# Machine learning applied to aerial images for vegetation detection

Yasser Salah Eddine Bouchareb (yasserb@aims.ac.za)
يَاسِر صَلَاح الدِّينْ الأَيُوبِي بُوشَارِب
African Institute for Mathematical Sciences (AIMS)

Supervised by: Dr. Jonathan Shock
University of Cape Town, South Africa

Co-Supervised by: A/Prof. Adam West
University of Cape Town, South Africa

24 May 2018

*Submitted in partial fulfillment of a structured masters degree at AIMS South Africa*

AIMS | African Institute for Mathematical Sciences SOUTH AFRICA

## Abstract

Drone images of vegetation have been taken by department of Botany within UCT. In this work we briefly describe different techniques in image processing, to deal with those high-resolution images and extract regions of interest. Furthermore, for accurate and fast results, we use the produced images by the image processing tasks, to train a convolutional neural network (CNN) model to segment these images and identify large shrubs.

Evaluation of this model is based on true positive rate and false positive rate, but we did look at the accuracy and the loss as well.

**Keywords:** Feature Extraction, Classification, Image segmentation, Morphology, Convolutional Neural Networks, Pixel-Wise Segmentation.

## الملخص

بِاسْتِخْدَامِ الطَّائِرَاتِ الذَّاتِيَّةِ أَخَذَ قِسْمُ الْبِيُولُوجِيَا فِي جَامِعَةِ كايب تَاونَ صُورِلجُزْءٍ مِنَ الْغِطَاءِ النَّبَاتِيِّ فِي جَنُوبِ افريقيا. فِي هَذَا الْعَمَلِ، نَقُومُ بِمُنَاقَشَةِ تِقْنِيَّاتٍ مُخْتَلِفَةٍ فِي مُعَالَجَةِ الصُّورِ، وَنَسْتَعْمِلُهَا لِإِسْتِخْرَاجِ اِهْمِ الْمَعْلُومَاتِ فِي الصُّورِ. عِلَاوَةٌ عَلَى ذَلِكَ، مِنْ أَجَلِ الْحُصُولَ عَلَى نَتَائِجِ دَقِيقَةٍ وَسَرِيعَةٍ، نَسْتَخْدِمُ الصُّورَ النَّاتِجَةَ مِنْ مُعَالَجَةِ الصُّورِ لِتَدْرِيبِ نَمُوذَجِ شَبَكَةِ عَصَبِيَّةٍ تَلَافِيفِيَّةٍ عَلَى تَقْسِيمِ هَذِهِ الصُّورِ وَتَحْدِيدِ الشُّجَيْرَاتِ الْكَبِيرَةِ. يَعْتَمِدُ تَقْيِيمُ هَذَا النَّمُوذَجِ عَلَى الْمُعَدَّلِ الْإِيجَابِيِّ الصَّحِيحِ وَالْمُعَدَّلِ الْإِيجَابِيِّ الْخَاطِئِ، لَكِنَّ لِمَزِيدٍ مِنَ التَّوْضِيحِ نَظَرْنَا إِلَى الدِّقَّةِ وَالْخَسَارَةِ أَيْضًا.

**الْكَلِمَاتُ الْمِفْتَاحِيَّةُ:** اِسْتِخْرَاجُ اِهْمِ الْمَيْزَاتِ، التَّصْنِيف، تَجْزِئَةَ الصُّورَةِ، مُورْفُولُوجِيَا، شَبَكَاتٍ عَصَبِيَّةٍ تَلَافِيفِيَّةٍ، التَّقْسِيمَ بالبكسل.

## Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

[illegible]

Yasser Salah Eddine Bouchareb, 24 May 2018

# Contents

Abstract i

1 Introduction 1

2 Data Preprocessing 3
- 2.1 Model Architecture . . . . . 3
- 2.2 Image Preprocessing (IP) . . . . . 3
- 2.3 Mathematical Morphology . . . . . 5
- 2.4 Features Extraction and Classification . . . . . 7

3 Deep Neural Network 9
- 3.1 Machine learning . . . . . 9
- 3.2 Traditional Neural Networks . . . . . 9
- 3.3 Convolutional Neural Network (CNN) . . . . . 17

4 Results and Discussion 20
- 4.1 Data Collection . . . . . 20
- 4.2 Image Processing . . . . . 20

5 Conclusion and Future Work 29

References 32

# 1. Introduction

Computer Vision is the use of computers to extract relevant information of the physical world from images (Xavier, 2003). Human visual system is incredible, and the goal of computer vision is to give computers the ability to understand a picture in much the same way that a human observer can.

Human visual system is capable of grouping pixels in an image with the same characteristic into a set of regions. This basic ability is usually referred as segmenting the image. Simple pixel property are used as basis for the segmentation task, such as grey level or colour.

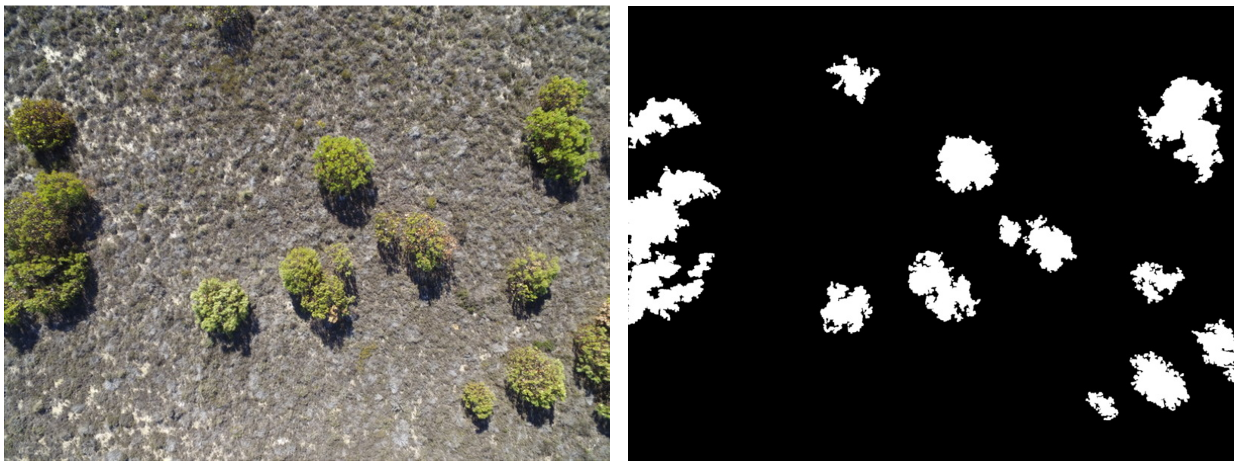

Figure 1.1: Image segmentation. The original image on the left is partitioned into two meaningful regions, which are white for shrubs and black elsewhere, in the segmented image on the right

Figure 1.1 is a simple example of image segmentation. The original image on the left is a sample of a large data set collected by Miss. Simon, H.R. and her supervisors Prof. West, A.G. and Dr. Slingsby, J (Simon, 2017) for their studies. This data was used to clarify the ambiguity on the seasonal fluctuations in vegetation greenness (Normalised Difference Vegetation Index, NDVI) within South Africa's hyperdiverse fynbos biome, between on-the-ground and remotely-sensed findings. Such changes have been presumed to be the consequences of varying water availability between the wet and dry months, but may also be related to a species phenological, functional and life-history traits.

The hypothesis was that plants in the fynbos would exhibit diverse overall and seasonal NDVI signals. To approach this problem one need to differentiate between different species (Protea repens, Protea amplexicaulis, etc...). Our goal in this work is to use the segmentation task to identify one of these species in a given image, which is the large shrubs.

To do that we need to detect and extract regions of interest, that usually refer to segmentation, although it is difficult to conceive, we can think of image segmentation as the first look that we made at the world when we were newborn. In image segmentation contrary to the classification problems, recognition of these regions in not required. So, the answer of the image segmentation process will be something as: "there are four shrubs in the image".

In literature, various segmentation approaches have been proposed. And always there is new tasks to consider. Different segmentation techniques use different properties of the image, some of them use the gray intensities, some use edges and spatial details, while others combine different technique, and many more. Researchers are always trying to come up with new methods. Some works have been done using the following:

- Semantic Texton Forests (Shotton et al., 2008) are randomized decision forests used for both clustering and classification. They create batches centred at each pixel in the image using specific split function, and then pass them through the Semantic texton forests (STFs).
- Snakes and Live-Wire boundaries (Marsh, 2014), both techniques uses edge information and spatial detail to perform segmentation.
- Markov Random Field model (Li, 1994).
- Fuzzy and Non-Fuzzy techniques (PAL and PAL, 1993).

Most of the techniques we mentioned above are not suitable for noisy environments, or they do not perform good for objects where boundaries are not well defined. Neural network architectures which we adapt in this study have also been used for image segmentation (Ronneberger et al., 2015) and they did well even when high level of noise is present in the image. One of the advantage of neural network is the parallel processing ability, which help to get the results in a real time.

The rest of the essay will focus on the mathematical aspect of our model. Chapter 2 briefly reviews different preliminary tasks applied to an image in order to reduce noise, to extract useful information, and introduces the basic concepts in image segmentation. Chapter 3 presents the classification techniques to be used. Also, the performance results on the dataset of our model are reported in Chapter 4. Finally, Chapter 5 highlights some future research directions and concludes the essay.

# 2. Data Preprocessing

Images can be quite noisy, and that depend on the types of cameras used, sun rays and so on, and that highly affect the result of the segmentation. Before we perform classification, we need to pass through few preliminary steps namely, Image Preprocessing (IP), Image Segmentation (IS) and Feature Extraction (FE) as shown in Figure 2.1. Let us look at the role of each step and define the properties required according to our work that is going to lead us to an optimal choice of mathematical technique.

## 2.1 Model Architecture

The general steps of our model are: Image preprocessing, image segmentation, feature extraction, and classification (see Figure 2.1). This chapter briefly discusses each step of the diagram.

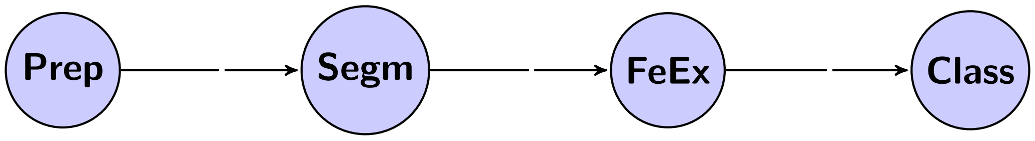

Figure 2.1: The model. Prep: Image preprocessing, Segm: Image segmentation, FeEx: Features extraction, Class: Per-pixel classification

## 2.2 Image Preprocessing (IP)

The procedure of extracting the maximum useful informations from an image by performing operations using variety of techniques (like finding a particular shape in it, or removing noise, etc...), is refer to as Image Processing. There are plenty methods of manipulating images, and we will go only through few of them.

**2.2.1 Segmentation.** In finding shrubs in images, we will first need to break it up into features of interest. This is called image segmentation (IS). More precisely, IS is the process of separating an object from it background, and this is done by labelling the pixels in a way that pixels with the same label should have certain common characteristic.

Let $\mathcal{I}$ be the set of image pixels. The shrubs in the image could be represented as a subset $T \subset \mathcal{I}$, where the elements of $T$, $t \in T$ can be seen as disks with centre $x_t$ and radius $r_t$, $t = D(x_t, r_t)$. Thus pixels belong to the same shrub lie inside the disk.

**2.2.2 Blurring.** As we stated before, images are prone to contain high level of noise, which result in small problems when processing the image. The noise in image $\mathcal{I}$, can be represented as a set of disks with centre $x_n$ and radius $r_n$ such that $r_n \ll r_t$.

In order to get rid of this noise we blur the image. Blurring is one of the common way to do that, where we smooth out high intensities and make extreme changes between pixels visually indistinct. A popular way to do that is to use the Gaussian blur also known as the Gaussian filter, where we apply a transformations to each pixel of the image and get the pixel new value. The Gaussian function $g$

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where, $\mu$ is the mean of the Gaussian distribution (GD), and $\sigma$ is the standard deviation (SD). Working with images needs to use the two dimensional Gaussian function, which is simply the product of two 1-D Gaussian functions since the variable are independent, and it is given by:

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

where $x$ and $y$ are the coordinate of the pixel in the image, and $\sigma$ is the SD of the distribution. The distribution is assumed to be centred i.e. have a mean of $\mu = 0$.

The transformations in the blurring are done by convolving (See Section 3.3.1) the 2-D GD with the image, and to do that we need a discrete approximation to the Gaussian function. However the Gaussian function is non zero-every where and that theoretically require an infinity large convolutional kernel. Luckily the distribution approach zero at about 3 SD from the centre, so we can limit the kernel to contain only the values within 3 SD from the centre. Kernels or filters are matrices with dimension less than the original image, they operate on the image to extract useful informations. They have well define shape, size and origin.

The weights of the kernel are calculated by numerical integration of the continuous GD over each discrete kernel tap[1]. Note that different algorithms perform with a slightly different Gaussian blur kernel, and the choice of the SD is extremely application dependent. For this application we used the following function,

$$G_i = \alpha * e^{\frac{-(i-(n-1)/2)^2}{(2*\sigma)^2}}, \text{ for } i = 0 \ldots n-1. \tag{2.2.1}$$

In Equation (2.2.1), $\alpha$ is the scale factor chosen so that $\sum_{i=0}^{n-1} G_i = 1$, and $n$ is the size of the kernel and it is recommended to use an odd number.

So on a Gaussian blur with a $3 \times 3$ kernel, for every pixel, we perform an averaging calculation of the $3 \times 3$ surrounding pixels, which result in the new value of the pixels. This new value is the blurred colour, so big size of kernel implies more blurred image.

The difference between this and the mean blur is that we are really prioritising the pixels in the middle, so the further away we get from the pixel of interest, the less weight we have in the combined average. Actually, that is what motivate us to chose the Gaussian filter, because in our work we want the pixels belonging to the set of shrubs to have the priority. Also the Gaussian kernel is separable, which allows fast computation. Moreover, when we have noisy images and we want to perform thersholding a Gaussian blurring is very useful to smooth over all of those irregularities.

**2.2.3 Binarization and Thresholding.** A simple way of segmenting an image is to perform a thresholding. The basic idea of thresholding is to get two sets of pixels $\mathcal{W}$ and $\mathcal{B}$, $\mathcal{W}$ for white pixels (foreground) and $\mathcal{B}$ for black pixels (background) by replacing each pixel with a white pixel if gray scale intensities of that pixel $g(x,y)$ exceeds a certain threshold $\theta$, and replace it with a black pixel if it does not. The result of this replacement is a binary image $f(x,y)$, we express it mathematically as follow.

$$f(x,y) = \begin{cases} 1 \text{ if } g(x,y) > \theta, \text{ i.e. } g(x,y) \in \mathcal{W}, \\ 0 \text{ if } g(x,y) \leq \theta, \text{ i.e. } g(x,y) \in \mathcal{B}. \end{cases}$$

[1]Gaussian kernel is separable so usually we compute the 1-D kernel which is a vector, and use it to generate the whole matrix, a kernel tap is an element of the 1-D vector

## 2.3 Mathematical Morphology

During segmentation, let us consider an image with two objects and we segment them by setting some rule that anything more than this pixel value is foreground and anything lower than this is background. Let us consider that these are two shrubs that belongs to the foreground. Now let us assume a scenario where this two shrubs are supposed to be independent sub curves but for some unfortunate reason of the way the images were taken as poor quality and all, that the two shrubs are touching each other but we want to separate them out. How would you do that? Morphology.

The word morphology refers to the study of the forms of things and their structures. Where morphology is a non-linear operation that modifies the shape or the morphology of objects in an image using a structuring element (Kofi, 2014) when it come to image processing.

In literature, images usually represented by a set of pixels, denote $\mathcal{I}$. Pixels set which constitute specific shape is called structuring element, and denote $\mathcal{B}$. The set $\mathcal{B}$ is characterized by a well defined shape, size, and origin, and its shape can be regarded as a hyper parameter to the morphological operation. The Morphological operation uses both sets to analysed the original image $\mathcal{I}$, and transform it to a new set of pixels with certain properties. This transformation is done by intersecting both $\mathcal{I}$ and $\mathcal{B}$, where only features in $\mathcal{I}$ similar in shape and size to the ones in $\mathcal{B}$ are considered. Morphological operation help us simplify the images, by suppressing irrelevant features. This technique is widely used for image segmentation and edge detection.

In the following, let the binary image $\mathcal{I}$ be a closed subset of the discrete space $\mathbb{Z}^2 : \mathcal{I} \subset \mathbb{Z}^2$, and $\mathcal{B}$ the structuring element be subsets of the discrete space $\mathbb{Z}^2, \mathcal{B} \subset \mathbb{Z}^2$.

**2.3.1 Definition.** Let $v \in \mathbb{Z}^2$. The translation of $\mathcal{I}$ by $v$ is defined as

$$\mathcal{I}_v = \{x + v : \ \forall x \in \mathcal{I}\}.$$

**2.3.2 Definition.** The Reflection $\hat{\mathcal{I}}$ is the symmetrical set of $\mathcal{I}$ with respect to the origin i.e

$$\hat{\mathcal{I}} = \{-x : \ x \in \mathcal{I}\}.$$

**2.3.3 Dilation.** The set which is formed by adding all the vectors of two sets is denoted the dilated set. If vectors $a$ and $b$ belong to the sets $\mathcal{I}$ and $\mathcal{B}$, respectively, then the dilated set of $\mathcal{I}$ by $\mathcal{B}$ will consist of all vectors $c = a + b \in \mathbb{Z}^2$. We denoted by $\mathcal{D}_{\mathcal{B}}(\mathcal{I})$, and it is defined as

$$\mathcal{D}_{\mathcal{B}}(\mathcal{I}) = \{a + b : a \in \mathcal{I}, \ b \in \mathcal{B}\}. \tag{2.3.1}$$

The name of this operation is coming from the fact that image features dilate and grow in size due to the translation.

From equation (2.3.1) above, we can consider dilation as the union of all translates of set $\mathcal{I}$ by $\mathcal{B}$.

$$\mathcal{D}_{\mathcal{B}}(\mathcal{I}) = \bigcup_{b \in \mathcal{B}} \mathcal{I}_b. \tag{2.3.2}$$

The dilation process is done by putting the origin of the structuring element on each pixel of the foreground set $\mathcal{W}$ in the original image, and then take the union of all the resulting subsets of $\mathcal{I}$.

**2.3.4 Erosion.** Erosion is the morphological dual to dilation. Where as the dilated set is formed by vector addition of two sets, the eroded set is formed by vector subtraction. The eroded set is the set of vectors $c = a - b \in \mathbb{Z}^2$, for which all of the vectors $b \in \mathcal{B}$ there exists an $a \in \mathcal{I}$.

The erosion of $\mathcal{I}$ by $\mathcal{B}$ denoted by $\mathcal{E}_{\mathcal{B}}(\mathcal{I})$, is defined as

$$\mathcal{E}_{\mathcal{B}}(\mathcal{I}) = \{a - b : \ \forall b \in \mathcal{B}, \exists a \in \mathcal{I}\}, \tag{2.3.3}$$

or, the set of $z$ such that the structuring element translated by $z$ fits fully inside $\mathcal{I}$,

$$\mathcal{E}_{\mathcal{B}}(\mathcal{I}) = \{z | \ \mathcal{B}_z \subset \mathcal{I}\}. \tag{2.3.4}$$

The erosion generally causes image objects to shrink in size, or erode - hence the name.

From equation (2.3.4) above, we can consider dilation as the intersection of the negative translates of set $\mathcal{I}$ by $\mathcal{B}$.

$$\mathcal{E}_{\mathcal{B}}(\mathcal{I}) = \bigcap_{b \in \mathcal{B}} \mathcal{I}_{-b},$$

The Erosion process is done by placing the origin of $\mathcal{B}$ on each pixel of the foreground set $\mathcal{W}$ in the original image. And then if the whole structuring element fit in the foreground set $\mathcal{W}$ we translate to the next pixel, otherwise we remove that pixel.

The Dilation and Erosion transforms also have the following common properties:

1. **Translation Invariance** Let $x \in \mathcal{I}$, $x \in \mathcal{B}$. Both dilation and erosion are translation invariant by $x$:
$$\mathcal{D}_{\mathcal{B}_x}(\mathcal{I}) = \mathcal{D}_{\mathcal{B}}(\mathcal{I}_x) = (\mathcal{D}_{\mathcal{B}}(\mathcal{I}))_x.$$
$$\mathcal{E}_{\mathcal{B}_x}(\mathcal{I}) = \mathcal{E}_{\mathcal{B}}(\mathcal{I}_x) = (\mathcal{E}_{\mathcal{B}}(\mathcal{I}))_x.$$

2. **Increasing in** $\mathcal{I}$ Let $\mathcal{I}_1$ and $\mathcal{I}_2$ denote images, such that $\mathcal{I}_1 \subset \mathcal{I}_2$, both dilation and erosion are increasing in $\mathcal{I}$
$$\mathcal{D}_{\mathcal{B}}(\mathcal{I}_1) \subset \mathcal{D}_{\mathcal{B}}(\mathcal{I}_2) \ .$$
$$\mathcal{E}_{\mathcal{B}}(\mathcal{I}_1) \subset \mathcal{E}_{\mathcal{B}}(\mathcal{I}_2) \ .$$

3. **Decreasing in** $\mathcal{B}$ Let $\mathcal{B}_1$ and $\mathcal{B}_2$ denote structuring elements such that $\mathcal{B}_1 \subset \mathcal{B}_2$, erosion is decreasing in $\mathcal{B}$
$$\mathcal{E}_{\mathcal{B}_1}(\mathcal{I}) \supset \mathcal{E}_{\mathcal{B}_2}(\mathcal{I}) \ .$$

4. **Duality** Dilation and erosion are duals
$$(\mathcal{D}_{\mathcal{B}}(\mathcal{I}))^c = \mathcal{E}_{\hat{\mathcal{B}}}(\mathcal{I}^c) \ .$$
$$(\mathcal{E}_{\mathcal{B}}(\mathcal{I}))^c = \mathcal{D}_{\hat{\mathcal{B}}}(\mathcal{I}^c) \ .$$

5. **Non-Inverse** Dilation and erosion are not the inverse of each other
$$\mathcal{E}_{\mathcal{B}}(\mathcal{D}_{\mathcal{B}}(\mathcal{I})) \neq \mathcal{I}.$$
$$\mathcal{D}_{\mathcal{B}}(\mathcal{E}_{\mathcal{B}}(\mathcal{I})) \neq \mathcal{I}.$$
Both dilation and erosion are nonlinear operations, and are generally non-invertible.

This two operations alone are powerful but we can combine them with other operations and get a magical way of cleaning images. Next we are going to introduce the combinations operations which are opening and closing.

**2.3.5 Opening.** Opening is formed by first eroding the set $\mathcal{I}$, after which this eroded set, $\mathcal{E}_{\mathcal{B}}(\mathcal{I})$, is dilated. Note that the structuring element $\mathcal{B}$ used is the same for the two operations.

The Opening of $\mathcal{I}$ by $\mathcal{B}$ is defined as

$$\mathcal{O}_{\mathcal{B}}(\mathcal{I}) = \mathcal{D}_{\mathcal{B}}(\mathcal{E}_{\mathcal{B}}(\mathcal{I})).$$

With the opening of an image, contours of objects are smoothed, narrow isthmuses are cut, and small islands and sharp peaks are removed.

The opening of $\mathcal{I}$ by $\mathcal{B}$ is obtained by taking the union of all the translates of $\mathcal{B}$ which fit into $\mathcal{I}$.

$$\mathcal{O}_{\mathcal{B}}(\mathcal{I}) = \bigcup_{b \in \mathcal{B}} \{b + x | b + x \subset \mathcal{I}\}.$$

**2.3.6 Closing.** Analogous to relationship between dilation and erosion, closing is the morphological dual to opening. The difference between opening and closing lies in the order in which the erosion and dilation transforms are applied. Closing is formed by first dilating a set $\mathcal{I}$, after which this dilated set, $\mathcal{D}_{\mathcal{B}}(\mathcal{I})$, is eroded. The Closing of $\mathcal{I}$ by $\mathcal{B}$ is defined as

$$\mathcal{C}_{\mathcal{B}}(\mathcal{I}) = \mathcal{E}_{\mathcal{B}}(\mathcal{D}_{\mathcal{B}}(\mathcal{I})). \tag{2.3.5}$$

Closing an image result in filling the holes.

We said that an image $\mathcal{I}$ is open when it does not change when applying opening, and close when it is unchanged under closing. It is worth noticing that opening and closing are not inverses. In general, it is rarely to get back the original image when applying opening and then closing or the inverse way.

**2.3.7 Preprocess Images.** Using the tools box of the image preprocessing task we mentioned above, we start by performing a Gaussian blur to reduce the noise in the image. Then applying a coloured thresholding will pick up regions of interest. One can use Gaussian filter again after thresholding for more efficiency. Convert the result to the gray scale level and binarize using mean pixel value will only show high intensities pixels, thus we can use them to identify shrubs we are interested in.

**2.3.8 Find Contours.** Unfortunately, after the whole process we did above, we could not achieve the wanted results, due to large noise in the images. That is why we introduced the Contours.

We can consider contours as curves bounding points that have the same colour or gray intensity, they are very useful for object detection. And since the noise in the image is represented by a shape, a set of disks with radius $r_c \ll \alpha$ where $\alpha$ is some threshold for the radius of shrubs. We only consider the curves in $\mathcal{I}$ with radius large than $\alpha$. Thus we get ride of the remaining noise.

## 2.4 Features Extraction and Classification

The main object of this work is to segment the shrubs out of a given image, and to do that we need to extract relevant information from the image, that is the goal of introducing feature extractor (FE). FE receives an image $\mathcal{I}$ in its input and returns the set of relevant features $\mathcal{I}^c$ of $\mathcal{I}$. Consequently, it reduces image size by considering only useful information, i.e. if the size of $\mathcal{I}$ is $n \times m$, and the size of FE is $p$, then the size of the result is $(n-p) \times (m-p)$.

We could follow the process described in Sections 2.2 and 2.3 to achieve what we want, but the fact that this process is slow and it need computational power, we did not adopt it. However, due the fact we do not have a labels set for our images, we decide to use that process to get few labels, and train a machine learning approach using features extracted to partition the feature space and consequently determine the shrubs.

The goal of image segmentation is to classify each pixel in the image to either light or not. Thus, this problem can be converted into a binary classification problem where small intensity patches are extracted from each image and used to predict the label of each pixel in the patch.

As stated above, our objective is to discriminate the region of shrubs from anything else. The result of the segmentation is an image of the same dimension as the input image, where each pixel of the resulting image light when it belong to a shrub and dark otherwise.

# 3. Deep Neural Network

Deep learning is a subfield of machine learning that learns from different data type images, texts or sounds by emulating the workings of the human brain. Traditional neural network usually consist of two or three hidden layers, where deep neural network consist of more than that. That is why we refer to them as deep networks. In this chapter we will mention machine learning tasks, and briefly describe neural network, then we will go through one of the wide applied tasks to images which is convolutional neural network.

## 3.1 Machine learning

Machine Learning (ML) is the use of algorithms to extract information from data. This information is used to build a model, which is used as well to infer things about future data that we can get. ML algorithms are often categorized into different tasks.

**Supervised Learning** In this task, usually we have large amount of data, inputs and the corresponding outputs, and we feed that to the algorithm and leave it to learn. When the output is a continues value it refer to regression, and when we predict discrete values or classes it refer to classification.

**Unsupervised Learning** This type of tasks finds structure out of the dataset by clustering data. So, it creates an internal representation of the input that is useful for subsequent supervised learning. Things in one cluster must be close to each other, and different cluster must be faraway.

**Semi-supervised Learning** This task is halfway between supervised and unsupervised learning. As what we faced in this work, labelled data is often costly to generate, where the unlabelled data is generally not. So, we use both data to train, and the goal is to build a model that predicts future test data better than the model learned from the labelled training data alone.

**Reinforcement Learning** Usually this learning method interacts with its environment by allowing machines to determine the ideal behaviour to improve its performance, if so a simple reward is given to encourage it to continue, otherwise we punish it by taking a small tribute in order to maximize its performance.

## 3.2 Traditional Neural Networks

Neural networks are one type of machine learning techniques they were first introduced at least $50$ years ago (Wikipedia, b). The ANN are inspired by the biological neural networks of animal brain. The connections between neurons are also modelled on biological brains, as is the way these connections develop over time with training (see Figure 3.1). In the following we will show how these models work, and we will take a look on the mathematical aspect behind.

**3.2.1 Network Architectures.** A typical feed forward neural network consist of an input layer the size of which depends on the input, one or more hidden layer where the number of neurons per layer is a tunable hyper parameter, and an output layer the size of which depends on the type of problem we are dealing with classification or regression (see Figure 3.2).

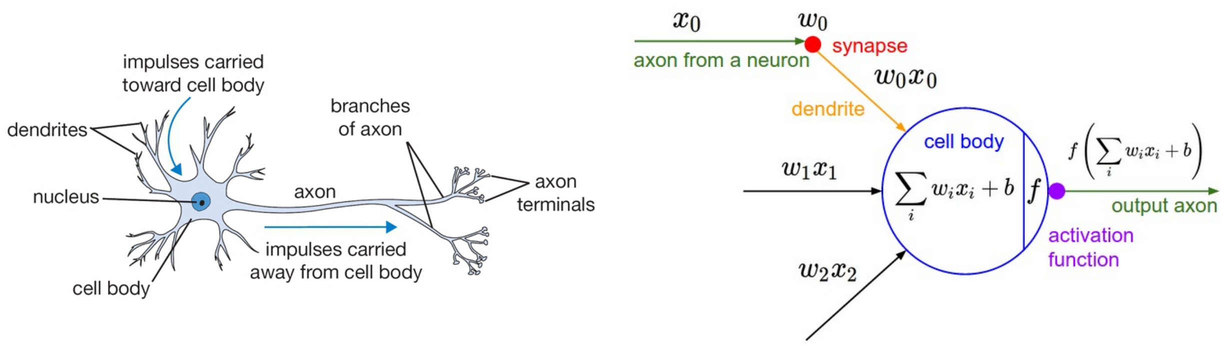

Figure 3.1: The neurons. Left image shows the Biological neuron structure and the one in right shows the artificial neural network mathematical model (http://cs231n.github.io/neural-networks-1)

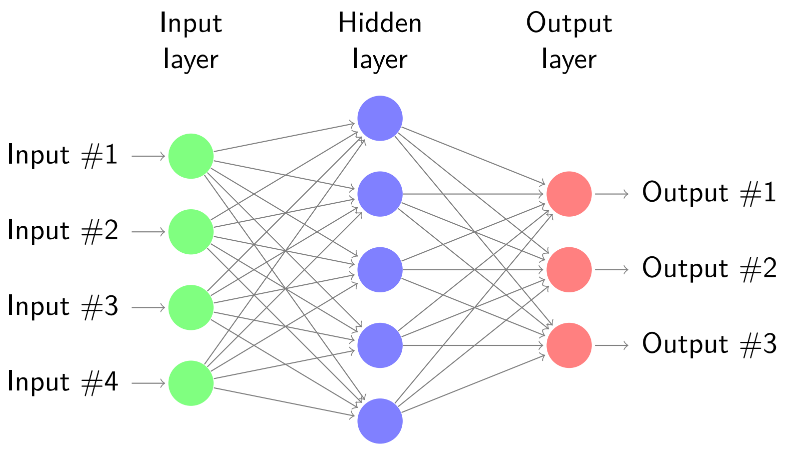

Figure 3.2: Network architectures

**3.2.2 What to learn?.** When we say a NN should learn we mean that it should determine the weight of edges between layers and biases for every layer. Since NN are parametric models, the problem of modelling the given data is reduced to just determining the fixed set of network parameter that is the network weight and biases.

**3.2.3 How to learn?.** Such values are determined by an estimation and we consider the maximum likelihood estimation (MLE) a core ML concept. MLE is one of the most fundamental estimations that is drives basic algorithm like logistic regression.

Consider $\mathcal{S} = \{(x_n, y_n)\}_{n=1}^{N}$ be set of $N$ samples $x_n$ with corresponding labels $y_n$ used to train NN. If $X = \{x_1, \cdots, x_N\}$ represents all our inputs and $Y = \{y_1, \cdots, y_N\}$ all our labels, then the MLE determine the parameters which maximizing the conditional probability of seeing data is

$$\hat{\theta}_{MLE} = arg \max_{\theta} P(Y|X;\theta)\,, \tag{3.2.1}$$

where $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ being the set of NN parameter weight and biases, and $\hat{\theta}_{MLE} = \{\hat{W}^{(l)}, \hat{b}^{(l)}\}_{l=1}^{L}$ the MLE of these network parameters.

Assuming the samples are independent and identically distributed (iid), we can multiply the probability for the $N$ samples $y_n \sim P(Y|X;\theta)$.

$$\hat{\theta}_{MLE} = arg \max_{\theta} \prod_{n=1}^{N} P(y_n|x_n;\theta)\,. \tag{3.2.2}$$

Maximizing the likelihood is the same as maximizing the log-likelihood, which is much easier to compute.

$$\hat{\theta}_{MLE} = arg \max_{\theta} \sum_{n=1}^{N} \log P(y_n|x_n;\theta)\,. \tag{3.2.3}$$

We can evaluate this function in two ways:

- When are the NN used for classification.
- When are the NN used for regression.

In the classification setting the labels can be modelled as Bernoulli variables as they can only take a finite set of specific values. In our case we are interested in two classes. Let us assume that

$$\begin{aligned} P(y_n = 1|x_n;\theta) &= h_\theta(x)\,, \\ P(y_n = 0|x_n;\theta) &= 1 - h_\theta(x)\,, \end{aligned}$$

which can be written in the following way that makes things much easier

$$P(y_n|x_n;\theta) = (h_\theta(x_n))^{y_n}(1 - h_\theta(x_n))^{1-y_n}\,,$$

where $h_\theta(x)$ can be consider as a sigmoid function (3.2.5) of $\theta^T x$.

Regardless of the value of $N$, the number of values each sample can take is fixed and is equal to the number of classes.

$$\begin{aligned} \hat{\theta}_{MLE} &= arg \max_{\theta} \sum_{n=1}^{N} \log(h_\theta(x_n)^{y_n}(1 - h_\theta(x))^{1-y_n}) \\ &= arg \max_{\theta} \sum_{n=1}^{N} y_n \log(h_\theta(x_n)) + (1 - y_n)\log(1 - h_\theta(x))\,. \end{aligned}$$

The negative log-likelihood is the error we wish to minimize. For a single sample $x_n$ the loss function is given by:

$$E_n = -\left(y_n \log(h_\theta(x_n)) + (1 - y_n)\log(1 - h_\theta(x))\right)\,. \tag{3.2.4}$$

Which kind of look like cross-entropy loss, and that what's motivate us for using binary cross entropy.

Similar reasoning for regression, we assume the Gaussian distribution with the mean being the estimated label $\hat{y}$ and variance being some $\sigma^2$.

The MLE estimator is consistent (Wikipedia, a). Thus giving infinite amount of training data

$$\hat{\theta}_{MLE} \xrightarrow[N\to\infty]{\mathcal{P}} \theta^*\,,$$

where $\theta^*$ is the true value.

In other words, the estimated values of the parameters will take their optimal values when we have large data. This is way in ML we claim data is every thing, with more data the chances of fitting a good model are much higher.

**3.2.4 Transfer Function (Activation).** The importance of introducing the non-linear term is to avoid the whole model collapse into a large linear regressor model. Because of the associativity of the convolution only nonlinear activation functions are used between subsequent convolutional layers. Otherwise there will not be any learning if we just use linear activations functions.

Let us consider $B_1$ and $B_2$ be two subsequent convolutional filters applied to an image $\mathcal{I}$ without a non linear activation between. Because of the associativity property of convolution these two layers are effective as just a single layer.

$$B_1 * (B_2 * \mathcal{I}) = (B_1 * B_2) * \mathcal{I} = B * \mathcal{I} \text{ for some convolution } B$$

**The Logistic Function (Sigmoid)** Based on my understanding, early networks used this to squish the relevant weighted sum into that interval between 0 and 1, motivated by this biological analogy of neurons either being active or inactive. The mathematical formula is giving by:

$$sigm(x) = \frac{1}{1+e^{-x}} \tag{3.2.5}$$

Before moving on, here is a useful property of the derivative of the sigmoid function, which we write as $sigm'$:

$$\begin{aligned} sigm'(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} \\ &= \frac{1}{(1+e^{-x})^2} \cdot e^{-x} \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\ &= sigm(x) \cdot (1 - sigm(x)) \end{aligned}$$

**The Rectified Linear Units (ReLU)** ReLU seems to be much easier to train, motivated partially by a biological analogy with how neurons would either be activated or not. So, if it passes a certain threshold it would be the identity function, but if did not then it would just not be activated, so be zero. So, it is kind of a simplifications. Sigmoid didn't help training, or it was very difficult to train at some point, and people just tried ReLU and it happened to work very well for these incredibly DNN. mathematically ReLU is given by

$$Relu(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0\,, \end{cases} \tag{3.2.6}$$

or

$$Relu(x) = \max(0, x)\,. \tag{3.2.7}$$

**3.2.5 Cost Function.** In the grand scheme of things we do not know what that individual weight means, and to be honest we might not even care that much. But what we really care about is how well does it predicts the output values, how accurate is that, base on different inputs. That is why we introduced the cost function, it is take as input the weights and biases and split out a single number describing how bad those weights and biases are. The way it is defined depends on the network behaviour over all the ten of thousand of pieces of training data. The cost function is the average loss incurred by each sample, and it is given by

$$E = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n \log(h_\theta(x_n)) + (1-y_n)\log(1-h_\theta(x))\right). \tag{3.2.8}$$

**3.2.6 Update Weights and Biases.** We now need to find the value of $\theta$ that minimize this cost, and this where NN training takes place. We will use Stochastic gradient descent (SGD) for training, this involve minimizing the cost every time we look at a random sample by using gradient of the cost with respect to weight and biases.

$$\begin{cases} W^{(t+1)} & = W^t - \eta \frac{\partial E}{\partial W} \\ b^{(t+1)} & = b^t - \eta \frac{\partial E}{\partial b}. \end{cases} \tag{3.2.9}$$

with the learning rate $\eta$ chosen adequately.

Here we need to find $\frac{\partial E}{\partial W}$ and $\frac{\partial E}{\partial b}$. To determine these gradients and to update weights and biases, involves two step process.

- Forward Propagation (FP)
- Backward Propagation (BP)

And for that we introduce the following notation.

- $x_n \in \mathbb{R}^d$ $d$-dimensional input.
- $w_{ji}^{(l)}$ is the weight on the edge connecting the $i^{th}$ neuron in layer $(l-1)$ and the $j^{th}$ neuron in layer $l$. $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ weight matrix between layer $l$ and $l-1$.
- $b_i^{(l)}$ is the weight connecting bias neuron with the $i^{th}$ neuron in the layer $l$.
- $n_l$ number of neurons in layer $l$.
- $f^l$ the non linear activation function in layer $l$.
- $a_i^{(l-1)}$ the output activation of layer $(l-1)$.
- We consider $u_j^l = w_{ji}^l \cdot a_i^{(l-1)} + b_j^l$.
- $a_j^l = f^l(u_j^l)$.
- $a^{(0)} = x$ the input to the network.
- $a^L = \hat{y}$ the activation of the last layer equal to the prediction.

**3.2.7 Forward Propagation.** We start by a random initialize of the weight $w_{ji}^{(l)}$ and biases $b_i^{(l)}$ for every layer, then we pick a sample $(x_n, y_n)$ and compute $u^{(1)}, a^{(1)}, \cdots, u^{(L)}, a^{(L)}$ in this order using the following two formulae alternately.

$$u_j^l = w_{ji}^l \cdot a_i^{(l-1)} + b_j^l$$
$$a_j^l = f^l(u_j^l)\,.$$

Hence, the values propagate forward in the network and we compute the prediction $\hat{y}$. We talked before about the cost function for one single neuron in Equation (3.2.8), now we need to generalize since we have too many output neurons. Thus $\hat{y}$ along with the true label $y_n$ is used to compute the loss $E_n$ for this sample, in all the output neurons $n_L$ and represent

$$E_n = -\sum_{j=1}^{n_L} \left(y_j(x_n)\log(h_{\theta j}(x_n)) + (1-y_j(x_n))\log(1-h_{\theta j}(x_n))\right),$$

where $h_{\theta j}(x_n) = a_j^L$.

We consider the total cost as the average loss incurred by each sample, and this is a reasonable assumption.

$$E = \frac{1}{N}\sum_n E_n \tag{3.2.10}$$

Now that we have the cost we need to compute the gradient of the cost with respect to the weights and biases, this gradient computation is done with BP.

**3.2.8 Backward Propagation.** BP is just chain rule, just like how FP gets its name from the fact that the values of $u_j^l$ and $a_j^l$ propagate in the forward direction in the network, the BP gets its name from the fact that the gradient with respect to $u_j^l$ and $a_j^l$ propagate in the inverse direction. So we determined the cost gradient with respect to $u_j^l$ and $a_j^l$ through the network and this is used to determine the cost gradients with respect to the weights and biases. We use $\frac{\partial E}{\partial a}$, $\frac{\partial E}{\partial u_j^l}$ to determine $\frac{\partial E}{\partial w}$, $\frac{\partial E}{\partial b}$.

Starting with the last layer $L$ the cost gradient with respect to the activation $\frac{\partial E}{\partial a_k^L}$ can be simplified using simple differentiation.

$$\begin{aligned}\frac{\partial E}{\partial a_k^{(L)}} = \frac{\partial E}{\partial \hat{y}_k} &= -\frac{1}{N}\sum_n \left(\frac{y_k}{\hat{y}_k} + \frac{1-y_k}{1-\hat{y}_k}\right)\\ &= -\frac{1}{N}\sum_n \left(\frac{y_k}{a_k^{(L)}} + \frac{1-y_k}{1-a_k^{(L)}}\right)\end{aligned}$$

Next we compute the cost derivative with respect to $u_k^L$ for all neuron in the last layer $\frac{\partial E}{\partial u_k^{(L)}}$, let us set this gradient to be $\delta_k^{(l)}$.

$$\delta_k^{(l)} = \frac{\partial E}{\partial u_k^{(l)}}$$

To compute those $\delta_k^{(L)}$ for the last layer we apply chain rule with the activation $a_k^L$ on the same neuron.

$$\begin{aligned}\delta_k^{(L)} = \frac{\partial E}{\partial u_k^{(L)}} &= \frac{\partial E}{\partial a_k^{(L)}} \cdot \frac{\partial a_k^{(L)}}{\partial u_k^{(L)}}\\ &= \frac{\partial E}{\partial a_k^{(L)}} \times f'^{(L)}(u_k^{(L)})\end{aligned}$$

Since we have both term the $\delta_k^{(L)}$ can be computed.

We have computed the cost as well as the gradients with respect to $a_k^L$ and $u_k^L$ for the last layer neurons. Now let's compute the value of the gradient with respect to activation $a_k^{L-1}$ for $(L-1)$ layer neurons. Using a chain rule with $u_k^L$ so:

$$\frac{\partial E}{\partial a_j^{(L-1)}} = \sum_k \frac{\partial E}{\partial u_k^{(L)}} \cdot \frac{\partial u_k^L}{\partial a_j^{(L-1)}} \tag{3.2.11}$$

The term on the left hand side represent the change in the cost with respect to the change in the activation of the $j$ neuron in the layer $(L-1)$. Change in this neuron activation affects all neurons in the next layer, so when the gradient propagate backwards neurons in layer $L$ converge on the $j^{th}$ neuron in layer $(L-1)$ as in the Figure 3.3.

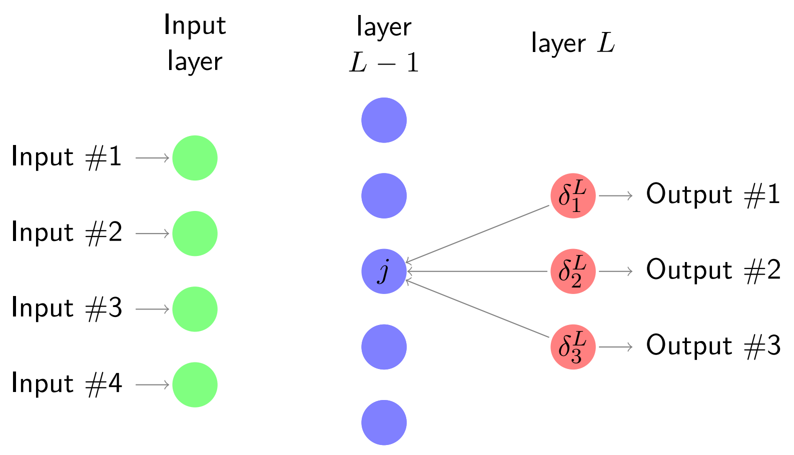

Figure 3.3: The neurons in layer $L$ converge to the neuron $j$ in layer $L-1$

The overall effect of the gradients is the sum of products of the delta and the coresponding weight on the edge.

$$\frac{\partial E}{\partial a_j^{(L-1)}} = \sum_k \delta_k^L \cdot w_{kj}^L \tag{3.2.12}$$

Generalizing this for any layer $l$ is just

$$\frac{\partial E}{\partial a_j^l} = \sum_k \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \tag{3.2.13}$$

Similarly, we express the cost gradient with respect to $u_j^l$ for the $j^{th}$ neuron using the gradient with respect to $a_j^l$. First we start with the chain rule

$$\frac{\partial E}{\partial u_j^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial u_j^l} = \frac{\partial E}{\partial a_j^l} \times f'^{(l)}(u_j^{(l)}) \tag{3.2.14}$$

$f'^{(l)}(u_j^{(l)})$ the derivative of the activation function with respect to $u_j^l$.
Substituting, we get

$$\frac{\partial E}{\partial u_j^l} = \sum_k \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \times f'^{(l)}(u_j^{(l)}) \tag{3.2.15}$$

The left side correspond to $\delta_j^l$.

$$\delta_j^l = \sum_k \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \times f'^{(l)}(u_j^{(l)}) \tag{3.2.16}$$

Next we find $\frac{\partial E}{\partial w_{ji}^{(l)}}$ and $\frac{\partial E}{\partial b_j^{(l)}}$, the derivative of the cost with respect to $w_{ji}$ represent how much this edge affects the cost. We apply chain rule

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial u_j^{(l)}} \cdot \frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{3.2.17}$$

$$= \delta_j^{(l)} \cdot a_j^{(l-1)} \tag{3.2.18}$$

$$\frac{\partial E}{\partial b_i^{(l)}} = \frac{\partial E}{\partial u_j^{(l)}} \cdot \frac{\partial u_j^{(l)}}{\partial b_i^{(l)}} \tag{3.2.19}$$

$$= \delta_j^{(l)} \tag{3.2.20}$$

Once we determine all the weights and biases in the network, perform a SGD update and get the new weight for the network. We repeat for every sample in the training set.

**3.2.9 Mini batch SGD.** In case of batch update we consider the mean of the gradient for all $m$ samples in a batch and substitute this value in the weight update.

$$\frac{\partial E}{\partial w_{ji}^{(l)(t)}} = \frac{1}{m} \sum_m \frac{\partial E_m}{\partial w_{ji}^{(l)(t)}} \tag{3.2.21}$$

The mini batch weights and biases update formulae are:

$$\begin{cases} W_{ji}^{(l)(t+1)} & = W_{ji}^{t} - \frac{\eta}{m} \sum_m \frac{\partial E_m}{\partial W_{ji}^{(l)(t)}} \\ b_j^{(l)(t+1)} & = b_j^{(l)(t)} - \frac{\eta}{m} \sum_m \frac{\partial E_m}{\partial b_j^{(l)(t)}} \end{cases} \tag{3.2.22}$$

Since we have a big, redundant data sets according to (Hinton, 2018b) is better to use mini batches with:

- Gradient descent with momentum.
- Root Mean Square propagation (RMSprop).

**3.2.10 Momentum.** Momentum is a small addition to SGD, but the result are very impressive. With SGD we take small noisier step toward our objective. This method push and accelerate SGD in the relevant direction and dampens oscillations. The momentum update is given by

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta E \\ \theta &= \theta - v_t \end{aligned} \tag{3.2.23}$$

where $\gamma < 1$ is friction coefficient which prevent our algorithm from speeding up without limit. $v_t$ is the exponentially weighted moving averages which represent the velocity.

So instead of SGD just taking every single step independently of all previous steps, it will gain momentum.

As stated before, SGD take smaller noisier step towards our objective, and we one to make it smaller and smaller as we train. One beneficial way to do that is to apply exponentiation decay to the learning rate.

#### 3.2.11 RMSprop.

**Rprop** rprop is the combination of the idea of adapting the step size separately for each weight with the idea of only using the sign of the gradient. Which is equivalent to using the gradient but also dividing by the size of the gradient.

So to decide how much to change the weight, we do not look at the magnitude of the gradient we just look at the sign, but we do look at the step size that we decide for that weight and the step size adapt over time. We increase the step size for a weight multiplicatively, if the sign of the last two gradient are the same, otherwise we decrease multiplicatively.

**RMSprop** Rprop does not work with mini batch (Hinton, 2018b), many people tried but it is hard to get it to work. The reason is it violates the central idea behind SGD, which is when we have small learning rate the gradient gets effectively averaged over successive mini batch.

That lead us to a method in which we combined the robustness of rprop, the efficiency of mini batches, and the effective averaging of the gradient over mini batches which called RMSprop (Hinton, 2018b).

Here we keep a moving average $m_t$ of the squared gradient for each weight

$$m_t = \beta \cdot m_{t-1} + (1-\beta) \cdot (\nabla_\theta E)^2 \tag{3.2.24}$$

where $t$ is indicator for parameter update, increment by one each time we update.

The RMSprop update is given by

$$\theta = \theta - \eta \frac{\nabla_\theta E}{\sqrt{m_t}} \tag{3.2.25}$$

## 3.3 Convolutional Neural Network (CNN)

Convolutional neural network (CNN or convNet), it is an ANN that is so far been most popularly used for analysing image. CNNs combine both DNN and kernel convolutions which we talked about in Chapter 2. Here instead of what we did before which was run a Gaussian blur kernel and then machine learn on that, we just give the CNN the opportunity to learn which features are interesting (could be edge detection, corner detection, maybe something highlight what ever in the middle of the image.)

So, if CNN is just some form of ANN what differentiates it from just a standard multilayer perceptron (MLP). First, CNNs has 3 fundamental features that reduce the number of parameters.

**Sparse interactions between layers** CNN can reduce the number of parameters through indirect interactions. So, if we run deep, we do not need every neuron to be connected to every other to carry information throughout the network, in other word, sparse interaction between layer should suffices.

**Parameters sharing** This further touch on the reduced parameter as did sparse interaction before. It is important to understand that CNN create spatial features,[1] and the individual points in the same depth of the feature map are created from the same kernel in other word they are created using the same set of shared parameters and this drastically reduces the number of parameters to learn from typical ANN.

[1]In reality most or some of the features especially that are in the deeper layers of CNNs are not interpretable by humans

**Equivariant representation** A function $f$ say to be equivariant with respect to another function $g$ if:

$$f(g(x)) = g(f(x)), \text{ for an input } x$$

Convolution is equivalent with respect to translation, this means convolve an image $\mathcal{I}$ then translate it is the same as translate $\mathcal{I}$ and the convolve it. Equivariant of convolutions with respect to translation is used in sharing parameters across the data. In image data for example, similar edges may occur throughout the image, so it makes sense to represent them with the same parameters.

Also, CNN has hidden layer call convolutional layers. Convolution layers are just like any other layers they receive inputs then transforms the input in some way and then output the transformed inputs to the next layer, with a CNN this transformations are a convolution operations.

**3.3.1 Convolution operations.** Note that convolutions are not only applicable to images, we can also convolve 1-D time series data. Mathematically, if input $f$ and kernel $g$ are both functions of one dimensional data like time series data then convolution is given by:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t - \tau) d\tau \tag{3.3.1}$$

The equation (3.3.1) represents the percentage of area of the filter $g$ that overlaps the input $f$ at time $\tau$ over all time $t$. Since $\tau < 0$ is meaningless, and $\tau > t$ represent the value of the function in the future, which we do not know, we can apply tighter bounds to the integral which becomes:

$$(f * g)(t) = \int_{0}^{t} f(\tau) \cdot g(t - \tau) d\tau \tag{3.3.2}$$

The equation (3.3.2) corresponds to a single entry in the 1-D convolution tensor, that is the $t^{th}$ entry. To compute the complete tensor we need to iterate $t$ over all possible values. In this work, we have multi dimensional input tensor that require multi dimensional kernel, in this case we consider an image input $\mathcal{I}$ and a kernel $H$, whose convolution is defined as follow:

$$(\mathcal{I} * H)(x, y) = \int_{0}^{x} \int_{0}^{y} \mathcal{I}(i, j) \cdot H(x - i, y - j) di\, dj \tag{3.3.3}$$

$$= \int_{0}^{x} \int_{0}^{y} \mathcal{I}(x - i, y - j) \cdot H(i, j) di\, dj \tag{3.3.4}$$

The equation (3.3.3) perform convolution by sliding the image over the kernel. while equation (3.3.4) does it the other way around, it is slide the kernel over the image. Since there are usually less possible values for $x$ and $y$ in the kernel than the image we use the second equation (3.3.4) for convolution. The discrete version for a kernel $H$ of size $h_1 \times h_2$ is

$$(\mathcal{I} * H)(x, y) = \sum_{i=0}^{h_1 - 1} \sum_{j=0}^{h_2 - 1} \mathcal{I}(x - i, y - j) \cdot H(i, j) \tag{3.3.5}$$

The result will be a scaler value, and we repeat the process for every point $x$ and $y$ for which the convolution exist on the image. This values are stored in a convolved matrix represented by $(\mathcal{I} * H)$. This output constitutes a part of a feature map. For the shape of the output we have the following relations

$$Out_{width} = \left\lfloor \frac{I_{width} - h_{width}}{stride} \right\rfloor + 1$$

$$Out_{height} = \left\lfloor \frac{I_{height} - h_{height}}{stride} \right\rfloor + 1$$

$$Out_{depth} = I_{depth}$$

Now, The basis of a CNN are Convolution layers, but CNNs can and usually do have other non convolutional layers as well.

**3.3.2 Pooling layer.** Involves a down sampling of features so that we need to learn less parameters during training. Typically there are two hyper parameter introduced with the pooling layer.

**Dimension of spatial extent** The value of $n$ for which we can take an $n \times n$ feature representation and map it to a single value.

**The stride** How many features the sliding window skips along the width and height which is similar to that we saw in convolution

In practice, we use pooling layer with a $2\times 2$ $\max$ filter and stride of $2$. This non overlapping filter halves the feature map. A $\max$ filter return the maximum value among the features in the region. Average filter which return the average of features in the region can also be used but $\max$ pooling work better in practice. Performing pooling reduces the chances of over fitting as there are less parameters to learn.

Applying a standard $\max$ pooling will reduce the number of features to $25\%$ of the original number, this is a significant decrease in the number of parameters. For the shape of the output without padding for image border, we have the following relations

$$\begin{aligned} Out_{width} &= \left\lfloor \frac{I_{width} - h_{width}}{stride} \right\rfloor + 1 \\ Out_{height} &= \left\lfloor \frac{I_{height} - h_{height}}{stride} \right\rfloor + 1 \\ Out_{depth} &= I_{depth} \end{aligned}$$

**3.3.3 Drop-out.** CNNs have a habit of over fitting, even with pooling layers. One way to avoid that is Drop out layer. This layer randomly sets units to zero. Therefore neurons in this layer will depend on themselves, and avoid relying on the rest of neurons. thereby enabling them to develop meaningful features, independent of others. When using drop out we can think of it as training different architecture paralleled, thus more accurate results. Drop out should be used such as between fully connected layers and perhaps after pooling layers (Srivastava et al., 2014).

**3.3.4 Batch normalization.** (Ioffe and Szegedy, 2015) This layer acts as a regularizer, and helps to avoid over fitting. It reduce the need of drop out and sometimes even eliminate this need. Usually we add it after convolutional layers in order to normalize the values of our parameters for every mini batch, and this is done by setting them to zero mean and unit variance.

**3.3.5 Fully connected layer.** One could ask a question what exactly is the use of fully connected layer? The CNN exploit the already existing correlation between local pixels, and give us a high level features representation of the input image. Adding a fully connected layer help learn a possibly non linear combination of these features.

The output of the last convolution layer is a 3-D volume, and is usually very deep due to the increased number of kernels that are introduced at every convolutional layer. We says every convolutional layer because convolution, activation and pooling layers can occur at many times before the fully connected layer and hence is the reason for the increased depth.

**3.3.6 Flatten layer.** To convert this 3-D volume into 1-D we want the output width and height to be 1. This is done by flattening the 3-D layer into one dimensional vector, which is in an admissible form for the fully connected layer.

# 4. Results and Discussion

According to the objectives assigned to this work, we should present the results on image preprocessing, feature extraction and classification. In this chapter, we will look at the data we are working on, and then go through the results.

## 4.1 Data Collection

This work is derived from the work of Miss. Simon, H.R., supervised by Prof. West, A.G., Slingsby, J. entitled *Unpacking a pixel: drivers of seasonality in vegetation greenness (NDVI) in fynbos* (Simon, 2017), she collected the data for her studies using drones. where in order to obtain very-high-resolution ($< 1$ m) true-colour (RGB) and NDVI data (the one used to train our model), aerial imagery acquisition flights were conducted over the study area using a DJI (Dà-Jiāng Innovations Science and Technology Co., Ltd) Phantom III Professional UAV. For RGB imagery (Figure 4.1), the UAV was mounted with a 12 megapixel camera supplied by DJI, and flew an area of $\sim 550 \times 550$ m (at $30$ m altitude, $8$ m/s speed, $80\%$ overlap). For NDVI imagery, the UAV was mounted with a Parrot Sequoia (MicaSense, 2017), a pocket-sized multispectral camera and flew the same area of $\sim 550 \times 550$ m (at $120$ m altitude, $6$ m/s speed, $80\%$ overlap). The Sequoia contained four narrow-band 1.2 megapixel sensors that captured light in the green ($\sim 550$ nm, $40$ Nm bandwidth), red ($\sim 660$ nm, $40$ Nm bandwidth), near-infrared ($\sim 790$ nm, $40$ Nm bandwidth) and red-edge ($\sim 735$ nm, $10$ Nm bandwidth) regions. The camera was fitted with a sunshine sensor (supplied by MicaSense) to correct for changes in light intensity among time periods. To further minimize the effect of solar angle and cloud cover, measurements were taken on clear days within one hour of midday. Additional radiometric calibration imagery was taken using white and grey ($18\%$) reflectance panels, according to standard protocol as outlined by MicaSense (2017). To plan and monitor these flights, the DJI GO and Atlas Flight applications were used with an Apple iPad Mini. To obtain NDVI imagery across seasons, this process was repeated four times, every two months (bi-monthly), in on $28$ February, $24$ April, $06$ June and $28$ August $2017$.

As we mentions above, we managed to get a large data set (more than $16.000$ image around $45.5$ Gb) but due to the lack of computing power to process that much of data, we decide to work on small part of the data set ($500$ image around $2$Gb), we used the very high resolution RGB images ($4000 \times 3000$ pixels), and the equivalent labels set ($500$ binary image around $160$Mb).

## 4.2 Image Processing

For this part of the project we used the open source software OpenCV $3.4.0$ (Itseez, 2015). First, we blurred the image to smooth high intensities pixels, for that we used a filter of size $5 \times 5$ and standard deviation $\sigma = 1$. Then we applied a threshold on the r,g,b (red, green, blue) values, where we consider only the range $(35, 95, 95) < (r, g, b) < (75, 230, 230)$ in the original images the result is shown in Fig 4.1. This threshold values are chosen after running a statistic on the pixel values of several images using Mathematica. Follows by a Gaussian filter of size $5 \times 5$ and standard deviation $\sigma = 1$ to the output image (see Fig 4.2) in order to get rid the noise. Then we binaraized the output by considering the mean of pixels as a threshold (see Fig 4.2). In order to get ride of noisy pixels we applied the morphological operator "opening" with kernel size of $15 \times 15$. As we can see in the output of the opening (Fig 4.3) there is small holes in the region of interest, the dilation operator help us to fill them out (See Fig 4.3). Find contour function was used with a threshold to the area of $15000$ unit to extract only large region and remove the remaining noisy region (see Fig 4.4).

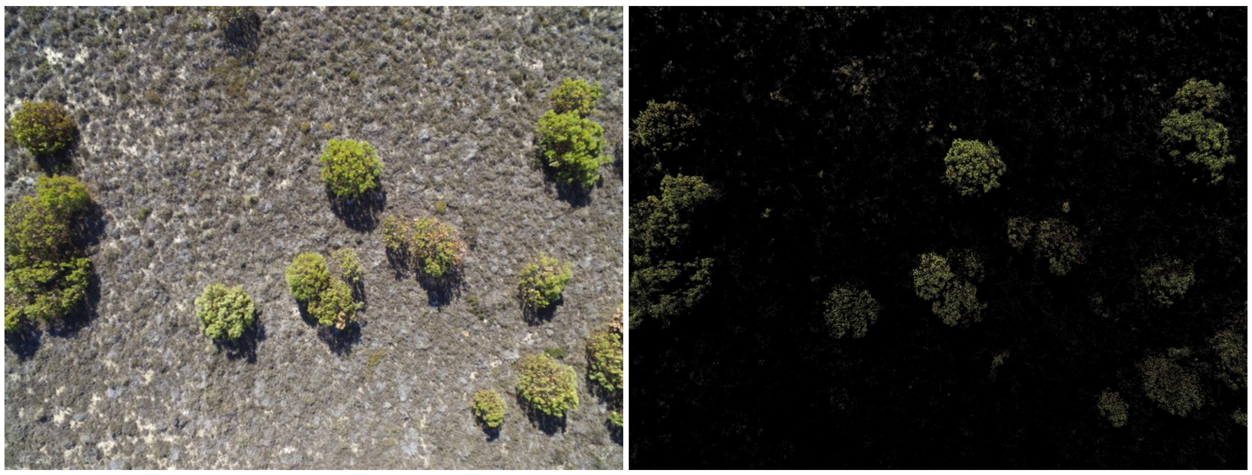

Figure 4.1: The image on the right is the result of applying a threshold to the original image on the left

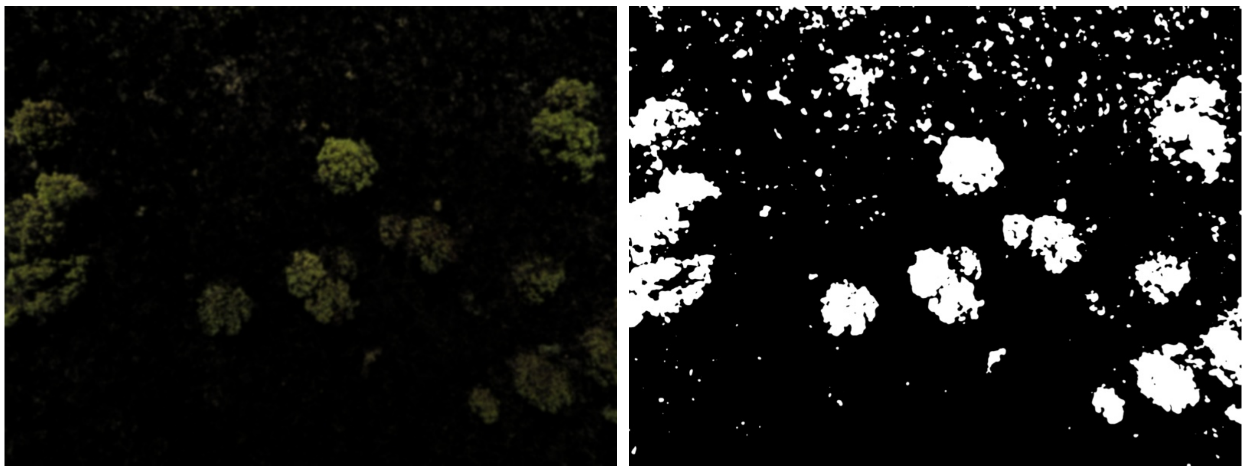

Figure 4.2: The blurred image on the left, and the binary image on the right

Using produced images as labels set, we trained a convolutional neural network model to segment shrubs. Our architecture is topologically based on the convolutional layers in the U-net network (Ronneberger et al., 2015). It consist of two symmetrical networks encoder (see Fig 4.5 and Table 4.1) and decoder (see Fig 4.6 and Table 4.2), as well as a fully connected network (see Fig 4.7 and Table 4.3). The encoder network extract low resolution feature maps from the input image, it consist of $4$ stages, where each stage consists of two convolutional layers each one of these layers followed by a batch normalization layer and an activation layer, then a max pooling layer that halves the resolution of the input. In the other hand, the decoder network uses a deconvolution layer instead of max pooling to map the extracted feature to a full input resolution feature maps. However, this map will only produces coarse segmentation maps due to the information loss when applying max pooling. Therefore, concatenate layers are introduced, they merge higher resolution feature maps with the map of the extracted features. The last part of our architecture consist of one convolutional layer followed by a max pooling layer plus a drop out layer, the fully connected layer with a drop out layer was added after flattening the output

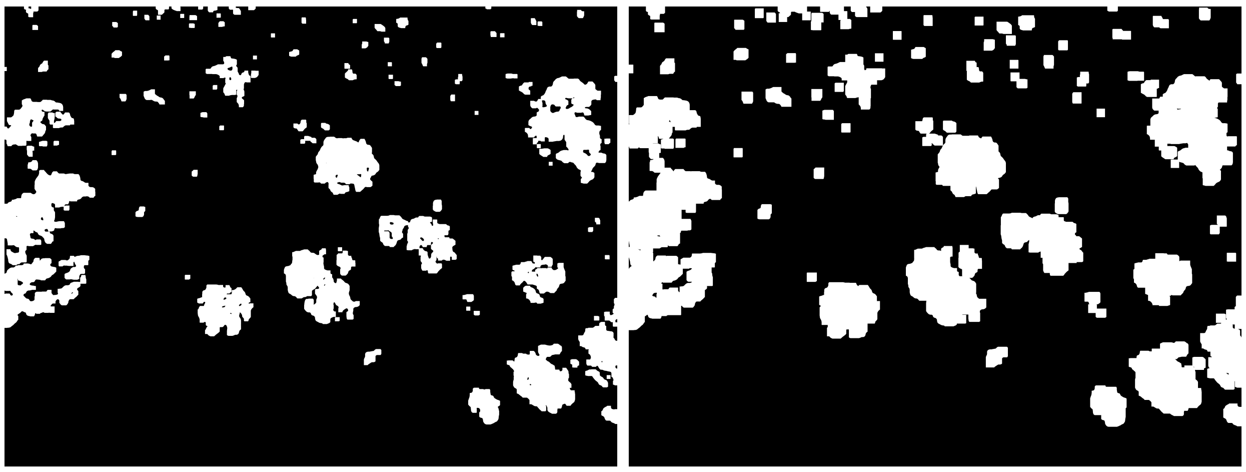
Figure 4.3: The result of applying an opening operator to the binary image is on the right, and the dilation of this result is on the left

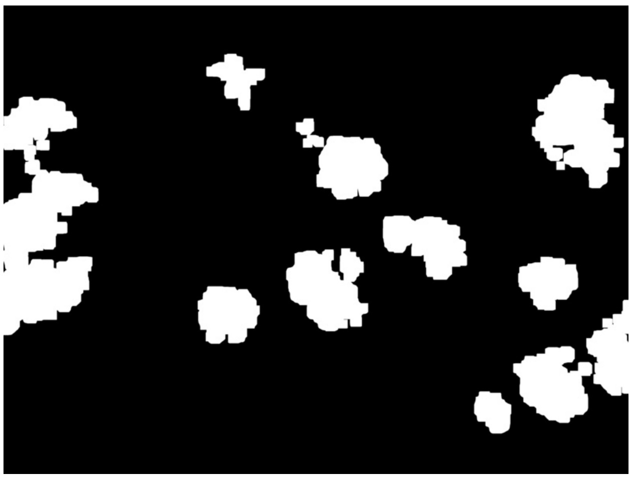
Figure 4.4: The final result after extracting the large contours, the image is used later to train the CNN model

of the max pooling.

The output layer uses a sigmoid function with the number of neurons equal to the number of pixels in the input image.

All experiments were performed using Keras (Chollet et al., 2015), and for automatic differentiation and optimization during training we used Tensorflow backend (Abadi et al., 2015). All that in Python $3.5$. In order for the output to have the same length as the original input we used same-padding in convolutional layers in all architectures. To reduce the number of parameters and speed up training we used drop out after the last convolutional layer and after the fully connected layer as well.

For evaluating the quality of the model predictions and measure classification performance we used the scikit learn Python library (Pedregosa et al., 2011).

The fact that our data consist of high quality images $4000 \times 3000$ pixels, we start by creating batches of size $64 \times 64$ from each image and its label to avoid training for long time. We removed the batches

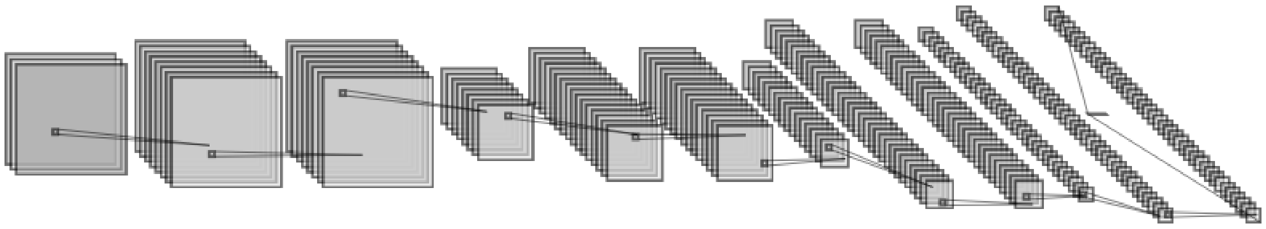

Figure 4.5: The encoding architecture, from left to right: input image; $2\times$(convolutional layer + batch normalization layer + ReLU); max pooling layer; and so on alternatively for $4$ stages

| Layer | Taype | Feature Map | Feature Map size | Kernel Size |
|---|---|---|---|---|
| $0$ | input | $3$ | $64 \times 64$ | |
| $1$ | convolutional+BN+ReLU | $64$ | $64 \times 64$ | $3 \times 3$ |
| $2$ | convolutional+BN+ReLU | $64$ | $64 \times 64$ | $3 \times 3$ |
| $3$ | max pooling | $64$ | $32 \times 32$ | $2 \times 2$ |
| $4$ | convolutional+BN+ReLU | $128$ | $32 \times 32$ | $3 \times 3$ |
| $5$ | convolutional+BN+ReLU | $128$ | $32 \times 32$ | $3 \times 3$ |
| $6$ | max pooling | $128$ | $16 \times 16$ | $2 \times 2$ |
| $7$ | convolutional+BN+ReLU | $256$ | $16 \times 16$ | $3 \times 3$ |
| $8$ | convolutional+BN+ReLU | $256$ | $16 \times 16$ | $3 \times 3$ |
| $9$ | max pooling | $256$ | $8 \times 8$ | $2 \times 2$ |
| $10$ | convolutional+BN+ReLU | $512$ | $8 \times 8$ | $3 \times 3$ |
| $11$ | convolutional+BN+ReLU | $512$ | $8 \times 8$ | $3 \times 3$ |

Table 4.1: The encoding network, BN for batch normalization; ReLU the activation function

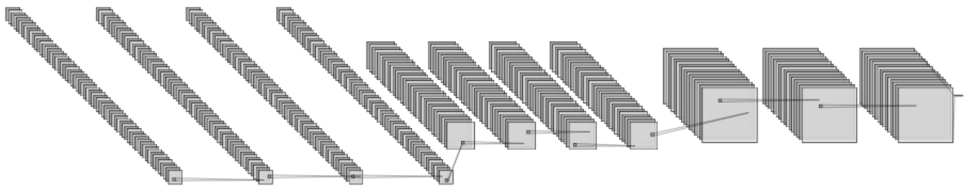

Figure 4.6: The decoding architecture, from left to right: Up sampling the output of the encoder and concatenate it with the convolution in the $3$rd stage in the encoder; $3\times$(convolutional layer + batch normalization layer + ReLU); and so on alternatively

with no informations[1] to balance the data. Then we perform a normalization to the RGB batches such that all pixels values are in the range $[0, 1]$, and a binarization to the labels. The weights were all

[1]The batches where there is no pixel belong to a shrub

| Layer | Type | Feature Maps | Feature Map size | Kernel size |
|---|---|---|---|---|
| $0$ | Up Sampling+concatenate | $256$ | $16 \times 16$ | $2 \times 2$ |
| $1$ | Convolutional+BN+ReLU | $256$ | $16 \times 16$ | $3 \times 3$ |
| $2$ | Convolutional+BN+ReLU | $256$ | $16 \times 16$ | $3 \times 3$ |
| $3$ | Convolutional+BN+ReLU | $256$ | $16 \times 16$ | $3 \times 3$ |
| $4$ | Up Sampling+concatenate | $128$ | $32 \times 32$ | $2 \times 2$ |
| $5$ | Convolutional+BN+ReLU | $128$ | $32 \times 32$ | $3 \times 3$ |
| $6$ | Convolutional+BN+ReLU | $128$ | $32 \times 32$ | $3 \times 3$ |
| $7$ | Convolutional+BN+ReLU | $128$ | $32 \times 32$ | $3 \times 3$ |
| $8$ | Up Sampling+concatenate | $64$ | $64 \times 64$ | $2 \times 2$ |
| $9$ | Convolutional+BN+ReLU | $64$ | $64 \times 64$ | $3 \times 3$ |
| $10$ | Convolutional+BN+ReLU | $64$ | $64 \times 64$ | $3 \times 3$ |
| $11$ | Convolutional+BN+ReLU | $64$ | $64 \times 64$ | $3 \times 3$ |

Table 4.2: The decoding network, each convolutional layer followed by a batch normalization. ReLU activation is used

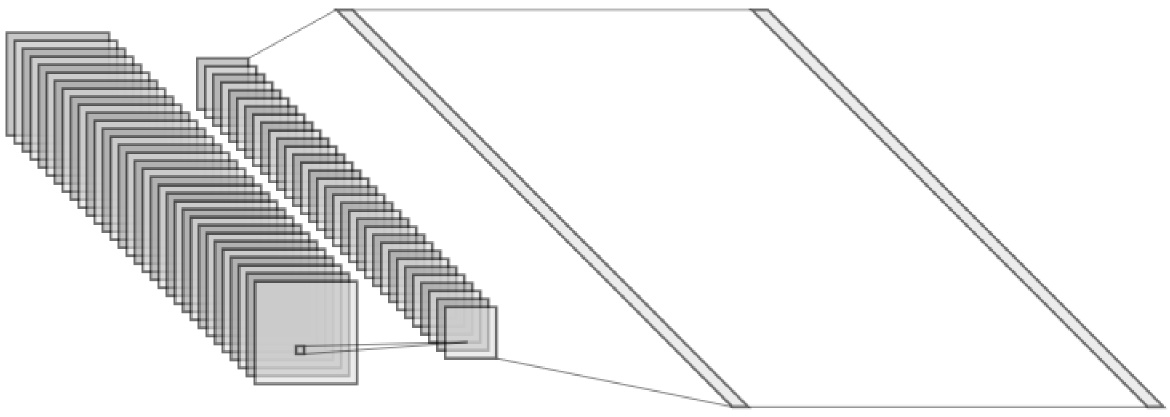

Figure 4.7: The fully connected architecture, from left to right: convolutional layer followed by ReLU activation; max pooling and drop out; 2 fully connected layer

| Layer | Type | Feature Map | Feature Map size | Kernel size |
|---|---|---|---|---|
| $0$ | Convolutional | $64$ | $64 \times 64$ | $1 \times 1$ |
| $1$ | max pooling + drop out | $64$ | $64 \times 64$ | $2 \times 2$ |
| $2$ | fully connected+ drop out | $4096$ Neurones | $1 \times 1$ | |
| $3$ | fully connected | $4096$ Neurones | $1 \times 1$ | |

Table 4.3: The fully connected network, ReLU activation function was used except for the output layer we used Sigmoid

initialized using samples from a uniform distribution described in (Choi et al., 2018). The number of output neurons was $64 \times 64 = 4096$ neuron, and the total number of parameters is $293.596.512$.

Preliminary experiments showed a severe over fitting given the low number of training samples compared to the capacity of the architecture. Numerous attempts and hyper parameter tuning led us to finally settle on using stochastic gradient descent (SGD) with a fixed learning rate of $0.05$ and momentum of $0.9$ to train all the variants for $200$ epoch. Before each epoch, the training set is shuffled and each mini-batch ($64$ images due to memory constraints) is then picked in order thus ensuring that each image is used only once in an epoch. We also use an early-stopping strategy to stop the learning process when the validation loss stops improving in $8$ consecutive epochs. We select the model which performs highest on a validation dataset. We use the binary-cross-entropy loss as the objective function for training the network.

The training was done in the Ace Lab Cluster (Cawood et al., 2018), using $1$-GPU node of the $4$ available, each node has $2$ Nvidia K$40$ GPUs. On average the training time was $4$ hours.

The predicted binary image (see left image in Fig 4.8) of our model is in the same size of the RGB input, where the region of shrubs is highlited. And the right image in Fig 4.8 show us the pixels falsely segmented as foreground.

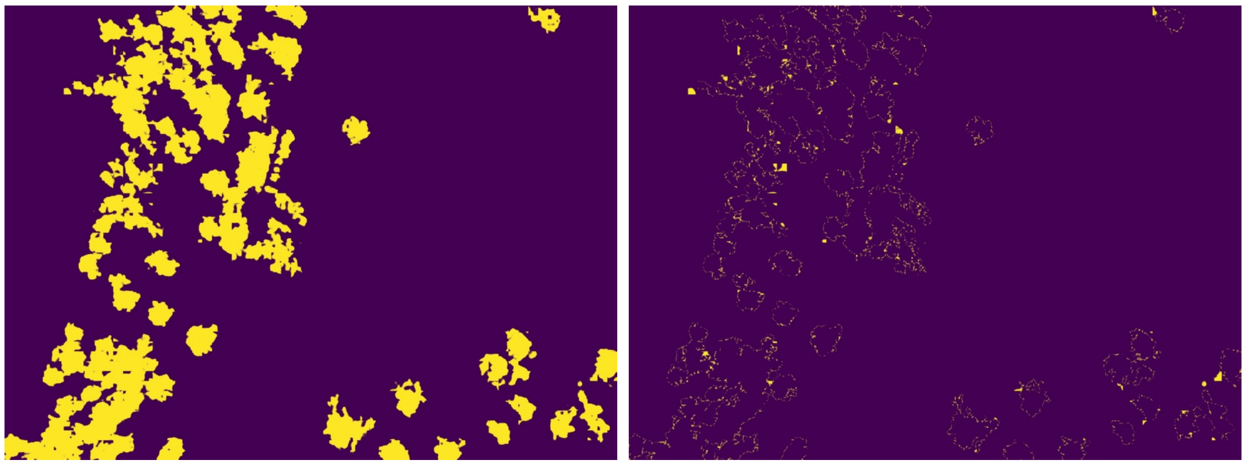
Figure 4.8: The output of the model in the left (predicted image), and the false positive pixels on the right

Figure 4.9 is the result of overlapping the prediction and the false positive over the original image.

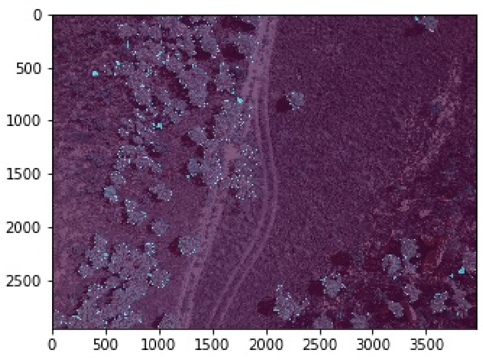

Figure 4.9: The pixels highlighted with the green colour are the falsely predicted pixels

**4.2.1 Accuracy-Loss.** The results in Figure 4.10 shows that the accuracy on the original dataset increases after each epoch and reach $0.93728$ for the training dataset and $0.86390$ on the testing dataset for $200$ epoch. Furthermore, the results in Figure 4.11 shows that the loss decreases after each epoch and reach $0.14683$ for the training dataset in the $200$th epoch, and decrease with some fluctuation and reach $0.34837$ on the testing dataset in the same epoch.

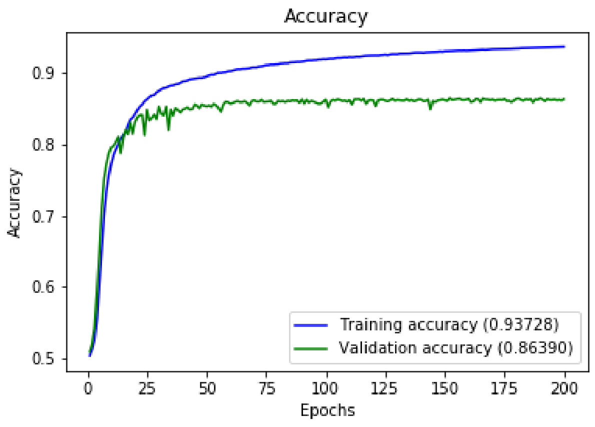

Figure 4.10: The classification accuracies, on the training and validation sets

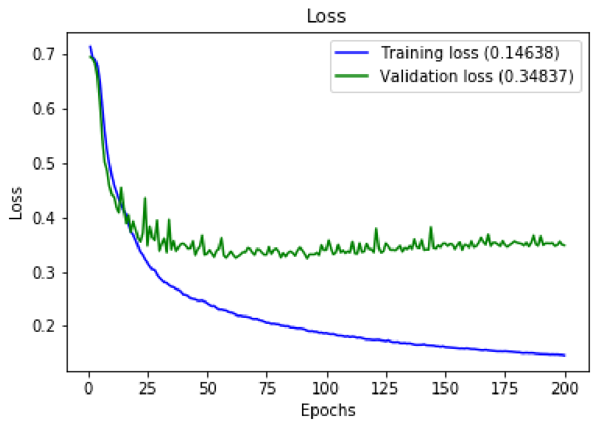

Figure 4.11: The classification losses, on the training and validation sets

Our neural network can still over fit the data. Thus, an early-stopping strategy to stop the learning process when the validation loss stops improving in $8$ consecutive epochs was used in the following model. As well as a reducing learning rate strategy when the validation loss stops decreasing in $4$ consecutive epochs. For the same hyper parameters used before, with these strategies, the model show better performance on the testing set $0.87774$ (see Fig 4.12). And the loss was much better than the previos model $0.28611$ on the testing set.

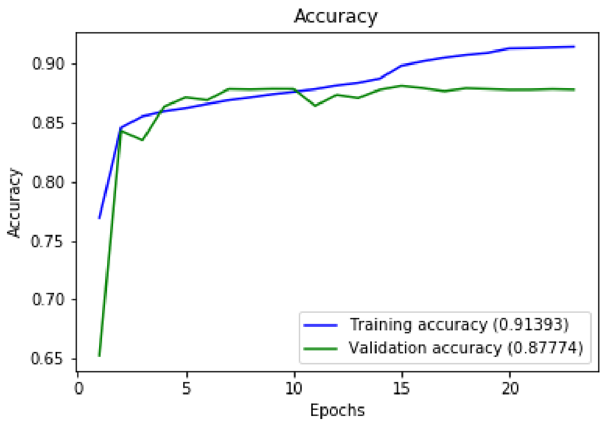

Figure 4.12: The classification accuracies, on the training and validation sets

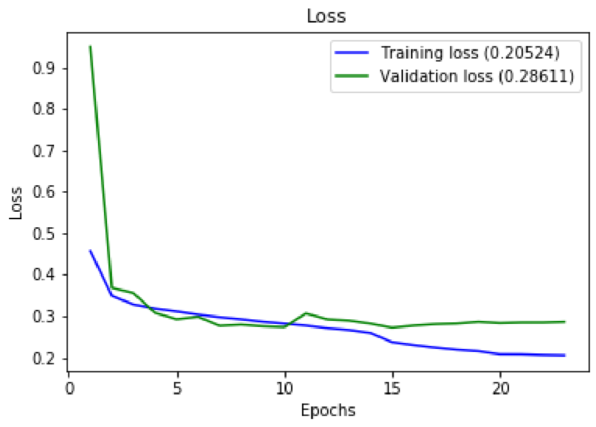

Figure 4.13: The classification losses, on the training and validation sets

**4.2.2 Precision-Recall.** In order to over come the disadvantage of the simple accuracy we introduce two new measure, Precision and Recall.

Precision $(P)$ is given by

$$P = \frac{TP}{TP + FP},$$

where $(TP)$ is the number of true positives, and $(FP)$ the number of false positives. Low $FP$ implies high precision. In the other hand, Recall $(R)$ is given by

$$R = \frac{TP}{TP + FN},$$

where $(FN)$ is the number of false negatives. Low $FN$ implies high recall.

The best model is when $P = R = 1$, but a model with high precision and high recall is a good model, and most of its prediction are correct.

Precision-recall curve is a useful visualization tool to see the performance of our model. The more area under the curve is, the better the model we have.

Figure 4.14 show us the precision recall curve of our model. We get both high precision and high recall, and indeed, most of our prediction are labelled correct. The precision score was $P = 0.9785$, and the recall score was $R = 9698$.

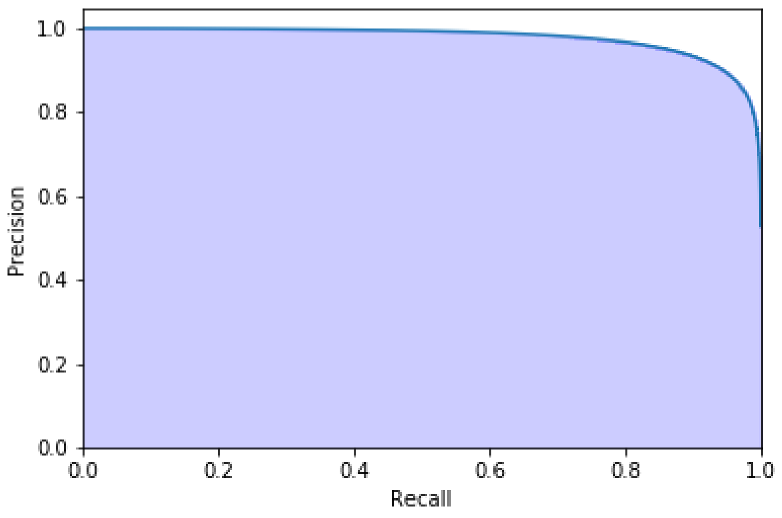

Figure 4.14: Precision recall curve with precision score of $P = 0.9785$, and recall score of $R = 0.9698$ shows a good performance of the model

The $TP$, $TN$, $FP$ and $FN$ actually mean:

- True Positive $(TP)$: When the actual pixel is ON and the predicted pixel is also ON.
- True Negative $(TN)$: When the actual pixel is OFF and the predicted pixel also OFF.
- False Positive $(F_P)$: When the actual pixel is OFF and the predicted pixel is ON.
- False Negative $(FN)$: When the actual pixel is ON and the predicted one is OFF.

Using the above notation we introduce the confusion matrices. In binary classification a confusion matrix is a $2 \times 2$ matrix that compare the actual classes, which can be positive or negative, with it predicted class, which also can be positive or negative. The confusion matrix of our model is given by Figure 4.15, where the one on the right is a normalized confusion matrix. The top left corner stand for $TP$, top right for $FN$, bottom left $FP$ and the bottom right stand for $TN$.

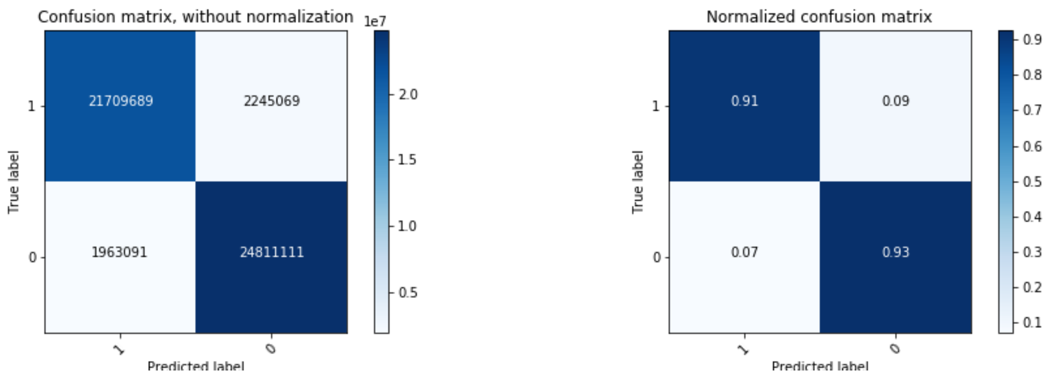

Figure 4.15: Confusion matrix.

Unfortunately, in this work we could not look at that many things due to the short period of time for this essay. In our approach for binarizing the image we did not consider variances in image. Therefore, an optimal approach to find the threshold $\theta$ which considers the pixel variance is given by Otsu (Jassim and Altaani, 2013). This can help improving the labels set.

In deep learning large data set with deep architecture shows increasing success, and good results. However, in this project we could not use the whole data set to train due to the lack of memory and computational time during training and testing. To reduce number of parameters and computational time during training, one can think of different approach to the model.

The challenge is to segment large shrubs, as we can see in Fig 4.8 the model did well, but not as we expected given the low number of training samples compared to the capacity of the architecture. The model miss classify some of the small shrub because they have almost the same feature map as large shrubs. And this similarities made a lot of difficulties when choosing the threshold for the RGB colour Some of the images contain large amount of noise and we could not deal with it when processing so we just remove those images as a part of cleaning the data.

# 5. Conclusion and Future Work

This project discussed different tasks in image segmentation, and their use to generate a labelled data set. Also, this work investigated the use of recent developments in deep learning based image segmentation for an areal vegetation detection problems. We used a deep convolutional neural network architecture (CNN) inspired from U-net architecture for semantic segmentation. Experiments demonstrated that providing a labelled data set, CNNs are much better approach to solve this problem in a real time than traditional image processing methods.

At present, our label set is not that good since the range of the RGB colour that we considered was not that accurate. And we used mean pixels value when binarizing the images for simplicity, also we only focused on the large shrubs due to the short period. Therefore, for future work, we suggest:

1. Better threshold range for the RGB colours, which take into account different species.
2. Instead of considering the mean pixels value, one can consider the pixel variance (Jassim and Altaani, 2013) or more efficient value.
3. Improve the labels set by considering different segmentation task when processing the images. And take into account different species.
4. It is desirable that researchers study how feature extraction functions look in terms of its properties and form. Therefore, they can focus on how to come up with their own functions based on properties they should cover depending on the problem to be treated.
5. This problem can be converted into a classification problem where small intensity patches are extracted from each image and used to predict the label of the pixel at its centre.
6. When data of varying water availability, and data of the characteristics of each species are available, one can build a model to solve the initial problem, and predict the seasonal fluctuation in vegetation greenness.

Python codes are available at https://github.com/BYSD/AIMS-Essay.

End-to-end learning of deep segmentation architectures is a harder challenge and we hope to see more attention paid to this important problem.

# Acknowledgements

First of all, I would like to thank Almighty ALLAH for every thing in my life. Without his guidance I could never be able to finish this work.

To make any project, essential requirement is able guidance and references without which project is incomplete. Thanks to Dr. Jonathan Shock and Prof. Adam West who have provided me an opportunity to gain knowledge through this type of project.

Thanks to Mr. Jordan and Mr. Kayode for their help. A special thanks goes my colleagues Juliana, Lambert, Nkosinathi and Rojo. for the encouragement, the team work and the struggles we had together.

I acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources to this research project.

At last but not the least, I am thankful to all my colleagues, friends and other persons who have directly or indirectly helped me during preparation of this essay.

# References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

BRITZ, D. Understanding convolutional neural networks for nlp, November 7 2015.

Brownlee, J. Crash course in convolutional neural networks for machine learning, November 7 2015.

Cawood, M., Macleod, D., and Tshililo, I. *ACE Lab Cluster*, 2018. Center of High Performance Computing CHPC, Cape Town, South Africa.

Choi, H., Cho, K., and Bengio, Y. Fine-grained attention mechanism for neural machine translation. *Neurocomputing*, 284:171–176, 2018. doi: 10.1016/j.neucom.2018.01.007. URL https://doi.org/10.1016/j.neucom.2018.01.007.

Chollet, F. et al. Keras. https://keras.io, 2015.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hinton, G. Neural networks for machine learning, 2018a. Geoff Hinton's Coursera class.

Hinton, G. Neural networks for machine learning, 2018b. Slide 29 of lecture 6a Overview of mini-batch gradient descent.

Ioffe, S. and Szegedy, C. Batch normalization : Accelerating deep network training by reducing internal covariate shift. arXiv Preprint arXiv:1502.03167v3, 2015.

*The OpenCV Reference Manual*. Itseez, 2.4.9.0 edition, April 2014.

Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

Jassim, F. A. and Altaani, F. H. Hybridization of otsu method and median filter for color image. International Journal of Soft Computing and Engineering, 2:1–6, 2013.

Kofi, A. R. *Revised Mathematical Morphological Concepts: Dilation, Erosion, Opening and Closing*. Phd, Kwame Nkrumah University of Science and Technology, Kumasi, 2014.

Li, S. Markov random field models in computer vision. In: Eklundh JO. (eds) Computer Vision — ECCV '94. ECCV 1994. Lecture Notes in Computer Science, vol 801. Springer, Berlin, Heidelberg, 1994.

Marsh, M. A literature review of image segmentation techniques and matting for the purpose of implementing 'grab-cut', 2014.

Montoya-Zegarra, J. A., Wegner, J. D., Ladický, L., and Schindler, K. Semantic segmentation of aerial images in urban areas with class-specific higher-order cliques. Photogrammetry and Remote Sensing, ETH Zurich and Computer Vision Group, ETH Zurich, 2015.

Ng, A. Supervised learning, 2012. CS229 Lecture notes.

PAL, N. R. and PAL, S. K. A review on image segmentation techniques. Machine Intelligence Unit, Indian Statistical Institute, 1993.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Shotton, J., Johnson, M., and Cipolla, R. Semantic texton forests for image categorization and segmentation. IEEE: 978-1-4244-2243-2/08, 2008.

Simon, H. *Unpacking a pixel: drivers of seasonality in greenness (NDVI) in fynbos*. Msc, University of Cape Town (UCT), Department of Biological Sciences, 2017.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15 1929-1958, 2014.

The University of Uckland. Gaussian filtering. Consistency, https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf, Accessed April 2018.

Varone, M. and al. What is machine learning? a definition, may 2018.

Wikipedia. Maximum likelihood estimation. the free encyclopedia, https://en.wikipedia.org/wiki/Maximum_likelihood_estimation#Properties, Accessed April 2018a.

Wikipedia. Artificial neural network. the free encyclopedia, https://en.wikipedia.org/wiki/Artificial_neural_network, Accessed March 2018b.

Xavier, M. P. *Image segmentation integrating colour, texture and boundary information*. ISBN: 84-688-4933-2, 2003. Diposit legal: GI-1601.

Youtube. Neural network that changes everything - computerphile. Consistency, https://www.youtube.com/channel/UC9-y-6csu5WGm29I7JiwpnA, Accessed April 2018.