56 | 94 | 0.136 | 0.162 |
| | | 0.95 | 0.51 | 52.20 | 43.15 | 0.75 | 59 | 0.151 | 0.148 |
| | | 0.73 | 0.86 | 45.14 | 98.61 | 0.61 | 93 | 0.143 | 0.152 |
| | | **0.87** | **0.57** | **12.80** | **95.70** | **0.02** | **86** | **0.064** | **0.045** |
| | | 0.91 | 0.71 | 32.22 | 4.03 | 0.78 | 78 | 0.133 | 0.208 |
| | | 0.06 | 0.15 | 50.67 | 30.70 | 0.39 | 10 | 0.083 | 0.085 |
| | | 0.69 | 0.83 | 96.81 | 46.33 | 0.53 | 43 | 0.137 | 0.160 |
| | | 0.10 | 0.37 | 12.97 | 43.12 | 0.60 | 46 | 0.129 | 0.131 |
| | | 0.36 | 0.51 | 97.23 | 9.55 | 0.55 | 90 | 0.148 | 0.144 |
| | | 0.37 | 0.24 | 56.57 | 89.86 | 0.85 | 22 | 0.962 | 0.981 |
| | | 0.28 | 0.62 | 79.69 | 21.44 | 0.54 | 9 | 0.947 | 0.935 |
| | | 0.33 | 0.07 | 81.30 | 57.90 | 0.92 | 12 | 0.967 | 0.961 |
| | | 0.39 | 0.77 | 67.09 | 87.02 | 0.37 | 88 | 0.895 | 0.826 |
| | | 0.80 | 0.69 | 43.41 | 79.56 | 0.71 | 36 | 0.940 | 0.963 |
| | | 0.13 | 0.90 | 41.16 | 17.93 | 0.56 | 50 | 0.939 | 0.909 |
| | | **0.20** | **0.59** | **41.47** | **21.81** | **0.28** | **46** | **0.764** | **0.496** |
| | | 0.61 | 0.18 | 74.88 | 12.85 | 0.92 | 97 | 0.970 | 1.003 |
| | | 0.47 | 0.15 | 37.77 | 93.54 | 0.83 | 34 | 0.965 | 0.960 |
| | CWFID1 | 0.55 | 0.02 | 28.03 | 14.94 | 0.45 | 71 | 0.899 | 0.879 |
| | | 0.92 | 0.68 | 34.32 | 51.53 | 0.75 | 79 | 0.933 | 0.997 |
| | | 0.94 | 0.32 | 52.75 | 48.55 | 0.30 | 33 | 0.821 | 0.708 |
| | | **0.57** | **0.24** | **43.05** | **71.38** | **0.16** | **4** | **0.648** | **0.435** |
| | | 0.75 | 0.34 | 90.48 | 50.18 | 0.50 | 14 | 0.917 | 0.925 |
| | | 0.21 | 0.86 | 71.78 | 64.77 | 0.42 | 86 | 0.925 | 0.821 |
| | | **0.75** | **0.57** | **0.48** | **3.55** | **0.05** | **52** | **0.395** | **0.352** |
| | | **0.54** | **0.22** | **17.59** | **30.56** | **0.11** | **60** | **0.431** | **0.370** |
| | | 0.79 | 0.43 | 94.76 | 79.91 | 0.33 | 68 | 0.873 | 0.800 |
| | | 0.75 | 0.40 | 8.17 | 15.40 | 0.55 | 86 | 0.930 | 0.935 |
| | | 0.19 | 0.60 | 16.05 | 79.20 | 0.43 | 26 | 0.917 | 0.943 |
| | | **0.38** | **0.92** | **74.71** | **9.25** | **0.31** | **17** | **0.870** | **0.868** |
| | | 0.57 | 0.23 | 30.09 | 45.56 | 0.63 | 77 | 0.966 | 1.049 |
| U-ReNet | | 0.53 | 0.33 | 60.24 | 21.81 | 0.89 | 36 | 0.996 | 1.063 |
| | | 0.56 | 0.93 | 11.23 | 17.39 | 0.57 | 19 | 0.972 | 1.024 |
| | | 0.95 | 0.79 | 43.77 | 24.58 | 0.26 | 98 | 0.874 | 0.801 |
| | | 0.86 | 0.63 | 79.68 | 99.19 | 0.59 | 88 | 0.955 | 1.040 |
| | | 0.04 | 0.17 | 7.18 | 10.01 | 0.80 | 68 | 0.969 | 1.012 |
| | | 0.41 | 0.23 | 72.27 | 14.49 | 0.39 | 74 | 0.943 | 0.921 |
| | | 0.75 | 0.92 | 41.71 | 83.91 | 0.04 | 54 | 0.572 | 0.470 |
| | CWFID2 | 0.56 | 0.14 | 87.84 | 12.28 | 0.58 | 74 | 0.978 | 1.008 |
| | | 0.91 | 0.43 | 2.73 | 98.88 | 0.21 | 72 | 0.920 | 0.640 |
| | | 0.52 | 0.84 | 91.75 | 42.38 | 0.51 | 74 | 0.985 | 1.006 |
| | | 0.08 | 0.85 | 4.27 | 0.85 | 0.51 | 39 | 0.964 | 1.022 |
| | | **0.71** | **0.52** | **46.23** | **1.74** | **0.53** | **26** | **0.964** | **0.974** |
| | | **0.30** | **0.20** | **40.59** | **15.64** | **0.95** | **4** | **0.977** | **1.064** |
| | | **0.26** | **0.79** | **3.00** | **31.33** | **0.93** | **98** | **0.979** | **1.048** |
| | | 0.11 | 0.93 | 38.42 | 19.75 | 0.37 | 53 | 0.936 | 0.975 |
| | | 0.69 | 0.27 | 60.23 | 76.11 | 0.29 | 79 | 0.928 | 0.898 |
| | | 0.18 | 0.30 | 44.34 | 36.60 | 0.39 | 33 | 0.967 | 1.000 |
| | | **0.92** | **0.91** | **20.08** | **9.84** | **0.83** | **29** | **0.984** | **1.400** |
| | | 0.80 | 0.76 | 46.74 | 26.34 | 0.19 | 55 | 0.053 | 0.065 |
| | | 0.10 | 0.35 | 53.30 | 33.30 | 0.37 | 18 | 0.080 | 0.081 |
| | | 0.73 | 0.33 | 76.35 | 28.54 | 0.06 | 90 | 0.067 | 0.086 |
| | Broccoli | 0.05 | 0.58 | 47.57 | 45.47 | 0.40 | 91 | 0.100 | 0.091 |
| | | **0.50** | **0.46** | **44.62** | **13.10** | **0.56** | **90** | **0.143** | **0.120** |
| | | **0.56** | **0.21** | **42.96** | **47.69** | **0.04** | **89** | **0.061** | **0.042** |
| | | 0.25 | 0.09 | 12.64 | 3.67 | 0.77 | 5 | 0.134 | 0.163 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0.46** | **0.58** | **87.34** | **41.56** | **0.89** | **32** | **0.150** | **0.129** |
| | | 0.02 | 0.77 | 53.63 | 68.06 | 0.87 | 20 | 0.162 | 0.151 |
| | | **0.27** | **0.48** | **42.49** | **49.08** | **0.88** | **49** | **0.149** | **0.126** |
| | | 0.33 | 0.10 | 90.35 | 76.70 | 0.04 | 56 | 0.049 | 0.044 |
| | | 0.47 | 0.31 | 2.83 | 41.03 | 0.11 | 34 | 0.068 | 0.057 |
| | | 0.76 | 0.12 | 35.00 | 63.71 | 0.28 | 21 | 0.066 | 0.081 |
| | Broccoli | 0.51 | 0.53 | 51.64 | 24.99 | 0.55 | 50 | 0.118 | 0.138 |
| | | 0.72 | 0.52 | 26.71 | 55.75 | 0.68 | 70 | 0.147 | 0.140 |
| | | 0.72 | 0.31 | 21.72 | 89.86 | 0.47 | 35 | 0.106 | 0.103 |
| | | **0.53** | **0.29** | **53.91** | **22.84** | **0.51** | **12** | **0.091** | **0.078** |
| | | 0.65 | 0.18 | 12.20 | 88.17 | 0.51 | 60 | 0.113 | 0.117 |
| | | 0.23 | 0.55 | 11.52 | 77.36 | 0.41 | 65 | 0.091 | 0.160 |
| | | **0.65** | **0.17** | **56.48** | **48.39** | **0.83** | **51** | **0.151** | **0.127** |
| U-ReNet | | 0.68 | 0.61 | 58.15 | 10.50 | 0.79 | 10 | 0.068 | 0.077 |
| | | **0.20** | **0.05** | **63.95** | **45.50** | **0.63** | **91** | **0.098** | **0.084** |
| | | 0.17 | 0.22 | 41.77 | 73.00 | 0.96 | 85 | 0.152 | 0.172 |
| | | 0.91 | 0.39 | 53.36 | 92.88 | 0.96 | 31 | 0.149 | 0.143 |
| | | 0.77 | 0.50 | 39.27 | 51.33 | 0.49 | 24 | 0.079 | 0.089 |
| | | 0.41 | 0.76 | 65.97 | 64.62 | 0.78 | 65 | 0.156 | 0.153 |
| | | 0.63 | 0.35 | 35.62 | 81.66 | 0.86 | 25 | 0.156 | 0.139 |
| | BroccoliD | 0.68 | 0.88 | 77.15 | 17.97 | 0.44 | 73 | 0.088 | 0.108 |
| | | 0.06 | 0.50 | 84.66 | 2.17 | 0.38 | 81 | 0.123 | 0.132 |
| | | **0.65** | **0.01** | **50.00** | **47.44** | **0.40** | **38** | **0.094** | **0.073** |
| | | 0.63 | 0.20 | 66.88 | 31.61 | 0.44 | 91 | 0.097 | 0.090 |
| | | **0.25** | **0.60** | **72.59** | **3.86** | **0.57** | **87** | **0.112** | **0.089** |
| | | 0.99 | 0.95 | 38.27 | 86.44 | 0.62 | 93 | 0.146 | 0.133 |
| | | **0.83** | **0.38** | **34.64** | **33.82** | **0.44** | **73** | **0.099** | **0.081** |
| | | 0.83 | 0.79 | 50.85 | 55.84 | 0.16 | 35 | 0.071 | 0.077 |
| | | 0.29 | 0.38 | 10.65 | 99.77 | 0.33 | 21 | 0.119 | 0.100 |
| | | 0.50 | 0.69 | 77.03 | 62.85 | 0.65 | 90 | 0.969 | 0.980 |
| | | **0.24** | **0.98** | **84.49** | **8.52** | **0.44** | **81** | **0.912** | **0.866** |
| | | **0.03** | **0.91** | **4.58** | **82.85** | **0.46** | **17** | **0.911** | **0.842** |
| | | 0.51 | 0.54 | 66.78 | 0.43 | 0.99 | 49 | 0.990 | 1.007 |
| | | 0.66 | 0.06 | 42.48 | 48.84 | 0.55 | 46 | 0.952 | 0.923 |
| | | **0.67** | **0.01** | **97.42** | **12.72** | **0.08** | **21** | **0.524** | **0.387** |
| | | 0.00 | 0.35 | 7.09 | 80.23 | 0.38 | 26 | 0.859 | 0.809 |
| | | **0.82** | **0.18** | **67.78** | **89.82** | **0.36** | **25** | **0.854** | **0.615** |
| | | 0.04 | 0.53 | 38.01 | 51.88 | 0.72 | 11 | 0.953 | 0.932 |
| | CWFID1 | 0.69 | 0.66 | 40.65 | 98.03 | 0.53 | 32 | 0.929 | 0.945 |
| | | 0.32 | 0.18 | 54.61 | 24.82 | 0.91 | 37 | 0.991 | 0.989 |
| | | **0.30** | **0.25** | **41.07** | **40.23** | **0.40** | **88** | **0.877** | **0.751** |
| | | 0.12 | 0.30 | 61.88 | 13.64 | 0.56 | 25 | 0.940 | 0.857 |
| | | 0.28 | 0.40 | 84.79 | 95.09 | 0.97 | 84 | 0.963 | 0.971 |
| | | 0.70 | 0.24 | 1.78 | 65.94 | 0.96 | 69 | 0.963 | 1.033 |
| | | 0.56 | 0.79 | 65.78 | 24.85 | 0.39 | 70 | 0.937 | 0.819 |
| | | 0.92 | 0.63 | 53.62 | 42.86 | 0.46 | 90 | 0.923 | 0.882 |
| | | **0.38** | **0.38** | **29.59** | **75.47** | **0.02** | **84** | **0.350** | **0.345** |
| U-ReNet2 | | 0.35 | 0.76 | 75.34 | 9.37 | 0.68 | 62 | 0.956 | 0.963 |
| | | 0.65 | 0.13 | 21.07 | 86.99 | 0.89 | 0 | 0.960 | 0.973 |
| | | 0.31 | 0.75 | 77.39 | 37.35 | 0.05 | 43 | 0.802 | 0.811 |
| | | 0.74 | 0.18 | 55.11 | 62.49 | 0.74 | 17 | 1.389 | 1.511 |
| | | 0.78 | 0.36 | 54.48 | 24.23 | 0.37 | 86 | 1.340 | 1.334 |
| | | **0.01** | **0.04** | **3.32** | **71.61** | **0.70** | **78** | **1.404** | **1.438** |
| | | 0.31 | 0.74 | 58.06 | 12.54 | 0.93 | 68 | 1.430 | 1.485 |
| | | 0.61 | 0.83 | 14.96 | 50.77 | 0.69 | 18 | 1.399 | 1.429 |
| | | 0.45 | 0.73 | 78.65 | 88.81 | 0.71 | 90 | 1.388 | 1.391 |
| | | 0.50 | 0.05 | 58.47 | 6.13 | 0.16 | 31 | 1.060 | 1.022 |
| | CWFID2 | 0.90 | 0.57 | 92.63 | 74.55 | 0.17 | 58 | 1.167 | 1.028 |
| | | 0.71 | 0.00 | 53.16 | 53.07 | 0.27 | 0 | 1.313 | 1.133 |
| | | 0.77 | 0.20 | 50.87 | 52.32 | 0.92 | 1 | 1.489 | 1.565 |
| | | **0.13** | **0.32** | **97.66** | **14.25** | **0.90** | **67** | **1.470** | **1.489** |
| | | 0.76 | 0.70 | 85.89 | 29.15 | 0.03 | 1 | 0.855 | 0.878 |
| | | 0.70 | 0.04 | 99.71 | 62.52 | 0.04 | 32 | 0.856 | 0.891 |
| | | 0.46 | 0.74 | 73.14 | 37.80 | 0.25 | 7 | 1.211 | 1.150 |
| | | **0.46** | **0.06** | **12.77** | **47.56** | **0.96** | **13** | **1.446** | **1.510** |
| | | 0.51 | 0.97 | 44.80 | 41.69 | 0.91 | 36 | 1.436 | 1.476 |
| | | 0.57 | 0.05 | 50.20 | 85.36 | 0.66 | 38 | 1.398 | 1.418 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| U-ReNet2 | CWFID2 | 0.75 | 0.92 | 97.61 | 35.07 | 0.26 | 30 | 1.236 | 1.030 |
| | | **0.63** | **0.39** | **48.69** | **18.15** | **0.93** | **89** | **1.428** | **1.464** |
| | Broccoli | 0.73 | 0.35 | 58.02 | 63.38 | 0.68 | 39 | 0.070 | 0.100 |
| | | 0.79 | 0.22 | 16.07 | 17.61 | 0.24 | 60 | 0.077 | 0.079 |
| | | **0.32** | **0.48** | **74.47** | **37.81** | **0.13** | **70** | **0.066** | **0.053** |
| | | **0.37** | **0.01** | **57.26** | **95.67** | **0.01** | **63** | **0.041** | **0.040** |
| | | 0.34 | 0.86 | 22.75 | 47.64 | 0.78 | 88 | 0.150 | 0.127 |
| | | 0.28 | 0.06 | 37.96 | 99.35 | 0.70 | 3 | 0.094 | 0.099 |
| | | 0.10 | 0.78 | 22.97 | 63.88 | 0.25 | 47 | 0.070 | 0.084 |
| | | 0.23 | 0.29 | 80.30 | 62.34 | 0.47 | 18 | 0.076 | 0.088 |
| | | 0.20 | 0.17 | 48.18 | 13.93 | 0.68 | 73 | 0.119 | 0.118 |
| | | 0.23 | 0.16 | 82.43 | 89.11 | 0.84 | 11 | 0.148 | 0.136 |
| | | 0.47 | 0.59 | 88.61 | 87.36 | 0.17 | 56 | 0.064 | 0.061 |
| | | 0.83 | 0.46 | 93.68 | 72.91 | 0.49 | 30 | 0.087 | 0.089 |
| | | **0.51** | **0.06** | **85.90** | **66.43** | **0.22** | **93** | **0.084** | **0.072** |
| | | 0.67 | 0.27 | 25.87 | 95.07 | 0.01 | 63 | 0.040 | 0.053 |
| | | 0.62 | 0.63 | 49.06 | 29.85 | 0.54 | 68 | 0.097 | 0.103 |
| | | **0.70** | **0.07** | **29.10** | **16.86** | **0.18** | **41** | **0.061** | **0.059** |
| | | 0.59 | 0.80 | 55.85 | 85.32 | 0.85 | 32 | 0.152 | 0.253 |
| | | 0.45 | 0.84 | 34.88 | 30.13 | 0.52 | 76 | 0.111 | 0.110 |
| | | 0.69 | 0.49 | 0.88 | 34.02 | 0.82 | 95 | 0.165 | 0.156 |
| | | 0.75 | 0.78 | 37.37 | 24.24 | 0.51 | 89 | 0.086 | 0.086 |
| | BroccoliD | 0.23 | 0.74 | 88.93 | 61.38 | 0.49 | 41 | 0.055 | 0.061 |
| | | 0.88 | 0.99 | 15.71 | 38.66 | 0.72 | 79 | 0.149 | 0.152 |
| | | **0.13** | **0.05** | **16.32** | **6.29** | **0.45** | **28** | **0.084** | **0.076** |
| | | 0.05 | 0.50 | 29.17 | 93.90 | 0.98 | 72 | 0.163 | 0.156 |
| | | 0.76 | 0.86 | 56.36 | 53.25 | 0.60 | 72 | 0.136 | 0.137 |
| | | 0.16 | 0.84 | 20.36 | 22.25 | 0.56 | 83 | 0.133 | 0.111 |
| | | **0.19** | **0.40** | **17.91** | **99.25** | **0.29** | **76** | **0.072** | **0.067** |
| | | 0.75 | 0.49 | 60.29 | 49.78 | 0.90 | 91 | 0.164 | 0.164 |
| | | 0.59 | 0.28 | 41.14 | 69.27 | 0.52 | 91 | 0.135 | 0.159 |
| | | 0.09 | 0.14 | 36.61 | 42.47 | 0.15 | 26 | 0.053 | 0.056 |
| | | 0.92 | 0.18 | 74.49 | 51.53 | 0.84 | 86 | 0.171 | 0.161 |
| | | **0.68** | **0.70** | **65.81** | **83.26** | **0.74** | **1** | **0.129** | **0.083** |
| | | 0.48 | 0.60 | 80.30 | 77.27 | 0.64 | 14 | 0.118 | 0.113 |
| | | 0.73 | 0.41 | 49.84 | 24.79 | 0.80 | 54 | 0.161 | 0.164 |
| | | 0.27 | 0.37 | 77.27 | 0.80 | 0.30 | 2 | 0.069 | 0.080 |
| | | 0.78 | 0.39 | 81.90 | 9.76 | 0.74 | 8 | 0.128 | 0.121 |
| | | 0.21 | 0.94 | 89.30 | 29.41 | 0.63 | 86 | 0.136 | 0.108 |
| | | **0.14** | **0.82** | **85.33** | **77.70** | **0.01** | **90** | **0.043** | **0.043** |
| | | 0.74 | 0.18 | 0.40 | 13.97 | 0.63 | 85 | 0.185 | 0.161 |
| | | 0.51 | 0.59 | 3.45 | 83.05 | 0.51 | 61 | 0.159 | 0.159 |
| ReSeg | CWFID1 | **0.61** | **0.57** | **81.59** | **43.81** | **0.94** | **3** | **1.017** | **0.950** |
| | | 0.59 | 0.64 | 73.52 | 1.99 | 0.68 | 26 | 0.993 | 1.010 |
| | | 0.25 | 0.82 | 44.81 | 59.02 | 0.57 | 64 | 1.143 | 1.327 |
| | | 0.20 | 0.90 | 97.69 | 51.49 | 0.70 | 95 | 1.067 | 1.248 |
| | | **0.09** | **0.48** | **86.80** | **85.84** | **0.61** | **38** | **1.105** | **0.975** |
| | | 0.29 | 0.88 | 89.92 | 11.93 | 0.16 | 60 | 0.996 | 1.016 |
| | | **0.89** | **0.03** | **75.55** | **80.00** | **0.59** | **86** | **1.148** | **1.113** |
| | | 0.90 | 0.35 | 61.04 | 70.08 | 0.16 | 55 | 0.986 | 1.044 |
| | | 0.75 | 0.54 | 90.50 | 20.71 | 0.04 | 87 | **0.421** | **0.403** |
| | | 0.28 | 0.94 | 76.16 | 48.01 | 0.29 | 91 | 1.112 | 1.235 |
| | | 0.45 | 0.15 | 69.39 | 31.39 | 0.44 | 52 | 1.016 | 1.033 |
| | | **0.89** | **0.03** | **96.48** | **19.12** | **0.10** | **89** | **0.449** | **0.436** |
| | | 0.27 | 0.60 | 29.59 | 77.10 | 0.93 | 87 | 0.985 | 1.112 |
| | | 0.29 | 0.66 | 53.26 | 60.30 | 0.75 | 74 | 1.053 | 1.057 |
| | | 0.64 | 0.28 | 8.18 | 75.54 | 0.41 | 60 | 1.075 | 1.132 |
| | | 0.20 | 0.28 | 40.06 | 80.47 | 0.05 | 76 | 1.036 | 1.083 |
| | | 0.21 | 0.77 | 35.57 | 98.28 | 0.23 | 20 | 0.962 | 1.098 |
| | | 0.44 | 0.47 | 13.08 | 27.14 | 0.93 | 12 | 1.061 | 1.754 |
| | | **0.86** | **0.10** | **43.41** | **90.49** | **0.77** | **49** | **1.102** | **0.961** |
| | | **0.66** | **0.20** | **45.55** | **51.45** | **0.55** | **5** | **0.974** | **0.951** |
| | CWFID2 | 0.52 | 0.26 | 72.95 | 50.54 | 0.22 | 14 | 1.174 | 1.111 |
| | | 0.91 | 0.65 | 77.50 | 42.04 | 0.82 | 11 | 1.798 | 1.693 |
| | | 0.05 | 0.02 | 41.38 | 94.20 | 0.93 | 63 | 1.826 | 1.775 |
| | | 0.02 | 0.58 | 11.49 | 76.23 | 0.74 | 98 | 1.660 | 1.581 |
| | | **0.68** | **0.60** | **42.44** | **78.99** | **0.75** | **98** | **1.669** | **1.491** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ReSeg | CWFID2 | 0.09 | 0.51 | 64.23 | 71.37 | 0.20 | 95 | 1.331 | 1.276 |
| | | 0.20 | 0.48 | 63.56 | 15.15 | 0.04 | 84 | 1.183 | 1.110 |
| | | **0.26** | **0.19** | **49.06** | **70.41** | **0.59** | **20** | **1.637** | **1.661** |
| | | 0.72 | 0.40 | 85.85 | 16.08 | 0.10 | 58 | 1.212 | 1.174 |
| | | 0.22 | 0.06 | 31.86 | 0.06 | 0.09 | 13 | 1.171 | 1.092 |
| | | 0.44 | 0.68 | 29.51 | 14.80 | 0.56 | 61 | 1.591 | 1.399 |
| | | 0.79 | 0.94 | 9.28 | 50.63 | 0.73 | 73 | 1.770 | 1.758 |
| | | 0.47 | 0.88 | 33.22 | 47.17 | 0.32 | 86 | 1.360 | 1.328 |
| | | 0.76 | 0.51 | 7.66 | 90.74 | 0.35 | 7 | 1.342 | 1.289 |
| | | 0.74 | 0.36 | 45.69 | 92.72 | 0.65 | 16 | 1.611 | 1.522 |
| | | 0.67 | 0.28 | 91.07 | 70.21 | 0.83 | 19 | 1.753 | 1.718 |
| | | 0.95 | 0.26 | 20.40 | 76.16 | 0.65 | 48 | 1.589 | 1.558 |
| | | **0.70** | **0.76** | **25.82** | **21.07** | **0.11** | **17** | **1.247** | **1.174** |
| | | 0.56 | 0.89 | 2.93 | 12.69 | 0.41 | 23 | 1.473 | 1.371 |
| | | **0.10** | **0.56** | **12.97** | **74.16** | **0.75** | **81** | **1.734** | **1.728** |
| | Broccoli | 0.60 | 0.45 | 23.96 | 65.82 | 0.55 | 96 | 0.122 | 0.121 |
| | | 0.61 | 0.53 | 95.57 | 0.30 | 0.59 | 83 | 0.121 | 0.129 |
| | | **0.31** | **0.82** | **69.89** | **87.74** | **0.08** | **11** | **0.065** | **0.048** |
| | | **0.69** | **0.73** | **53.53** | **88.86** | **0.01** | **80** | **0.058** | **0.040** |
| | | **0.69** | **0.74** | **30.67** | **33.29** | **0.09** | **19** | **0.063** | **0.054** |
| | | **0.14** | **0.48** | **49.09** | **48.53** | **0.31** | **54** | **0.082** | **0.077** |
| | | **0.05** | **0.42** | **66.36** | **20.00** | **0.11** | **68** | **0.069** | **0.054** |
| | | **0.61** | **0.02** | **1.24** | **39.77** | **0.07** | **87** | **0.117** | **0.068** |
| | | 0.41 | 0.33 | 4.68 | 34.86 | 0.88 | 98 | 0.169 | 0.148 |
| | | 0.83 | 0.34 | 19.04 | 81.62 | 0.76 | 29 | 0.173 | 0.180 |
| | | 0.90 | 0.77 | 69.79 | 82.91 | 0.76 | 72 | 0.150 | 0.134 |
| | | 0.18 | 0.52 | 16.41 | 70.13 | 0.46 | 35 | 0.116 | 0.096 |
| | | **0.54** | **0.69** | **20.75** | **43.85** | **0.14** | **77** | **0.078** | **0.067** |
| | | **0.40** | **0.34** | **30.77** | **63.63** | **0.06** | **33** | **0.066** | **0.052** |
| | | 0.17 | 0.24 | 62.35 | 89.40 | 0.59 | 16 | 0.149 | 0.118 |
| | | **0.61** | **0.33** | **66.74** | **31.01** | **0.08** | **69** | **0.071** | **0.052** |
| | | 0.57 | 0.33 | 25.19 | 67.41 | 0.64 | 22 | 0.122 | 0.124 |
| | | 0.33 | 0.32 | 85.56 | 59.34 | 0.56 | 53 | 0.136 | 0.108 |
| | | 0.20 | 0.85 | 25.89 | 25.21 | 0.50 | 95 | 0.119 | 0.103 |
| | | **0.02** | **0.59** | **62.39** | **88.81** | **0.28** | **81** | **0.084** | **0.076** |
| | BroccoliD LR=1e-2 | 0.21 | 0.75 | 98.49 | 40.48 | 0.42 | 9 | 0.205 | 0.195 |
| | | 0.23 | 0.10 | 76.88 | 63.69 | 0.55 | 96 | 0.212 | 0.197 |
| | | 0.91 | 0.27 | 31.30 | 18.81 | 0.45 | 95 | 0.212 | 0.199 |
| | | 0.13 | 0.96 | 81.94 | 47.56 | 0.51 | 97 | 0.231 | 0.197 |
| | | 0.65 | 0.96 | 54.45 | 18.69 | 0.03 | 53 | 0.210 | 0.193 |
| | | 0.91 | 0.65 | 52.29 | 31.69 | 0.06 | 54 | 0.210 | 0.200 |
| | | 0.09 | 0.50 | 61.90 | 1.98 | 0.72 | 55 | 0.213 | 0.202 |
| | | 0.32 | 0.85 | 85.52 | 74.21 | 0.06 | 46 | 0.128 | 0.183 |
| | | 0.65 | 0.02 | 40.55 | 91.54 | 0.24 | 70 | 0.211 | 0.194 |
| | | 0.85 | 0.20 | 47.85 | 78.88 | 0.27 | 59 | 0.212 | 0.194 |
| | | 0.03 | 0.16 | 54.22 | 70.56 | 0.79 | 80 | 0.214 | 0.200 |
| | | 0.56 | 0.24 | 56.97 | 20.03 | 0.83 | 90 | 0.213 | 0.204 |
| | | 0.91 | 0.32 | 85.34 | 69.15 | 0.93 | 61 | 0.212 | 0.200 |
| | | 0.20 | 0.09 | 29.59 | 30.01 | 0.69 | 9 | 0.208 | 0.208 |
| | | 0.86 | 0.26 | 60.53 | 18.95 | 0.95 | 64 | 0.210 | 0.197 |
| | | 0.91 | 0.63 | 35.57 | 76.79 | 0.15 | 84 | 0.145 | 0.134 |
| | | 0.83 | 0.86 | 91.27 | 67.07 | 0.76 | 55 | 0.214 | 0.205 |
| | | 0.86 | 0.38 | 1.53 | 90.51 | 0.53 | 43 | 0.276 | 0.208 |
| | | 0.30 | 0.11 | 9.93 | 93.70 | 0.38 | 73 | 0.218 | 0.205 |
| | | 0.08 | 0.94 | 53.46 | 27.21 | 0.55 | 1 | 0.207 | 0.187 |
| | BroccoliD LR=1e-3 | 0.41 | 0.35 | 26.96 | 82.69 | 0.95 | 65 | 0.142 | 0.183 |
| | | 0.67 | 0.45 | 2.35 | 81.29 | 0.50 | 11 | 0.125 | 0.099 |
| | | **0.02** | **0.15** | **95.98** | **82.86** | **0.15** | **49** | **0.065** | **0.053** |
| | | 0.74 | 0.38 | 99.41 | 88.64 | 0.83 | 23 | 0.106 | 0.080 |
| | | **0.79** | **0.42** | **22.36** | **40.16** | **0.17** | **67** | **0.070** | **0.053** |
| | | 0.73 | 0.00 | 50.67 | 81.61 | 0.43 | 52 | 0.086 | 0.070 |
| | | **0.62** | **0.37** | **48.93** | **8.55** | **0.02** | **77** | **0.059** | **0.045** |
| | | 0.76 | 0.84 | 8.59 | 18.07 | 0.80 | 1 | 0.135 | 0.108 |
| | | 0.33 | 0.25 | 85.51 | 40.95 | 0.54 | 83 | 0.091 | 0.079 |
| | | 0.59 | 0.69 | 0.78 | 57.82 | 0.35 | 5 | 0.200 | 0.152 |
| | | 0.82 | 0.16 | 9.44 | 34.76 | 0.53 | 83 | 0.096 | 0.115 |
| | | 0.52 | 0.29 | 34.39 | 93.01 | 0.44 | 92 | 0.092 | 0.086 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ReSeg | BroccoliD LR=1e-3 | 0.67 | 0.33 | 16.70 | 86.71 | 0.33 | 65 | 0.086 | 0.140 |
| | | 0.55 | 0.86 | 66.23 | 81.30 | 0.53 | 70 | 0.097 | 0.082 |
| | | **0.35** | **0.28** | **66.60** | **23.21** | **0.04** | **74** | **0.063** | **0.056** |
| | | 0.50 | 0.82 | 40.53 | 38.24 | 0.55 | 53 | 0.098 | 0.081 |
| | | 0.70 | 0.89 | 27.54 | 38.03 | 0.97 | 13 | 0.187 | 0.157 |
| | | 0.72 | 0.91 | 22.14 | 54.49 | 0.47 | 21 | 0.086 | 0.079 |
| | | 0.79 | 0.34 | 7.29 | 24.66 | 0.38 | 20 | 0.081 | 0.076 |
| | | 0.55 | 0.77 | 89.45 | 84.20 | 0.46 | 31 | 0.087 | 0.074 |
| ReConv | CWFID1 | 0.34 | 0.96 | 44.37 | 75.21 | 0.54 | 83 | 0.847 | 0.794 |
| | | 0.74 | 0.19 | 96.00 | 33.47 | 0.32 | 69 | 0.704 | 0.475 |
| | | 0.78 | 0.71 | 9.66 | 93.70 | 0.59 | 64 | 0.878 | 0.732 |
| | | 0.52 | 0.08 | 12.50 | 19.86 | 0.93 | 84 | 0.921 | 0.961 |
| | | **0.33** | **0.29** | **86.58** | **79.17** | **0.09** | **51** | **0.466** | **0.409** |
| | | 0.33 | 0.05 | 80.11 | 33.68 | 0.93 | 74 | 0.935 | 0.895 |
| | | **0.16** | **0.22** | **39.20** | **54.01** | **0.13** | **54** | **0.494** | **0.406** |
| | | 0.99 | 0.10 | 3.29 | 65.50 | 0.90 | 32 | 0.903 | 0.919 |
| | | 0.66 | 0.71 | 19.20 | 29.85 | 0.71 | 36 | 0.921 | 0.734 |
| | | 0.50 | 0.11 | 68.38 | 49.50 | 0.56 | 16 | 0.867 | 0.668 |
| | | 0.67 | 0.68 | 53.74 | 68.84 | 0.64 | 41 | 0.897 | 0.750 |
| | | 0.30 | 0.42 | 4.83 | 70.48 | 0.42 | 28 | 0.896 | 0.778 |
| | | 0.15 | 0.30 | 26.56 | 39.52 | 0.56 | 99 | 0.908 | 0.576 |
| | | 0.14 | 0.17 | 25.63 | 8.58 | 0.79 | 35 | 0.938 | 0.936 |
| | | **0.49** | **0.63** | **89.15** | **2.71** | **0.09** | **45** | **0.471** | **0.422** |
| | | **0.51** | **0.11** | **66.80** | **9.23** | **0.08** | **66** | **0.448** | **0.443** |
| | | 0.93 | 0.50 | 99.22 | 34.40 | 0.84 | 79 | 0.939 | 0.829 |
| | | **0.63** | **0.86** | **6.77** | **36.01** | **0.22** | **46** | **0.503** | **0.429** |
| | | 0.64 | 0.05 | 10.72 | 16.37 | 0.99 | 86 | 0.932 | 0.936 |
| | | **0.86** | **0.50** | **23.18** | **51.45** | **0.00** | **4** | **0.384** | **0.428** |
| | CWFID2 | **0.73** | **0.89** | **75.46** | **88.85** | **0.08** | **45** | **0.418** | **0.437** |
| | | 0.48 | 0.30 | 43.23 | 89.37 | 0.07 | 26 | 0.415 | 0.554 |
| | | **0.66** | **0.40** | **34.67** | **22.31** | **0.15** | **59** | **0.454** | **0.474** |
| | | 0.64 | 0.93 | 53.32 | 0.07 | 0.39 | 46 | 0.930 | 0.949 |
| | | 0.11 | 0.69 | 6.29 | 32.79 | 0.81 | 32 | 0.919 | 1.070 |
| | | 0.19 | 0.00 | 85.11 | 32.41 | 0.88 | 12 | 0.951 | 0.975 |
| | | **0.85** | **0.30** | **32.38** | **61.00** | **0.14** | **80** | **0.457** | **0.477** |
| | | 0.81 | 0.53 | 10.43 | 7.13 | 0.45 | 54 | 0.934 | 0.973 |
| | | 0.77 | 0.14 | 0.44 | 5.93 | 0.48 | 93 | 0.934 | 1.021 |
| | | 0.58 | 0.71 | 17.85 | 55.09 | 0.73 | 27 | 0.924 | 1.002 |
| | | 0.27 | 0.33 | 79.93 | 58.12 | 0.20 | 67 | 0.494 | 0.545 |
| | | 0.25 | 0.86 | 23.13 | 95.34 | 0.60 | 76 | 0.938 | 0.982 |
| | | 0.95 | 0.38 | 5.41 | 66.28 | 0.98 | 8 | 0.926 | 0.966 |
| | | 0.58 | 0.52 | 64.17 | 9.90 | 0.62 | 24 | 0.942 | 0.960 |
| | | 0.36 | 0.49 | 71.47 | 71.99 | 0.20 | 74 | 0.486 | 0.543 |
| | | 0.20 | 0.38 | 63.56 | 70.36 | 0.83 | 46 | 0.928 | 0.993 |
| | | **0.38** | **0.47** | **35.32** | **27.27** | **0.24** | **31** | **0.458** | **0.453** |
| | | 0.15 | 0.34 | 3.28 | 86.45 | 0.01 | 52 | 0.472 | 0.607 |
| | | 0.21 | 0.08 | 46.55 | 45.91 | 0.19 | 66 | 0.522 | 0.641 |
| | | 0.64 | 0.15 | 82.63 | 57.97 | 0.64 | 88 | 0.949 | 1.128 |
| | Broccoli | 0.23 | 0.56 | 7.30 | 18.36 | 0.39 | 27 | 0.082 | 0.070 |
| | | **0.53** | **0.22** | **45.15** | **67.67** | **0.23** | **99** | **0.083** | **0.066** |
| | | 0.54 | 0.60 | 9.68 | 88.30 | 0.70 | 38 | 0.090 | 0.077 |
| | | 0.69 | 0.95 | 10.99 | 95.66 | 0.64 | 19 | 0.090 | 0.071 |
| | | 0.84 | 0.03 | 89.71 | 83.81 | 0.87 | 85 | 0.085 | 0.076 |
| | | 0.08 | 0.63 | 68.06 | 55.45 | 0.78 | 68 | 0.085 | 0.069 |
| | | 0.04 | 0.07 | 42.65 | 64.11 | 0.92 | 22 | 0.084 | 0.076 |
| | | 0.59 | 0.81 | 57.25 | 44.16 | 0.83 | 50 | 0.082 | 0.069 |
| | | **0.73** | **0.52** | **40.73** | **59.96** | **0.30** | **65** | **0.084** | **0.072** |
| | | **0.85** | **0.91** | **85.47** | **7.61** | **0.87** | **59** | **0.085** | **0.068** |
| | | **0.15** | **0.10** | **48.56** | **37.93** | **0.06** | **88** | **0.081** | **0.066** |
| | | 0.46 | 0.14 | 84.25 | 6.41 | 0.69 | 67 | 0.084 | 0.074 |
| | | **0.78** | **0.67** | **86.70** | **18.95** | **0.75** | **57** | **0.083** | **0.069** |
| | | 0.73 | 0.32 | 1.07 | 20.35 | 0.49 | 81 | 0.118 | 0.085 |
| | | 0.02 | 0.19 | 93.27 | 13.81 | 0.70 | 46 | 0.081 | 0.068 |
| | | 0.23 | 0.80 | 48.99 | 95.91 | 0.72 | 55 | 0.085 | 0.070 |
| | | 0.75 | 0.39 | 99.05 | 50.80 | 0.69 | 0 | 0.076 | 0.069 |
| | | **0.15** | **0.50** | **47.40** | **37.79** | **0.68** | **26** | **0.083** | **0.071** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ReConv | Broccoli | **0.59** | **0.92** | **96.31** | **50.56** | **0.58** | **94** | **0.085** | **0.067** |
| | | 0.00 | 0.60 | 57.42 | 6.58 | 0.74 | 2 | 0.076 | 0.068 |
| | BroccoliD | **0.27** | **0.87** | **6.24** | **6.24** | **0.17** | **31** | **0.081** | **0.064** |
| | | 0.38 | 0.12 | 0.06 | 24.28 | 0.42 | 40 | 0.245 | 0.195 |
| | | 0.81 | 0.55 | 90.63 | 39.11 | 0.48 | 93 | 0.094 | 0.072 |
| | | 0.04 | 0.73 | 51.51 | 65.13 | 0.45 | 92 | 0.092 | 0.071 |
| | | **0.28** | **0.24** | **49.81** | **21.40** | **0.37** | **34** | **0.085** | **0.069** |
| | | **0.51** | **0.82** | **92.24** | **71.37** | **0.61** | **87** | **0.094** | **0.070** |
| | | 0.81 | 0.52 | 13.19 | 47.70 | 0.87 | 38 | 0.179 | 0.162 |
| | | 0.77 | 0.76 | 15.47 | 30.68 | 0.09 | 10 | 0.079 | 0.063 |
| | | **0.22** | **0.09** | **36.96** | **20.36** | **0.30** | **73** | **0.085** | **0.067** |
| | | **0.66** | **0.81** | **52.73** | **56.47** | **0.28** | **44** | **0.083** | **0.066** |
| | | 0.92 | 0.13 | 91.48 | 22.22 | 0.46 | 96 | 0.099 | 0.074 |
| | | **0.35** | **0.21** | **72.56** | **70.38** | **0.41** | **53** | **0.090** | **0.067** |
| | | **0.34** | **0.17** | **85.74** | **95.01** | **0.18** | **34** | **0.084** | **0.070** |
| | | **0.17** | **0.80** | **49.85** | **56.93** | **0.43** | **68** | **0.089** | **0.069** |
| | | 0.23 | 0.73 | 0.23 | 2.89 | 0.78 | 61 | 0.148 | 0.101 |
| | | 0.39 | 0.93 | 8.15 | 61.35 | 0.44 | 70 | 0.095 | 0.075 |
| | | 0.78 | 0.48 | 72.87 | 92.97 | 0.95 | 24 | 0.128 | 0.087 |
| | | 0.58 | 0.61 | 25.69 | 96.18 | 0.94 | 87 | 0.148 | 0.091 |
| | | 0.25 | 0.98 | 93.87 | 90.01 | 0.88 | 56 | 0.115 | 0.085 |
| | | **0.74** | **0.22** | **17.38** | **95.72** | **0.26** | **60** | **0.087** | **0.067** |

## A.4 Finer hyperparameter tuning

Using the results of the previous section, we have selected the configurations yielding to good results in terms of validation accuracies and used them to select finer ranges for the hyperparameters. We then performed 10 epochs of training randomly selecting the hyperparameters in this finer ranges.

| Dataset | GS | OD | D1D | D2D | EL_S | EL_A | AR | Train loss | Val loss |
|---|---|---|---|---|---|---|---|---|---|
| CWFID1 | **0.06** | **0.76** | **0.18** | **0.37** | **10.99** | **53.80** | **0.48** | **0.796** | **0.957** |
| | 0.25 | 0.88 | 0.26 | 0.35 | 28.22 | 71.51 | 0.49 | 0.845 | 1.089 |
| | 0.20 | 0.72 | 0.21 | 0.28 | 32.88 | 55.57 | 0.37 | 0.800 | 0.978 |
| | 0.23 | 0.95 | 0.34 | 0.35 | 18.44 | 54.87 | 0.52 | 0.845 | 0.967 |
| | 0.05 | 0.77 | 0.23 | 0.44 | 5.82 | 62.36 | 0.66 | 0.848 | 0.961 |
| | 0.30 | 0.59 | 0.20 | 0.34 | 37.51 | 72.88 | 0.30 | 0.809 | 19.603 |
| | 0.24 | 0.57 | 0.26 | 0.31 | 39.23 | 66.04 | 0.43 | 0.804 | 1.620 |
| | 0.25 | 0.94 | 0.16 | 0.32 | 37.55 | 84.99 | 0.28 | 0.845 | 9.909 |
| | 0.13 | 0.71 | 0.25 | 0.42 | 29.61 | 55.98 | 0.30 | 0.799 | 1.409 |
| | 0.33 | 0.64 | 0.26 | 0.31 | 11.69 | 73.03 | 0.19 | 0.824 | 6.990 |
| CWFID2 | 0.19 | 0.70 | 0.43 | 0.65 | 46.24 | 42.90 | 0.86 | 1.064 | 1.284 |
| | 0.15 | 0.33 | 0.70 | 0.62 | 91.07 | 17.66 | 0.52 | 1.020 | 1.164 |
| | 0.11 | 0.74 | 0.56 | 0.28 | 59.05 | 17.65 | 0.42 | 0.980 | 1.030 |
| | 0.18 | 0.88 | 0.67 | 0.17 | 70.15 | 5.59 | 0.48 | 1.084 | 1.228 |
| | 0.13 | 0.52 | 0.40 | 0.68 | 18.83 | 16.84 | 0.66 | 1.010 | 1.087 |
| | **0.12** | **0.42** | **0.44** | **0.47** | **46.82** | **87.44** | **0.74** | **0.975** | **1.017** |
| | 0.22 | 0.54 | 0.40 | 0.57 | 47.81 | 41.77 | 0.84 | 1.005 | 1.382 |
| | 0.21 | 0.52 | 0.80 | 0.47 | 6.52 | 38.59 | 0.80 | 1.047 | 1.121 |
| | 0.23 | 0.77 | 0.82 | 0.20 | 21.99 | 95.11 | 0.39 | 1.066 | 1.224 |
| | 0.18 | 0.26 | 0.32 | 0.49 | 7.23 | 23.47 | 0.81 | 0.967 | 1.313 |
| Broccoli | 0.07 | 0.61 | 0.74 | 0.24 | 23.21 | 104.29 | 0.04 | 0.567 | 0.587 |
| | 0.10 | 0.45 | 0.56 | 0.16 | 19.49 | 107.89 | 0.00 | 0.426 | 5.435 |
| | **0.03** | **0.57** | **0.74** | **0.19** | **42.21** | **119.33** | **0.12** | **0.614** | **0.572** |
| | 0.20 | 0.50 | 0.74 | 0.17 | 37.46 | 115.10 | 0.12 | 0.649 | 5.509 |
| | 0.10 | 0.56 | 0.71 | 0.14 | 21.73 | 115.07 | 0.04 | 0.516 | 1.733 |
| | 0.19 | 0.54 | 0.71 | 0.08 | 16.79 | 101.72 | 0.13 | 0.636 | 8.254 |
| | 0.06 | 0.59 | 0.62 | 0.13 | 19.84 | 80.15 | 0.01 | 0.413 | 1.253 |
| | 0.20 | 0.61 | 0.59 | 0.09 | 25.13 | 80.85 | 0.02 | 0.488 | 3.677 |
| | 0.10 | 0.53 | 0.56 | 0.09 | 21.38 | 103.08 | 0.03 | 0.453 | 10.302 |
| | 0.17 | 0.48 | 0.64 | 0.12 | 20.43 | 112.03 | 0.10 | 0.583 | 2.539 |

All the other networks are on the following table:

| Network | Dataset | GS | OD | EL_S | EL_A | AR | SR | Train loss | Val loss |
|---|---|---|---|---|---|---|---|---|---|
| UNet | CWFID1 | 0.16 | 0.41 | 73.07 | 52.95 | 0.03 | 28 | 0.289 | 0.317 |
| | | 0.47 | 0.45 | 67.83 | 77.84 | 0.11 | 23 | 0.315 | 0.324 |
| | | 0.49 | 0.63 | 25.35 | 17.32 | 0.07 | 14 | 0.315 | 0.325 |
| | | 0.19 | 0.11 | 82.19 | 87.77 | 0.02 | 47 | 0.270 | 0.325 |
| | | 0.44 | 0.06 | 66.25 | 32.99 | 0.10 | 7 | 0.316 | 0.322 |
| | | **0.54** | **0.45** | **67.42** | **46.75** | **0.08** | **0** | **0.317** | **0.313** |
| | | 0.47 | 0.02 | 78.82 | 94.18 | 0.10 | 46 | 0.319 | 0.324 |
| | | 0.20 | 0.50 | 65.45 | 46.53 | 0.11 | 29 | 0.327 | 0.326 |
| | | 0.46 | 0.27 | 26.15 | 24.40 | 0.14 | 43 | 0.369 | 0.338 |
| | | 0.16 | 0.24 | 20.10 | 48.03 | 0.11 | 8 | 0.321 | 0.329 |
| | CWFID2 | **0.64** | **0.74** | **13.08** | **84.31** | **0.55** | **16** | **0.410** | **0.438** |
| | | 0.24 | 0.87 | 88.39 | 22.73 | 0.74 | 61 | 0.441 | 0.597 |
| | | 0.37 | 0.35 | 12.76 | 64.92 | 0.41 | 37 | 0.361 | 0.451 |
| | | 0.42 | 0.33 | 15.22 | 72.45 | 0.74 | 24 | 0.549 | 0.584 |
| | | 0.13 | 0.54 | 20.51 | 53.37 | 0.78 | 24 | 0.622 | 0.576 |
| | | 0.55 | 0.23 | 84.64 | 74.43 | 0.74 | 59 | 0.748 | 0.780 |
| | | 0.41 | 0.64 | 89.80 | 7.23 | 0.73 | 44 | 0.636 | 0.601 |
| | | 0.38 | 0.49 | 12.45 | 79.42 | 0.80 | 48 | 0.761 | 0.834 |
| | | 0.62 | 0.80 | 17.74 | 62.90 | 0.55 | 16 | 0.438 | 0.462 |
| | | 0.16 | 0.47 | 94.42 | 92.41 | 0.42 | 64 | 0.387 | 0.467 |
| | Broccoli | 0.39 | 0.64 | 48.83 | 44.34 | 0.33 | 40 | 0.029 | 0.041 |
| | | 0.31 | 0.37 | 48.07 | 27.87 | 0.58 | 32 | 0.034 | 0.061 |
| | | **0.01** | **0.76** | **12.19** | **34.07** | **0.60** | **47** | **0.039** | **0.041** |
| | | 0.37 | 0.69 | 49.83 | 69.54 | 0.69 | 32 | 0.037 | 0.052 |
| | | 0.12 | 0.53 | 75.52 | 43.98 | 0.68 | 45 | 0.035 | 0.064 |
| | | 0.43 | 0.73 | 22.52 | 49.57 | 0.60 | 35 | 0.034 | 0.045 |
| | | 0.23 | 0.59 | 23.51 | 69.66 | 0.67 | 46 | 0.036 | 0.070 |
| | | 0.13 | 0.45 | 11.92 | 23.02 | 0.50 | 38 | 0.033 | 0.041 |
| | | 0.52 | 0.41 | 49.96 | 69.75 | 0.52 | 47 | 0.039 | 0.043 |
| | | 0.23 | 0.31 | 54.33 | 33.96 | 0.54 | 32 | 0.032 | 0.037 |
| | BroccoliD | **0.58** | **0.59** | **11.40** | **65.44** | **0.11** | **78** | **0.030** | **0.024** |
| | | 0.53 | 0.68 | 11.55 | 99.14 | 0.95 | 42 | 0.044 | 0.046 |
| | | 0.41 | 0.73 | 8.78 | 71.79 | 0.15 | 41 | 0.033 | 0.048 |
| | | 0.37 | 0.87 | 77.47 | 76.36 | 0.78 | 24 | 0.037 | 0.041 |
| | | 0.47 | 0.62 | 77.40 | 72.71 | 0.82 | 34 | 0.039 | 0.041 |
| | | 0.37 | 0.79 | 9.14 | 64.48 | 0.56 | 47 | 0.042 | 0.034 |
| | | 0.83 | 0.70 | 13.10 | 70.84 | 0.62 | 83 | 0.055 | 0.069 |
| | | 0.41 | 0.83 | 78.76 | 52.03 | 0.60 | 36 | 0.036 | 0.032 |
| | | 0.36 | 0.73 | 83.20 | 71.08 | 0.92 | 28 | 0.036 | 0.043 |
| | | 0.64 | 0.41 | 76.90 | 84.61 | 0.66 | 50 | 0.040 | 0.036 |
| U-ReNet | CWFID1 | 0.51 | 0.53 | 54.64 | 31.26 | 0.13 | 58 | 0.330 | 0.327 |
| | | 0.61 | 0.36 | 2.75 | 28.64 | 0.15 | 50 | 0.321 | 0.349 |
| | | 0.20 | 0.46 | 25.00 | 62.86 | 0.08 | 22 | 0.304 | 0.384 |
| | | **0.62** | **0.55** | **53.82** | **43.22** | **0.14** | **49** | **0.325** | **0.325** |
| | | 0.38 | 0.42 | 44.49 | 65.92 | 0.30 | 16 | 0.342 | 0.348 |
| | | 0.49 | 0.37 | 12.48 | 26.00 | 0.27 | 19 | 0.330 | 0.341 |
| | | 0.35 | 0.50 | 42.45 | 46.78 | 0.22 | 6 | 0.376 | 0.360 |
| | | 0.64 | 0.35 | 43.72 | 24.74 | 0.19 | 59 | 0.342 | 0.352 |
| | | 0.30 | 0.21 | 50.12 | 54.53 | 0.27 | 49 | 0.341 | 0.347 |
| | | 0.67 | 0.56 | 58.06 | 28.80 | 0.16 | 51 | 0.338 | 0.332 |
| | CWFID2 | **0.39** | **0.75** | **14.09** | **75.47** | **0.78** | **88** | **0.351** | **0.414** |
| | | 0.23 | 0.64 | 37.55 | 31.00 | 0.88 | 90 | 0.443 | 0.451 |
| | | 0.52 | 0.90 | 35.35 | 52.16 | 0.86 | 77 | 0.745 | 0.702 |
| | | 0.40 | 0.66 | 14.54 | 0.44 | 0.78 | 0 | 0.642 | 0.658 |
| | | 0.30 | 0.52 | 73.23 | 55.60 | 0.67 | 76 | 0.366 | 0.422 |
| | | 0.48 | 0.89 | 44.31 | 70.46 | 0.40 | 61 | 0.334 | 0.447 |
| | | 0.89 | 0.34 | 23.64 | 66.72 | 0.76 | 0 | 0.827 | 0.782 |
| | | 0.24 | 0.35 | 71.60 | 65.71 | 0.74 | 23 | 0.395 | 0.460 |
| | | 0.38 | 0.71 | 32.13 | 62.67 | 0.67 | 57 | 0.389 | 0.449 |
| | | 0.61 | 0.41 | 10.68 | 84.51 | 0.49 | 70 | 0.338 | 0.437 |
| | Broccoli | 0.43 | 0.32 | 88.60 | 36.03 | 0.13 | 11 | 0.021 | 0.056 |
| | | 0.54 | 0.35 | 84.98 | 16.22 | 0.57 | 22 | 0.029 | 0.045 |
| | | 0.24 | 0.48 | 48.03 | 17.42 | 0.32 | 50 | 0.028 | 0.045 |
| | | **0.40** | **0.52** | **88.30** | **26.95** | **0.88** | **57** | **0.037** | **0.049** |
| | | 0.66 | 0.57 | 53.01 | 20.76 | 0.09 | 71 | 0.026 | 0.034 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| U-ReNet | Broccoli | 0.47 | 0.25 | 44.80 | 25.35 | 0.42 | 57 | 0.029 | 0.044 |
| | | 0.26 | 0.46 | 49.92 | 20.94 | 0.09 | 56 | 0.026 | 0.035 |
| | | 0.29 | 0.44 | 84.64 | 44.48 | 0.25 | 65 | 0.028 | 0.041 |
| | | 0.53 | 0.33 | 89.13 | 25.97 | 0.79 | 28 | 0.036 | 0.054 |
| | | 0.52 | 0.24 | 81.50 | 45.44 | 0.41 | 70 | 0.030 | 0.041 |
| | BroccoliD | 0.55 | 0.78 | 71.24 | 41.58 | 0.45 | 53 | 0.021 | 0.039 |
| | | **0.56** | **0.31** | **59.35** | **14.73** | **0.57** | **86** | **0.025** | **0.038** |
| | | 0.27 | 0.75 | 27.91 | 45.52 | 0.51 | 79 | 1.010 | 0.931 |
| | | 0.20 | 0.57 | 57.58 | 28.89 | 0.48 | 52 | 0.021 | 0.045 |
| | | 0.26 | 0.57 | 31.47 | 31.87 | 0.48 | 76 | 0.027 | 0.058 |
| | | 0.56 | 0.51 | 31.89 | 33.00 | 0.56 | 87 | 1.010 | 0.931 |
| | | 0.62 | 0.34 | 71.59 | 46.32 | 0.59 | 40 | 0.026 | 0.046 |
| | | 0.78 | 0.56 | 51.81 | 28.14 | 0.44 | 56 | 0.024 | 0.052 |
| | | 0.54 | 0.49 | 62.42 | 46.44 | 0.50 | 57 | 0.023 | 0.045 |
| | | 0.36 | 0.55 | 12.81 | 33.95 | 0.59 | 85 | 0.040 | 0.066 |
| U-ReNet2 | CWFID1 | 0.26 | 0.54 | 71.39 | 68.38 | 0.35 | 52 | 0.364 | 0.360 |
| | | 0.39 | 0.56 | 15.67 | 30.17 | 0.35 | 24 | 0.527 | 0.408 |
| | | 0.46 | 0.73 | 86.64 | 70.17 | 0.29 | 60 | 0.415 | 0.383 |
| | | 0.27 | 0.57 | 75.61 | 10.25 | 0.24 | 39 | 0.361 | 0.346 |
| | | 0.51 | 0.42 | 94.44 | 83.08 | 0.13 | 79 | 0.350 | 0.365 |
| | | 0.66 | 0.50 | 72.44 | 54.91 | 0.22 | 59 | 0.389 | 0.363 |
| | | 0.29 | 0.73 | 34.45 | 36.94 | 0.10 | 78 | 0.367 | 0.355 |
| | | 0.82 | 0.26 | 9.69 | 78.47 | 0.34 | 57 | 0.557 | 0.447 |
| | | **0.56** | **0.59** | **89.49** | **19.06** | **0.10** | **65** | **0.348** | **0.324** |
| | | 0.32 | 0.46 | 11.37 | 48.29 | 0.17 | 29 | 0.325 | 0.362 |
| | CWFID2 | 0.42 | 0.58 | 7.23 | 21.41 | 0.42 | 51 | 0.634 | 0.685 |
| | | 0.03 | 0.60 | 77.13 | 67.31 | 0.68 | 9 | 1.057 | 1.079 |
| | | **0.12** | **0.14** | **52.31** | **63.50** | **0.45** | **77** | **0.645** | **0.719** |
| | | 0.37 | 0.07 | 12.73 | 45.36 | 0.98 | 9 | 1.081 | 1.094 |
| | | 0.04 | 0.06 | 19.18 | 54.82 | 0.78 | 6 | 0.834 | 0.800 |
| | | 0.30 | 0.06 | 50.07 | 17.02 | 0.76 | 60 | 1.078 | 1.022 |
| | | 0.14 | 0.32 | 9.94 | 34.40 | 0.86 | 59 | 0.918 | 0.983 |
| | | 0.11 | 0.27 | 0.37 | 59.53 | 0.59 | 82 | 0.895 | 1.101 |
| | | 0.01 | 0.51 | 74.14 | 39.75 | 0.45 | 87 | 0.653 | 0.716 |
| | | 0.39 | 0.28 | 19.43 | 55.87 | 0.66 | 41 | 0.658 | 0.692 |
| | Broccoli | 0.48 | 0.10 | 20.23 | 34.34 | 0.16 | 63 | 0.026 | 0.037 |
| | | 0.46 | 0.44 | 21.31 | 49.46 | 0.11 | 45 | 0.024 | 0.036 |
| | | 0.52 | 0.43 | 65.79 | 27.96 | 0.12 | 79 | 0.025 | 0.036 |
| | | **0.31** | **0.40** | **28.23** | **54.86** | **0.03** | **57** | **0.022** | **0.028** |
| | | 0.63 | 0.54 | 50.06 | 36.86 | 0.14 | 63 | 0.027 | 0.041 |
| | | 0.69 | 0.31 | 57.14 | 34.90 | 0.25 | 64 | 0.028 | 0.044 |
| | | 0.39 | 0.11 | 89.04 | 22.45 | 0.12 | 67 | 0.025 | 0.043 |
| | | 0.58 | 0.52 | 20.25 | 49.08 | 0.09 | 93 | 0.026 | 0.033 |
| | | 0.48 | 0.27 | 71.15 | 42.30 | 0.10 | 60 | 0.025 | 0.041 |
| | | 0.47 | 0.09 | 81.74 | 24.16 | 0.06 | 61 | 0.024 | 0.032 |
| | BroccoliD | 0.22 | 0.80 | 11.72 | 66.36 | 0.41 | 14 | 0.021 | 0.046 |
| | | 0.26 | 0.49 | 65.68 | 86.41 | 0.57 | 86 | 1.011 | 0.931 |
| | | 0.46 | 0.82 | 12.84 | 99.01 | 0.40 | 95 | 0.025 | 0.041 |
| | | 0.47 | 0.75 | 18.95 | 89.64 | 0.60 | 95 | 0.029 | 0.042 |
| | | 0.39 | 0.34 | 82.05 | 72.13 | 0.57 | 60 | 0.028 | 0.058 |
| | | **0.20** | **0.52** | **11.38** | **59.09** | **0.53** | **98** | **0.030** | **0.038** |
| | | 0.42 | 0.74 | 19.69 | 50.07 | 0.50 | 38 | 0.027 | 0.063 |
| | | 0.20 | 0.88 | 75.74 | 89.56 | 0.40 | 10 | 0.021 | 0.045 |
| | | 0.44 | 0.56 | 17.83 | 88.82 | 0.42 | 12 | 0.024 | 0.057 |
| | | 0.48 | 0.53 | 73.85 | 52.58 | 0.62 | 34 | 0.027 | 0.058 |
| ReSeg | CWFID1 | 0.64 | 0.26 | 84.60 | 64.99 | 0.68 | 74 | 0.431 | 0.359 |
| | | **0.28** | **0.27** | **77.79** | **84.04** | **0.21** | **58** | **0.342** | **0.299** |
| | | 0.29 | 0.48 | 28.57 | 98.05 | 0.45 | 87 | 0.379 | 0.316 |
| | | 0.16 | 0.38 | 18.72 | 68.53 | 0.66 | 79 | 0.421 | 0.364 |
| | | 0.25 | 0.18 | 87.86 | 95.04 | 0.75 | 79 | 0.563 | 0.402 |
| | | 0.22 | 0.54 | 87.52 | 36.99 | 0.86 | 62 | 0.930 | 0.855 |
| | | 0.38 | 0.50 | 88.72 | 89.88 | 0.76 | 83 | 0.670 | 0.472 |
| | | 0.62 | 0.69 | 14.04 | 21.53 | 0.71 | 68 | 0.636 | 0.443 |
| | | 0.88 | 0.55 | 27.69 | 22.68 | 0.43 | 62 | 0.397 | 0.365 |
| | | 0.89 | 0.39 | 78.61 | 17.43 | 0.34 | 67 | 0.377 | 0.328 |
| | CWFID2 | 0.38 | 0.76 | 11.62 | 57.59 | 0.71 | 85 | 0.995 | 0.995 |
| | | 0.54 | 0.34 | 17.44 | 22.07 | 0.42 | 37 | 0.878 | 0.868 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ReSeg | CWFID2 | 0.55 | 0.26 | 20.69 | 55.56 | 0.33 | 18 | 0.842 | 0.934 |
| | | 0.49 | 0.79 | 26.83 | 62.51 | 0.26 | 64 | 0.774 | 0.799 |
| | | 0.61 | 0.64 | 19.34 | 89.33 | 0.61 | 58 | 0.852 | 1.047 |
| | | 0.48 | 0.27 | 10.36 | 87.34 | 0.65 | 55 | 0.871 | 0.945 |
| | | **0.58** | **0.42** | **11.28** | **60.40** | **0.15** | **16** | **0.732** | **0.759** |
| | | 0.15 | 0.75 | 27.52 | 80.89 | 0.56 | 19 | 0.847 | 0.916 |
| | | 0.19 | 0.52 | 23.92 | 29.64 | 0.70 | 64 | 0.876 | 0.933 |
| | | 0.59 | 0.55 | 25.42 | 49.41 | 0.63 | 44 | 0.925 | 0.956 |
| | Broccoli | 0.45 | 0.79 | 52.61 | 53.44 | 0.11 | 39 | 0.042 | 0.041 |
| | | 0.32 | 0.50 | 27.26 | 70.04 | 0.10 | 23 | 0.040 | 0.037 |
| | | 0.26 | 0.68 | 24.35 | 76.63 | 0.13 | 56 | 0.041 | 0.038 |
| | | 0.42 | 0.78 | 57.58 | 30.03 | 0.13 | 77 | 0.041 | 0.037 |
| | | 0.25 | 0.42 | 50.95 | 55.37 | 0.11 | 60 | 0.042 | 0.041 |
| | | 0.41 | 0.45 | 15.62 | 77.65 | 0.10 | 23 | 0.042 | 0.043 |
| | | 0.36 | 0.63 | 55.96 | 82.82 | 0.13 | 33 | 0.042 | 0.040 |
| | | **0.49** | **0.75** | **52.23** | **50.90** | **0.10** | **46** | **0.040** | **0.036** |
| | | 0.31 | 0.79 | 53.35 | 34.04 | 0.11 | 65 | 0.040 | 0.036 |
| | | 0.21 | 0.48 | 27.71 | 88.37 | 0.15 | 19 | 0.039 | 0.043 |
| | BroccoliD | 0.19 | 0.47 | 92.69 | 36.93 | 0.16 | 50 | 0.037 | 0.034 |
| | | 0.17 | 0.39 | 73.57 | 42.23 | 0.13 | 53 | 0.038 | 0.033 |
| | | 0.15 | 0.33 | 98.89 | 81.43 | 0.18 | 58 | 0.038 | 0.035 |
| | | **0.27** | **0.60** | **16.35** | **18.04** | **0.02** | **50** | **0.034** | **0.029** |
| | | 0.42 | 0.31 | 83.37 | 42.51 | 0.14 | 75 | 0.041 | 0.035 |
| | | 0.58 | 0.26 | 91.76 | 83.01 | 0.10 | 59 | 0.041 | 0.033 |
| | | 0.02 | 0.45 | 26.05 | 65.59 | 0.01 | 61 | 0.033 | 0.029 |
| | | 0.53 | 0.42 | 20.51 | 22.30 | 0.06 | 53 | 0.037 | 0.029 |
| | | 0.16 | 0.35 | 25.66 | 95.05 | 0.11 | 69 | 0.042 | 0.036 |
| | | 0.09 | 0.43 | 17.54 | 88.45 | 0.00 | 57 | 0.035 | 0.030 |
| ReConv | CWFID1 | 0.76 | 0.53 | 83.40 | 23.58 | 0.08 | 59 | 0.349 | 0.453 |
| | | **0.29** | **0.72** | **11.18** | **13.03** | **0.11** | **50** | **0.398** | **0.383** |
| | | 0.84 | 0.61 | 21.63 | 62.71 | 0.10 | 72 | 0.463 | 0.434 |
| | | 0.54 | 0.62 | 82.86 | 66.20 | 0.09 | 42 | 0.425 | 0.388 |
| | | 0.33 | 0.66 | 12.38 | 75.38 | 0.12 | 53 | 0.452 | 0.398 |
| | | 0.36 | 0.44 | 78.81 | 39.62 | 0.06 | 64 | 0.413 | 0.408 |
| | | 0.80 | 0.40 | 34.24 | 82.58 | 0.03 | 45 | 0.354 | 0.456 |
| | | 0.59 | 0.29 | 20.75 | 50.41 | 0.19 | 55 | 0.445 | 0.448 |
| | | 0.85 | 0.63 | 11.90 | 33.42 | 0.11 | 69 | 0.414 | 0.389 |
| | | 0.81 | 0.34 | 76.71 | 68.59 | 0.05 | 67 | 0.334 | 0.474 |
| | CWFID2 | **0.67** | **0.53** | **60.53** | **51.01** | **0.01** | **49** | **0.330** | **0.315** |
| | | 0.78 | 0.87 | 11.94 | 79.45 | 0.21 | 67 | 0.898 | 0.974 |
| | | 0.37 | 0.70 | 80.13 | 50.70 | 0.08 | 44 | 0.913 | 0.935 |
| | | 0.52 | 0.47 | 63.07 | 37.35 | 0.18 | 52 | 0.909 | 0.933 |
| | | 0.60 | 0.39 | 61.50 | 93.35 | 0.21 | 40 | 0.916 | 0.990 |
| | | 0.82 | 0.31 | 82.42 | 89.43 | 0.04 | 34 | 0.901 | 0.947 |
| | | 0.36 | 0.62 | 60.34 | 38.54 | 0.16 | 45 | 0.908 | 0.928 |
| | | 0.74 | 0.44 | 80.62 | 83.55 | 0.16 | 64 | 0.918 | 0.973 |
| | | 0.73 | 0.53 | 81.35 | 50.38 | 0.16 | 64 | 0.922 | 1.008 |
| | | 0.53 | 0.49 | 20.76 | 53.83 | 0.14 | 47 | 0.899 | 0.965 |
| | Broccoli | 0.77 | 0.46 | 47.54 | 69.32 | 0.38 | 96 | 0.074 | 0.063 |
| | | 0.28 | 0.17 | 95.71 | 65.19 | 0.74 | 82 | 0.077 | 0.064 |
| | | 0.38 | 0.78 | 45.69 | 56.48 | 0.52 | 65 | 0.074 | 0.070 |
| | | 0.76 | 0.27 | 81.04 | 44.97 | 0.23 | 78 | 0.074 | 0.061 |
| | | 0.66 | 0.14 | 42.72 | 66.25 | 0.23 | 60 | 0.074 | 0.060 |
| | | 0.45 | 0.17 | 43.51 | 42.08 | 0.36 | 79 | 0.075 | 0.062 |
| | | 0.73 | 0.46 | 81.31 | 56.43 | 0.68 | 68 | 0.075 | 0.081 |
| | | 0.23 | 0.87 | 43.25 | 42.72 | 0.33 | 62 | 0.076 | 0.062 |
| | | 0.22 | 0.75 | 89.92 | 54.97 | 0.77 | 60 | 0.074 | 0.066 |
| | | **0.49** | **0.70** | **83.80** | **67.46** | **0.43** | **95** | **0.075** | **0.062** |
| | BroccoliD | 0.36 | 0.33 | 25.39 | 26.11 | 0.25 | 50 | 0.077 | 0.067 |
| | | 0.51 | 0.56 | 79.63 | 46.99 | 0.32 | 84 | 0.078 | 0.078 |
| | | 0.46 | 0.88 | 82.03 | 34.22 | 0.35 | 67 | 0.084 | 0.072 |
| | | 0.50 | 0.28 | 16.48 | 45.83 | 0.20 | 74 | 0.076 | 0.063 |
| | | 0.61 | 0.45 | 19.68 | 44.42 | 0.23 | 69 | 0.075 | 0.067 |
| | | 0.47 | 0.76 | 8.70 | 86.38 | 0.33 | 66 | 0.088 | 0.076 |
| | | 0.73 | 0.57 | 8.19 | 40.89 | 0.33 | 72 | 0.086 | 0.069 |
| | | 0.37 | 0.82 | 19.01 | 83.59 | 0.31 | 72 | 0.078 | 0.065 |
| | | **0.37** | **0.27** | **20.88** | **34.92** | **0.28** | **82** | **0.077** | **0.062** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ReConv | BroccoliD | 0.71 | 0.78 | 22.28 | 41.84 | 0.39 | 78 | 0.080 | 0.065 |

The training has then been performed by using the found hyperparameters and dividing the learning rate so to avoid selecting a too high value. The learning rate has been divided by a factor of 2 or 5 depending on the dataset.