Bachelor Thesis

Managing crops from UAV images with data processing and Deep Learning techniques

Bachelor's degree in Aerospace Vehicles

Mir-Teimur Mustafaev

17 Jan of 2022

UPC
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Universitat Politècnica de Catalunya
ESEIAAT (Escola Superior d'Enginyeries Industrials
Aerospacials i Audiovisuals de Terrasa)

Supervised by
José Antonio Soría Perez

Abstract

Deep Learning is the state-of-the-art of Artificial Intelligence. Companies around the world put this technology to practical use. Accordingly, this work pretends to investigate the benefits of its use in agriculture in combination with unmanned aerial vehicles.

The report introduces Deep Learning by giving its definition and the inspiration behind this technology. Further, its functioning was studied to gain an understanding of how it is applied to real-world tasks and what variations of Deep Learning exist. Accordingly, the suitable Deep Learning algorithm and its implementation was selected to fulfill the objectives of this project.

The report proceeds with explaining the computer code that was written to utilize an already existing implementation and the problems that were encountered.

Finally, the discussion of the accomplished results is presented along with the conclusions of the entire project and comments on future works.

Resumen

Deep Learning es el estado del arte de la Inteligencia Artificial. Empresas de todo el mundo ponen esta tecnología en práctica. Por consiguiente, este trabajo pretende investigar los beneficios de su uso en agricultura en combinación con vehículos aéreos no tripulados.

El informe presenta Deep Learning dando su definición y la inspiración detras de esta tecnología. Además, se estudió su funcionamiento para comprender cómo se aplica a las tareas del mundo real y qué variaciones existen de su arquitectura. En consecuencia, se seleccionó el algoritmo de Deep Learning adecuado y su implementación para cumplir con los objetivos de este proyecto.

El informe continúa con la explicación del código de ordenador que se escribió para utilizar una implementación ya existente y los problemas que se encontraron.

Finalmente, se presenta la discusión de los resultados obtenidos junto con las conclusiones de todo el proyecto y comentarios sobre trabajos futuros.

Contents

List of Figures

Nomenclature

AESA	Agencia Estatal de Seguridad Area
AI	Artificial Intelligence
ANN	Neural Network
BGD	Batch Gradient Descent
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DL	Deep Learning
EASA	European Aviation Safety Agency
FNN	Feedforward Neural Network
GPU	Graphics Processing Unit
IoT	Internet of Things
ML	Machine Learning
ML	Machine Learning
MSE	Mean Square Error
NN	Neural Network
R-CNN	Region Based Convolutional Neural Network
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RoI	Region of Interest

RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SOA	State of the Art
TLU	Threshold Logic Unit
TPU	Tensor Processing Unit
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
VRAM	Video Random Access Memory

Documents

1. **Report**
2. Budget

1 Introduction

1.1 Background

The human population continues to grow and according to some estimates it will increase by 25% by 2050 [1]. This in turn will require an increase in the global production of various crops. Nonetheless, due to global warming and the consequent rise of temperatures, greater production rates will pose a considerable challenge for humanity, since the amount of arable lands will be decreased.

The number of agricultural products can be raised by either using more arable land or speeding up the process of the agricultural cycle. The latter is generally preferred since it addresses these problems more efficiently at a relatively low cost. For instance, one solution is to implement a network of small computing devices such as IoT [2] capable of providing the most recent and extensive information about the state of a crop. However, another solution, and perhaps a simpler one, is to take photographs from UAVs and analyze the land by means of image processing techniques based on artificial networks.

A drone is capable of doing regular flights above pre-established waypoints and taking images of the agricultural areas, covering a considerable amount of land in a short period of time thus reducing, or even eliminating, the necessity of farmers to move in order to inspect the different areas. Moreover, artificial neural networks can help to extract image features and classify different parts of the land in a matter of seconds and giving, for instance, the answer about whether the crops of a certain planted area have reached their maturity or require some treatment.

1.2 Aim of the project

This work aims to combine drone technology and Artificial Intelligence and show their efficiency to analyze crop land. Taking the work developed by [3] as a reference, this study analyses the growth process of potato harvest by means of AI-based image processing techniques.

The goal is to build a neural network model the detect the position of plants in the image and determine its growth stage maturity.

1.3 Scope of the project

The tasks required to fulfill the purpose of this project are mostly of either technical or practical nature. Some of the major ones in this list are:

1. **Study of the Deep Learning related SOA**. As many neural network models exists, certain specification considerations must be taken into account to choose which suits best for the problem at hand. For example, in Feedforward Neural Networks (FNN), the training stage is developed sequentially from the input layers to the output. They are used in simple classification tasks and speech recognition of small datasets and, therefore, they run faster. On the other hand, the Convolutional Neural Network structure (CNN) use filterbanks to images, and seems more suited for this work.

2. **UAV considerations**. Hardware and necessary flight settings related to the acquisition of images are considered briefly to provide insight to the reader about its programming.

3. **Configuration of Neural Network** (NN). For this project, the Mask R-CNN (*Mask region-based Convolutional Neural Network*) has been used. This model combines several CNN architectures with image processing mechanisms which are not implemented from scratch. The python library provided by Matterport [4] has been used to develop the necessary scripts. These include the 1) the training phase script; 2) the inference script; and 3) the Accuracy estimation script. The first script is used to train the network to recognize potato crops in different growth stages. The second and third will serve to assess its performance.

4. **Data Engineering tasks**. Acquired images have been labeled using the *Labelme* software package [5]. To this goal, three mask classes representing the different growth stages of potato plants have been set: pat1, pat2 and pat3. A total of 303 images were obtained from different positions, both from land and aerial pictures. They have been used to construct three datasets: *training* (75% of the images), *validation* (10%) and *test* (15%).

In general, similar works using Mask-RCNN have shown good results [6, 7]. However, this work is set in the realm of UAVs using one for purposes of agricultural management. The aforementioned drone, developed in [3], will be programmed to loiter above certain agricultural waypoints to take pictures of the crops that will later be used to train the neural network offline. It was constructed on the base of a standard kit that consisted of a carbon fiber frame, 4 brushless JMT HYD 3508 motors, helices and a set of additional components [8]. The battery supplied 11.1 V at 4200 mA which allowed the drone to fly stably for 20 minutes.

The hardware used for flight control is APM 2.8 that ran ArduPilot which is an open source autopilot system. The flights were configured with Mission Planner program [9].

For remote control, the Radiolink AT9S PRO transmitter of 9 channels with R9DS receiver was utilized. Both functioning at 2.4 GHz radio band.

GoPro Hero 4 was used for taking pictures. It has a resolution of 12 MP and accepts JPG images and MP4 videos. One of the considerable advantages of GoPro cameras is that they have GitHub libraries which allow them to be controlled by Python scripts [10]. Nonetheless, for this work, GoPro was connected to a phone via mobile internet to control image acquisition process.

The performance of the trained network is assessed using: 1) loss plots, 2) confusion matrices and 3) inference examples. The first demonstrated the progress of the learning process, the second presented the quality of predictions for different classes and the last showed how the network perceives different objects.

1.4 Motivation

This project is motivated by two main goals: 1) mitigation of the effects of climate change, and 2) study of Deep Learning algorithms and UAV applications.

On the one hand, public awareness of climate change and the effects of human activity are impacting global earth warming in unprecedented ways, and it's no longer doubtful that soon enough it will have a drastic effect on our lives. Engineers and scientists around the world are trying to come up with new ways and technology to slow the rate at which this is happening. Unfortunately, some engineering specialists can do only so much to help the cause, as engineering fields are not particularly multidisciplinary, which is required to address this problem. Nonetheless, one of the primary goals of this project is to attempt to show that even aerospace engineering and machine learning, by using UAVs and Deep Learning, can help humanity to adapt to the new reality of extreme weather conditions.

The second motivation of this project is to increase my knowledge in two subjects that consistently gain popularity due to their promising potential for engineering - artificial intelligence (AI) and UAVs. In my opinion, getting more familiar with these two subjects is not only intriguing but also advantageous for any individual looking for being in demand for many employers.

1.4.1 Overview of the manuscript

This section aims to provide the overview of this work. Chapter 2 presents the state-of-the-art of Deep Learning. Its definition, functioning and applications are discussed. Once the basics are explained, the chapter proceeds with selection of the appropriate type of neural network and its implementation. Since the work is also related to the UAV technology, its definition, legislation and relevance for this project is also studied.

In Chapter 3, the process of acquiring the data, preparing it and writing the programming script is discussed.

Chapter 4 presents the results and discusses them in depth. In particular, the loss plots and confusion matrices are presented and studied to explain the evolution of the project.

Finally, Chapter 5 concludes the work with relevant considerations and discusses the future projects that could be done on the basis of this one.

1.5 Project deliverables

The project deliverables will consist of the report and the trained neural network capable of detecting potatoes in various growth stages.

2 State of the Art

2.1 Deep Learning

Deep Learning is a subfield of Machine Learning that works with artificial neural networks to solve computational tasks (Fig. 2.1). An ANN is inspired by the operational behavior of a biological neuron which consists of a cell body that generates electrical impulses conducted away through the axon. The dendrites are tree-structures that receive the electrical impulses from other neurons which are distributed along the network through specialized junctions called synapses (Fig. 2.2).

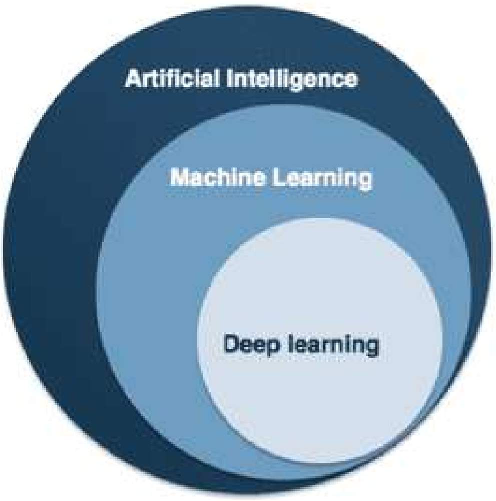
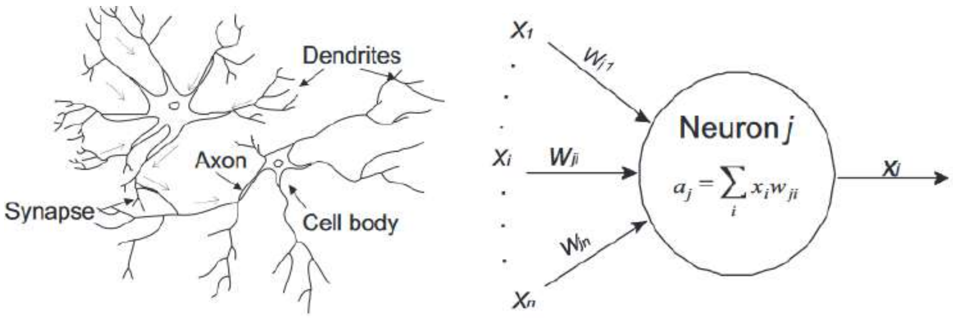


Figure 2.1: Difference between AI, ML and DL [11]

The word "deep" comes from the fact that neural networks are multilayered. Layers are categorized into three main groups: input, output, and hidden layer. Each layer is composed of single computational units, called perceptrons. The input and output are the first and last layer, respectively. However the complexity of this learning algorithm comes into play in the hidden layers where the knowledge is contained. Starting from the input layer, the hidden layers are configured in such a way that the output of each one is connected to the input of another forming a tree structure (just as the dendrites) to perform more complex calculations. The relevance of each neuron in this calculation is determined by the weight value, and the main goal of the training process consists in finding the correct value of such weights so that the network converged to the desired outputs. In addition, a neural network can have one or multiple outputs whose nature, either analogical (real value), digital or a matrix forming an image, may vary depending on the application at hand.

Therefore, Deep Learning is considered to be the most prominent attempt to recreate human intelligence by some of the world-leading experts in the field of Artificial Intelligence [12]. One of the reasons is the presence of several key similarities between the mathematical and biological structures [13]. First, the nodes described above are analogous to biological neurons in which they both act as highly connected computational devices. Second, similar to the synaptic strengths of the biological neurons, all the connections vary in importance and can be adapted to produce the required results.



(a) A biological neuron **(b)** An artificial neuron

Figure 2.2: Basic similarities of artificial and biological neuron[13]

The second reason for the prominence of this technology lies in the constant advances and improving of results that take place in many application fields of Deep Learning [14]. Nowadays, people have the opportunity to store large amounts of data and perform calculations much faster than in the previous century. As a consequence, neural networks can be continuously trained in order to improve their performance.

2.2 Training a perceptron

Mathematically speaking, a perceptron is a TLU (Threshold Logic Unit) carrying out the following operation:

$$y = g(\sum_{i=1}^{n} w_i x_i + b_i) \tag{2.1}$$

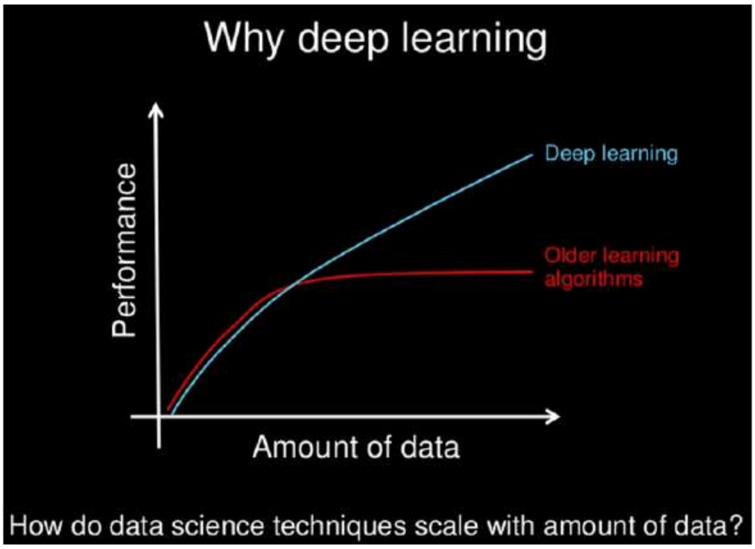


Figure 2.3: How Deep Learning is improved with more data [12]

where x_i represents the inputs with weights w_i applied to them which are then summed up and passed to an activation function $g(\cdot)$ to obtain the output.

The parameter b in this expression represents the bias which is configured to fit best for a dataset in a given application and simplify the process of learning [15]. A higher bias suggests making many assumptions and allows faster learning convergences to be obtained but makes the model less flexible and less precise.

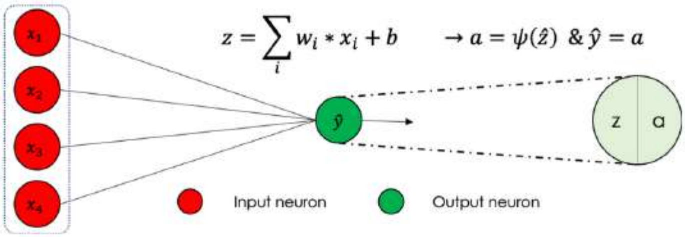

$z = \sum_i w_i * x_i + b$
$\rightarrow a = \psi(\hat{z})$ & $\hat{y} = a$
x_1
x_2
x_3
x_4
$\hat{y}$
z
a
Input neuron
Output neuron

Figure 2.4: Perceptron [16]

The activation function, on the other hand, is a non-linear density probability function, whose aim is to produce a value that can be separated into many different sets (or classes). Non-linearity is necessary to solve complex, real-world tasks or, otherwise, the outputs of each neuron are stacked into a simple linear regression, making the neural network useless for many real problems.

In this sense, ANN can be thought of in terms of its biological counterpart. For instance, the human eye retina receives light rays, the axons of ganglion cells then emit electric signals that are being transferred to the neurons [17]. Then, other neurons that are responsible for the recognition of objects in the image activate and allow objects to be recognized by a person. The way this is done in ANN is by applying threshold to numbers, which is the activation function's job.

Some of the most common activation functions are sigmoid, hyperbolic tangent, and ReLU (Rectified Linear Unit).

Sigmoid:

$$f(x)=\frac{1}{1+e^{-x}} \tag{2.2}$$

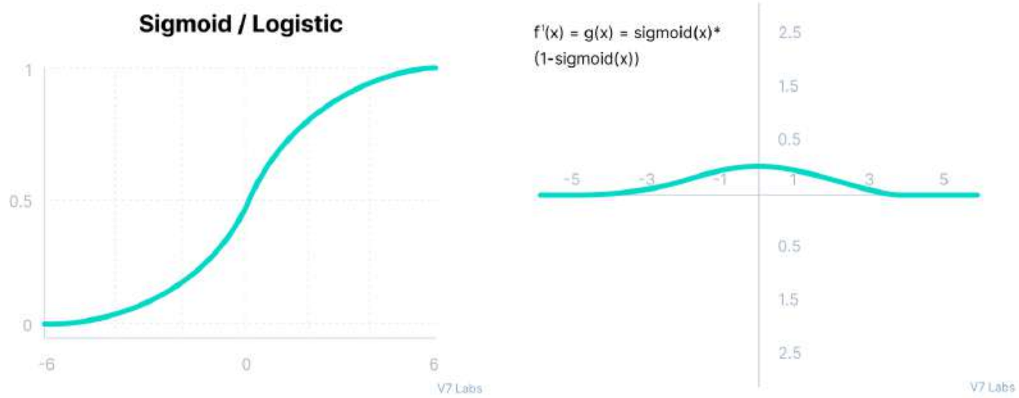
Hyperbolic tangent:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

Rectified linear unit:

$$f(x) = max(0, x) \tag{2.4}$$

The sigmoid function (Fig. 2.5) produces a value between 0 and 1 and is common in application models where a probability has to be measured [18], or for generating discrete probabilistic values associated to the labels that represent different classes in classification problems. For instance, a model prediction between a cat, a dog and any other pet may be generated. The main requirement for the activation function is that it must be differentiable to provide smooth gradient profiles to avoid discontinuities (see section 2.2.2).

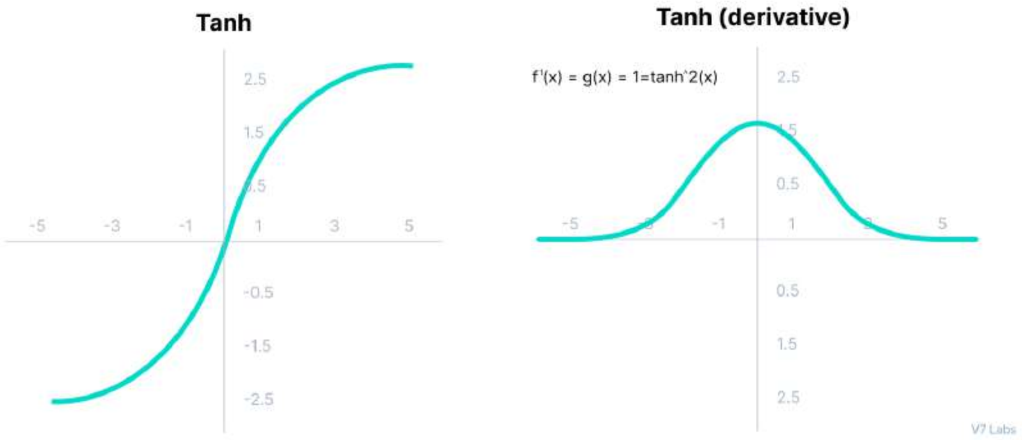


(a) Sigmoid function

(b) Sigmoid function's derivative

Figure 2.5: Non-linear functions(I) [18]

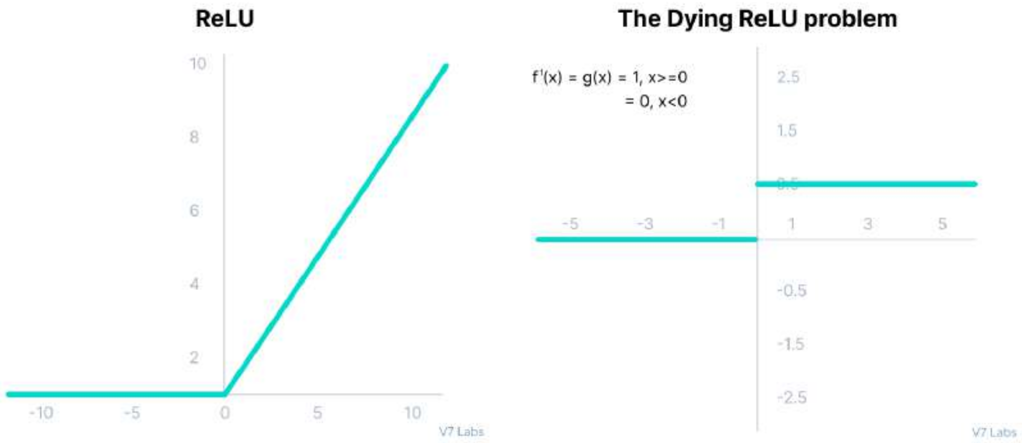
The hyperbolic tangent (Fig. 2.6) is a shifted version of the previous example that gives values between -1 and 1 which is preferable in regression problems:



(a) Hyperbolic tangent function **(b)** Hyperbolic tangent's function derivative

Figure 2.6: Non-linear functions(II) [18]

The ReLU function [2.4] in Fig. 2.7 will only activate the perceptron if the input value is bigger than zero. This is one of the most utilized activation functions in data-science as it allows for fast convergence of the backpropagation learning algorithm since negative outputs are set to zero [19]. However, the weights and biases of some neurons are not updated, which causes the problem of dead neurons to appear within the structure. Hence, different variations of ReLU and other solutions are used to solve this problem [20].



(a) ReLU function **(b)** ReLU function's derivative

Figure 2.7: Non-linear functions(III) [18]

2.2.1 Backpropagation

In an architecture of many layers, the expression of a single perceptron [2.1] can be generalized to a multi-layer network as:

$$f_i\left(\boldsymbol{x};\mathbf{W}\right)=g\left(b_{0,i}^{(K)}+\sum_{j=1}^{n_{K-1}}g\left(z_{K-1,j}\times w_{j,i}^{(K)}\right)\right)\tag{2.5}$$

The *forward propagation* is a step-by-step procedure necessary to the calculation of [2.5] where$f_i\left(\boldsymbol{x},\mathbf{W}\right)$ is the actual prediction according to input $\boldsymbol{x}\in\mathbb{R}^d$, where d represents the feature dimension of the input and K is the total number of layers. The number of outputs i has nothing to do with the input dimension d, and $z_{k,i}=b_{0,i}^{(k)}+\sum_{j=1}^{n_{k-1}}g\left(z_{k-1,j}\times w_{j,i}^{(k)}\right)$ is the output value of the i-est position in layer k, where $\boldsymbol{z}_0=\boldsymbol{x}$.

The learning phase, however, develops in the reverse order with the goal of finding the weights $\mathbf{W}$ that solve the application at hand. And, hence, the name *backpropagation.* This is a topic which has led many experts in the data science community to provide different solutions to overcome the obvious computational limitations of this optimization task.

For a network with the given input $\boldsymbol{x}$, a *loss function* $\mathcal{J}\left(\mathbf{W}\right)$ is used as a metric for determining the deviation between the actual and predicted values. One possibility for this metric is the MSE (*Mean Square Error*):

$$\mathcal{J}\left(\mathbf{W}\right)=\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(f\left(\boldsymbol{x}^{(i)};\mathbf{W}\right),y^{(i)}\right)\tag{2.6}$$

where $\mathcal{L}\left(x,y\right)=\left(x-y\right)^2$. That is, all outcomes from the samples in the training are compared and averaged to obtain a value representing the discrepancy that can be used by the NN to correct itself. For detection problems using binary classifiers, it is preferable to use the *cross entropy* function:

$$\mathcal{L}\left(x,y\right)=-\left(y\times\log x+\left(1-y\right)\times\log\left(1-x\right)\right)$$

Therefore, to train the network it is necessary to find the values $\mathbf{W}$ that minimize the cost function [2.6]. That is:

$$\mathbf{W}^*=\arg\min_{\mathbf{W}}\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(f\left(\boldsymbol{x}^{(i)};\mathbf{W}\right),y^{(i)}\right)=\arg\min_{\mathbf{W}}\mathcal{J}\left(\mathbf{W}\right)$$

However, this is a task that requires a lot of computational power, as computing the gradient $\frac{\partial\mathcal{J}(\mathbf{W})}{\partial\mathbf{W}}=0$ to obtain the optimal $\mathbf{W}^*$ becomes infeasible in almost all real applications. Therefore, some sort of a search algorithm is necessary. This is the main principle behind the gradient descent method (Fig. 2.8).

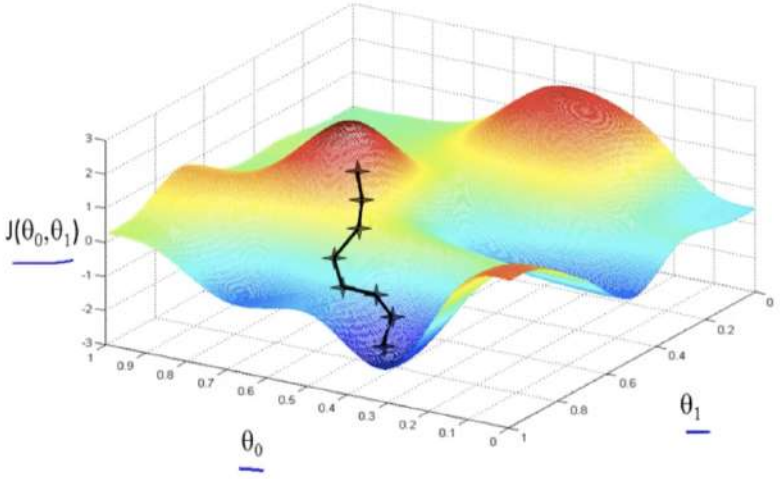


Figure 2.8: Gradient descent curve [21]

The underlying idea of this process is to decrease the error with small updates of each weight in the network. And the way these updates are determined is by taking small steps in the direction opposite to the maximum ascent. The size of these steps is determined by the learning rate, η. Therefore, the main steps of stochastic gradient descent algorithm are as follows:

1. Weight initialization $\sim \mathcal{N}\left(\mu, \sigma^2\right)$.
2. Repeat the next loop until convergence: $\frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}} \leq E$,
3. Gradient calculations: $\frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$
4. Update of network weights: $\mathbf{W}_t \leftarrow \mathbf{W}_{t-1} - \eta \times \frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$

Each loop t of this process constitutes one pass (or epoch) of the whole training algorithm. A common practice in step one is to configure the weights randomly with a Normal distribution $(0, \sigma^2)$, whereas E is configured to a non-zero consistent with the learning goal in order to avoid infinite loops. The third step is calculated using the chain rule. Fig. 2.9 illustrates an example of how it is done in practice:

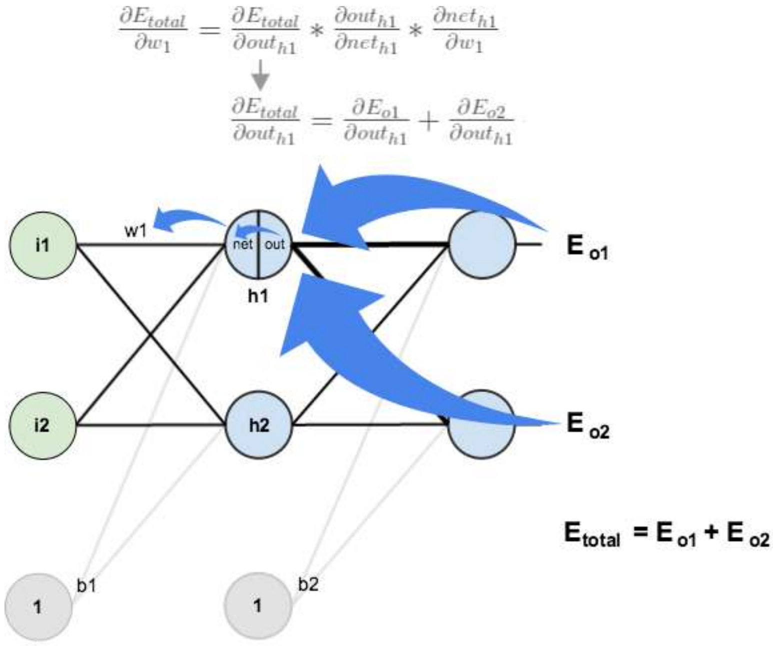

$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$
$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$
i1
w1
net
out
h1
E o1
i2
h2
E o2
$E_{total} = E_{o1} + E_{o2}$
1
b1
1
b2

Figure 2.9: Example of gradient calculation [22]

2.2.2 Limitations of Deep Learning

The update of weights is controlled by the learning rate η. The learning rate affects the step size during the backpropagation (Fig. 2.10).

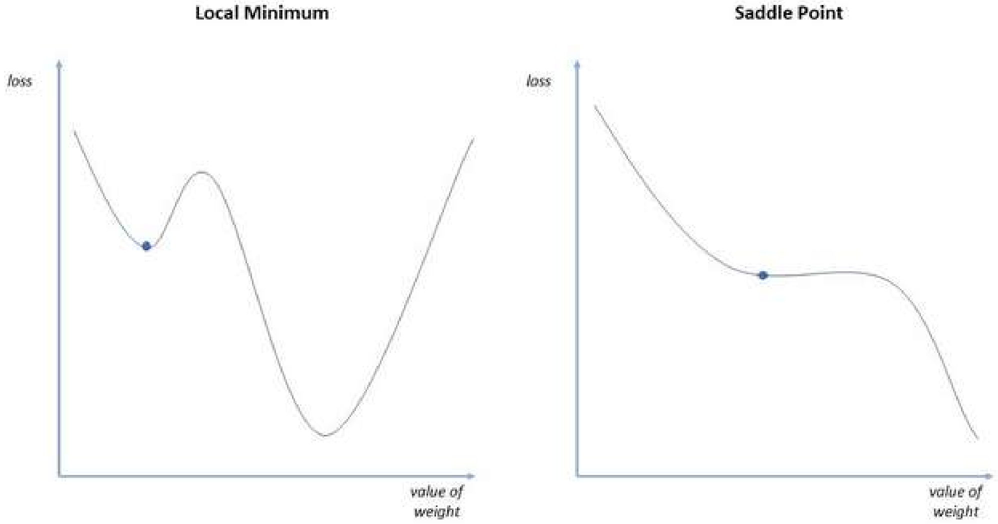


Figure 2.10: Effect of the learning rate [23]

On the one hand, if η is too small, the step is not big enough to overcome local minima which results in the precision reduction. Moreover, if there are many saddle points, the convergence will take longer times [23]. Both of these problems are represented in Fig. 2.11.

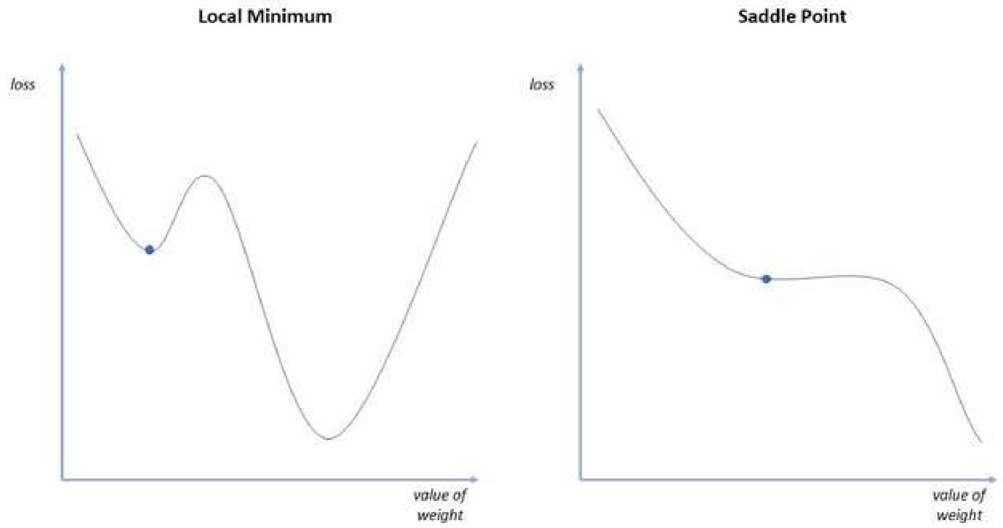


Figure 2.11: Example of a local minimum and saddle point [23]

A solution to this is to use adaptive learning rates, when the parameter is no longer fixed and can become larger or smaller during the training phase depending on the situation. This solution is implemented in such methods as ADAM, AdaDelta and AdaGrad [24, 25, 26].

In addition, calculating the gradient in step 3 can be a very computationally extensive task not feasible for real life applications. Some approaches do not use all the data points of the training set for this step. Using less data points has the effect of reducing the processing time at the cost of obtaining rough estimates of the gradient in each epoch. For example, SGD (*Stochastic Gradient Descent*) picks a data point of the training set randomly to compute the gradient, and then changes the data point in the next epoch. Whereas BGD (*Batch Gradient Descent*) picks a small set of data points and calculates the gradient as the average of all single SGDs of this batch.

BGD allows to converge to the target much quicker because the gradient computation is more accurate, and one can expect better gradient updates using large learning rates. In addition, thanks to the advent of GPUs, another approach known as *mini-batch* permits the parallelization of BGD computation and increases the speed significantly.

Another limitation has to do with gradient calculations. If the gradient approaches zero in some neurons, weights may stop being updated (see subsection 2.2). This happens because of the activation function derivative present in the chain rule. On the other hand, large gradients force the model to be very unstable and precision results show a random behavior each time the model is trained and evaluated in the inference stage.

Lastly, underfitting and overfitting are problems that are related to the number of epochs t used during the training stage (Fig. 2.12). Underfitting occurs when the model is not trained sufficiently. In here, t is too small and the network does not have the capacity to fully learn the data, which gives many incorrect predictions. This issue also occurs when the dataset is not representative enough of the problem to be solved. On the other side, when the model is trained beyond necessity (t too big) overfitting occurs. In this case, the problem is that the model loses its capacity of generalizing the problem, making it more sensitive to possible variance noise present in the input data.

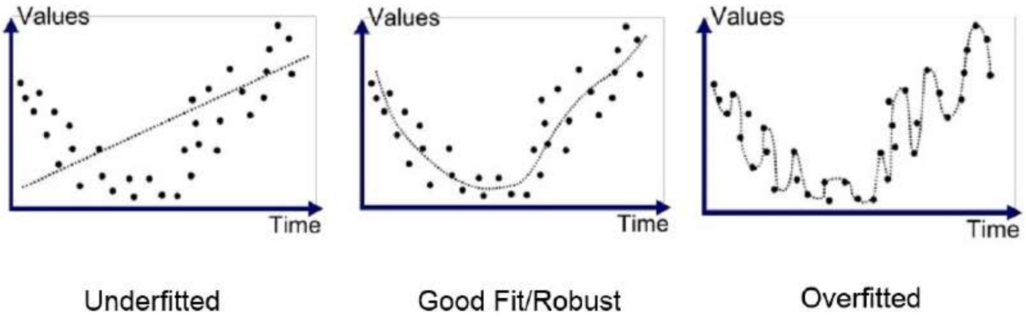
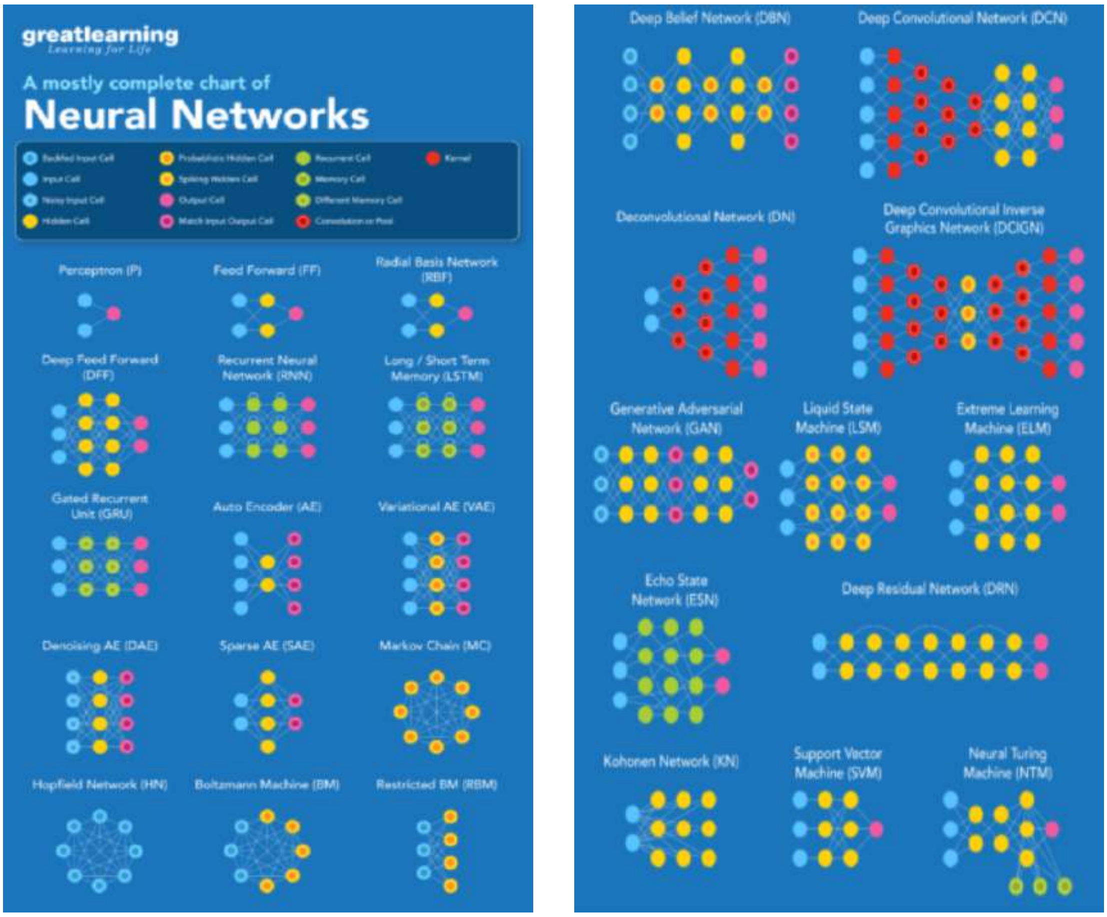


Figure 2.12: Classifications of fitting problems [27]

In general, deep neural networks require large amounts of data to learn and achieve acceptable precision.

2.3 Types and examples of use of ANNs

There is a multitude of types of neural networks [28] whose detailed explanation goes beyond this work. Fig. 2.13 presents a small overview of the majority of them. Nevertheless, some are worth discussing because they lie at the core of the technologies that increasingly influence our day-to-day life



(a) Classification of ANN(I) **(b)** Classification of ANN(II)

Figure 2.13: Classification of NN [29]

2.3.1 Feedforward Neural Network

FNNs (Feedforward Neural Networks) were the first to be tested in tasks of the computer vision field, speech recognition and machine translations among others, because they are relatively easy to train. Besides, every state-of-the-art NN today implements at least some part of the feedforward architecture.

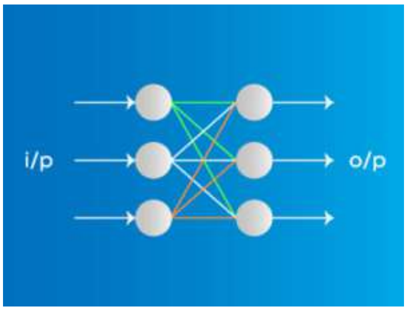


Figure 2.14: Feedforward Neural Network [29]

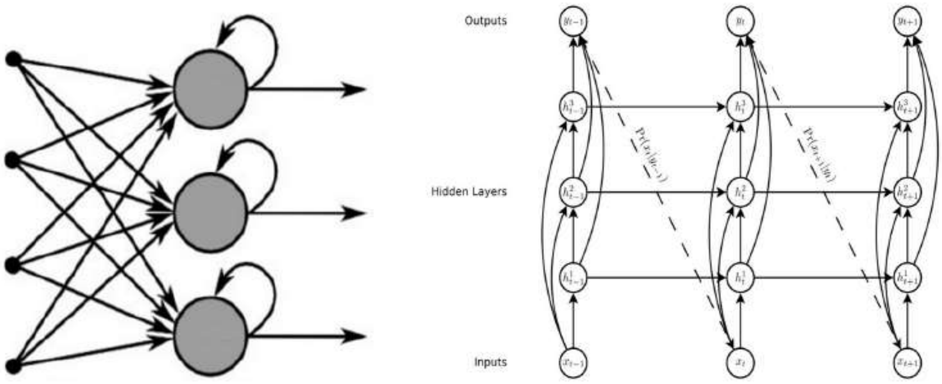
FNNs have shown certain utility in processing biomedical data for the early prediction of cancer diseases. One of the variations of FNN is trained to make an early prediction of cancer diseases. A massive amount of numerical data is produced by the healthcare system every day. By taking advantage of that data, the doctors and scientists managed to decrease the cancer death rate by 19% between 1999 and 2017 [30]. This dropdown is attributed to the faster prognoses due to the availability of big amounts of data. According to [30], neural networks can accelerate the process of prognosis. The author details the process of training a LogSLFN(Logistic single-hidden layer feedforward neural network). Two standard techniques of classifying cancer were used to determine the correct weights: (a) the use of ribonucleic acid sequencing for distinguishing cancer types, and (b) METAVIR score (the amount of scarring and inflammation in the liver biopsy and the rate of fibrosis). The results were equivalent to the state-of-the-art ML techniques and have big potential for increasing the performance.

Artificial intelligence is found in the building construction industry where FNNs are used to analyze the dynamic structure of skyscrapers [31]. In [31], *Huile et al* propose a methodological framework for VBI (*Vehicle-Bridge Integration)* system dynamic response prediction using an FNN and a variation of recurrent neural network. In this application, obtaining the dynamic responses between the vehicle and the track is crucial for assessing the structural and running stability. The study demonstrated that, although the track irregularities and high noise levels can have negative effect on the results, with proper training, the framework can provide accurate predictions and be put to practice in high-speed railways.

Lastly, FNNs can be applied in astronautics. *Zheng et al* [32] propose a novel way of solving the Tschauner-Hempel equations. These equations describe a spacecraft's motion relative to another spacecraft in an elliptical orbit. This problem does not have an analytical solution and was previously solved only by computationally expensive numerical means. As a result, the existing methods could not be employed in real-time applications. However, the authors of this study claim that a trained neural network is capable of dramatically reducing the computational time. According to their results, the new solution performs much better with certain typical problems such as close-range rendezvous and formation flying.

2.3.2 Recurrent Neural Network

Recurrent neural networks (RNN) are considered to be the most promising variation of DL algorithms. Unlike FNNs, information does not flow in one direction only. The output of some neurons in the network is not fed only to the neurons in the next layer but also to some of its other parts (Fig. 2.15). Additionally, the output of each neuron may also depend on its previous outputs, which is ideal for the development of sequential time systems. This feature makes RNNs responsive to real-time events, i.e., it can be very useful for speech recognition tasks, self-driving cars (SDC), natural language processing (NLP), and electronic hardware design among others.



(a) **(b)**

Figure 2.15: Examples of RNN [33, 34]

Today people already possess smart artificial assistants such as Siri, Alexa, Cortana, and others that are capable of correctly interpreting human speech. Nonetheless, even though they are capable of giving impressive results being based on traditional NLP algorithms, these assistants still leave room for improvement. More and more experts regard deep neural networks as the future of speech processing. Even now, Amazon scientists are applying DL to improve customers' experience with Alexa [35]. In [36], the scientists used a variation of RNN - Bidirectional Gated Recurrent Unit (BiGRU). As a result, the authors managed to improve the quality, intelligibility, and word error rates by 35.52%, 18.79%, and 19.13%, respectively.

Recurrent networks find applications in energy engineering. The efficiency and safety of lithium-ion batteries can be improved by accurate estimation of the state of charge (SOC) and capacity. Most of the present estimation methods leave a lot to be desired in terms of cost and efficiency. Notwithstanding, data-based estimation has great prospects. The Chinese researchers in [37] propose to train a neural network on the battery aging datasets to solve this problem. They have used several architectures - the long short-term memory (LSTM) and gated recurrent unit (GRU), mentioned previously. The obtained results showed higher accuracy than those of the traditional methods.

Technical diagnosis is another task that can be improved with RNNs. The photovoltaic systems are often affected by different kinds of faults that reduce their performance. The existing methods are overly expensive, which reduces their applicability. However, the scientists found a way to reduce the expenses and improve the monitoring process [38]. The research consisted of training a neural network with LSTM and GRU that would be capable of determining up to 6 types of faults. The data used for the training consisted of the produced voltage, current, irradiance of the PV system, and others. The result demonstrated that, unlike the state-of-the-art methods, the neural network was not only capable of determining the fact of fault but also its severity.

2.4 Convolutional neural network

Previous network architectures only allow to process the input data in the form of real numbers. But for processing images, some changes have to be made. In this sense*, Convolutional Neural Network* (CNN) is the most suitable type of network for this application. In CNN, perceptrons are replaced by convolutional RGB mask filters. Hence, the new weights of this architecture are array numbers of rectangular matrices.

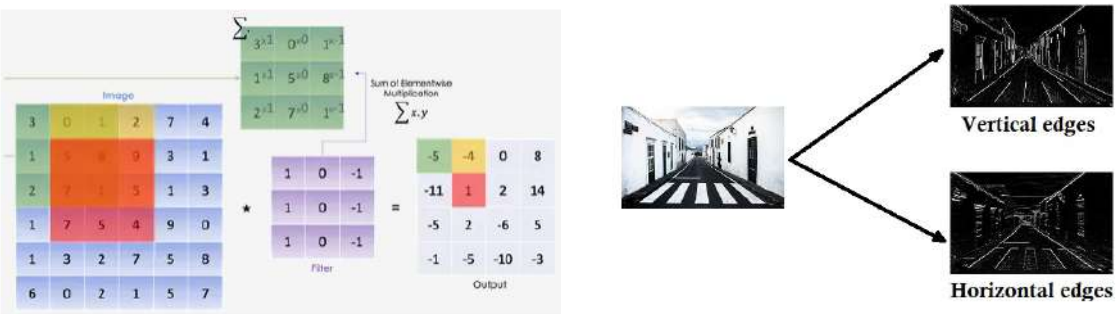
During the training process, the filterbanks are modified to highlight particular features which are later used in the detection process. This behavior makes CNN especially useful for self-driving cars. For instance, Tesla's autopilot is based on convolutional neural networks and it's already capable of driving autonomously in cities [39].

Other examples of usage include face detection [40], accurate medical image-based diagnoses [41], and drug prediction [42]. Whenever there is a need to do image analysis, provided there are enough samples, deep convolutional neural networks are the best candidates for the task.

An image is simply a three-dimensional tensor, where each slice corresponds to a channel in the RGB structure (red, blue or green). Accordingly, each value represents the light intensity in one of those colors. The intensity and the color of each pixel is obtained by summing three channel numbers in each position. A high-definition image can contain a total of $1920 * 1080 * 3 = 6220800$ data points. In connection with that, CNNs require computational units of intensive data processing such as GPU.

2.4.1 Architecture of a CNN

The process of feature extraction is done by the cascading convolution, threshold and maxpooling operations. The convolution is a well known operation in computer vision which updates each pixel in the image with a weighted sum of its neighbors by means of a kernel filter [43].



(a) Element-wise multiplication [16]

(b) Result of the convolution [44]

Figure 2.16: Convolutional product

Depending on the values of the 2D filter/kernel, this process can discover vertical and horizontal edges, tone degradation and colors of different contrast and light settings.

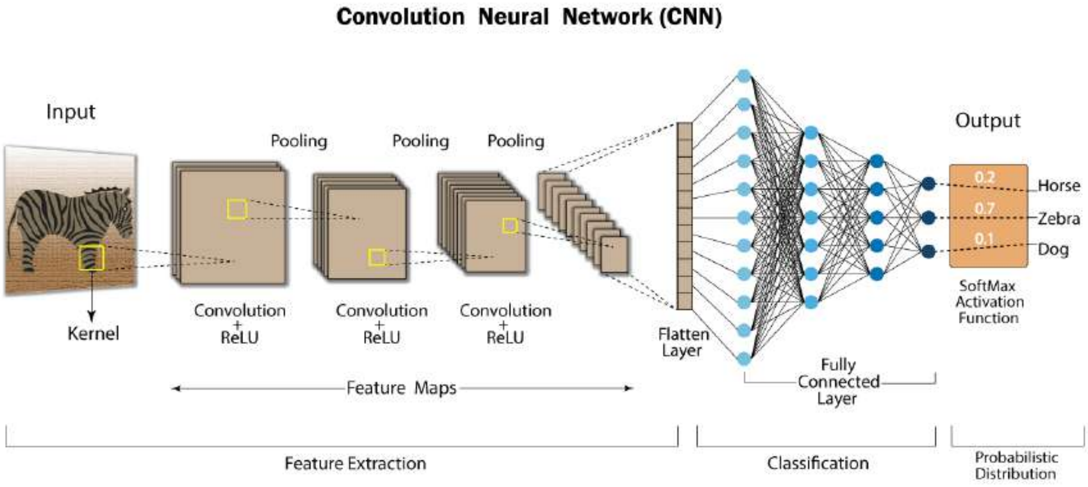


Figure 2.17: Structure of CNN [45]

To understand the utility of convolutions, the four situations in Fig. 2.18 can be assumed. To find the objects in the image, the analysis should characterize variations by color, texture scale and enclosure localization. Spoons, dishes and glasses on the table in Fig. 2.18 have different scales, while the cats in Fig. 2.18 are distinguished by color but not by the size or scale. On the other hand, the chameleon in Fig. 2.18 is distinguished by the texture but not by color. And finally, detecting the car in Fig. 2.18 requires the usage of general features since the parts of the car such as wheels or windscreen belong to the car but don't have the same color, texture or shape profile. Therefore, taking into account such situations can be challenging and Artificial Intelligence is necessary to address object classification in images.

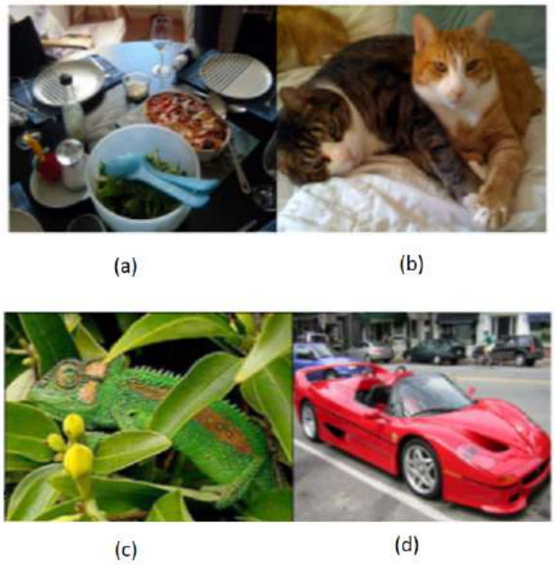


Figure 2.18: Complexity of object characterization in image segmentation [44]

The idea behind CNNs is that such knowledge can be learned by properly configurating the values of filterbanks. In this sense, the first layers are responsible for capturing single structures, such as vertical and horizontal lines, curves and irregular shapes (Fig. 2.19). An operation called *maxpooling* reduces the size of the image in the subsequent stage (see subsection 2.4.2). This causes the network to lose the ability to retain localized features but gains in image generalization. In this manner, combining the feature matrices from different layers, complex structures such as the shape of a house, a car, a crop field, and other object types can be learned from images.

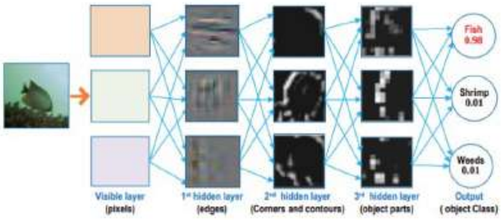


Figure 2.19: CNNs: Localization vs Generalization [44]

Typically, filters have a 3 × 3 size in all layers [46]. With each passing layer the number of filters increases to cover the feature spectrum of an image.

VGG16 (Fig. 2.20) is a standard structure widely used in the computer vision field. It is composed of 13 convolutional layers combined with 13 max-pooling layers and 3 fully-connected layers of perceptrons with a classical ANN structure. Since the parameters can be adjusted only in convolutional and full-connected layers, the number of layers with tunable parameters is 16. Hence, the name VGG16. The number of filters in the first block is 64, then this number is doubled in the later blocks until it reaches 512. This model is finished by two fully-connected hidden layers and one output layer. Both fully connected layers have 4096 neurons. The output layer consists of 1000 neurons. So, in principle, the CNN is able to assign up to 1000 different classes.

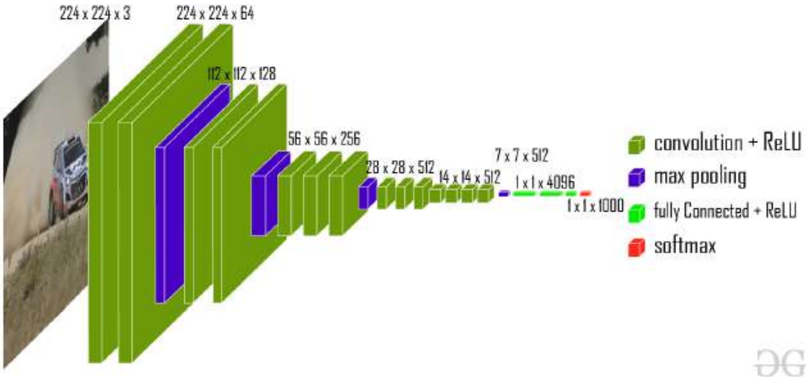


Figure 2.20: Structure of VG16 Convolutional Neural Network [47]

2.4.2 Convolution and pooling calculation

In some applications, computing the convolution requires preparing the image and configuring some parameters to move the filter along the image.

The border pixels of the image are affected by convolution calculations since the area of the kernel filter does not entirely overlap with the image in those regions. To minimize this problem, habitually the size of the image is increased with new pixels with artificial values. This operation is known as ***padding*** and consists in adding zeros along the contour if the image border. The degree of padding is determined by the padding parameter p. That is, $p = 1$ represents one loop of zeros along the image contour (Fig. 2.21). On the other hand, the ***stride*** specifies the number of pixel shifts used by the filter to move and convolve along the image area. In the example of Fig. 2.16a the stride is set to $s = 1$.

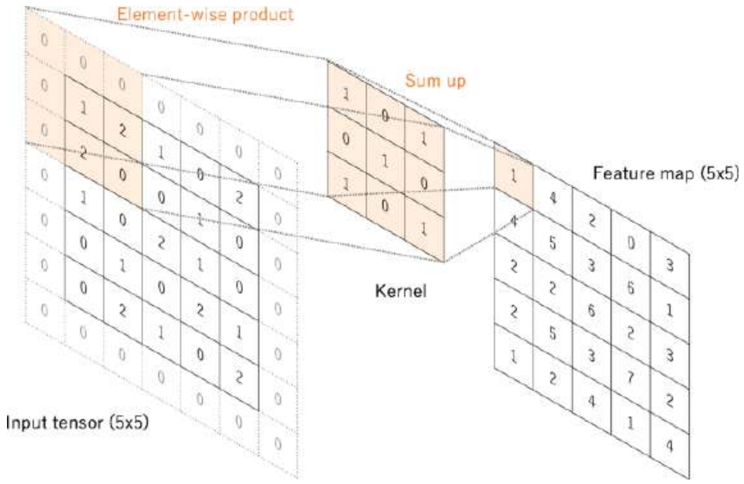


Figure 2.21: Example of padding [48]

An image can be seen as a tensor of dimension $I \in \mathbb{R}^{n_h \times n_w \times n_d}$ of height n_h and width n_h and depth n_d. Colored images have n_d=3 corresponding to each of the RGB channels. The filter kernel, however, is a square matrix of the same depth $F \in \mathbb{R}^{f \times f \times 3}$ where the side length is an odd number (generally $f = 3$ in CNNs).

With these definitions, the convolution at coordinate position (x, y) is obtained as:

$$C(I,F)_{x,y} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} \sum_{k=1}^{n_d} F_{i,j,k} I_{x+i-1,y+1-1,k} \tag{2.7}$$

This produces a two-dimensional tensor with dimensions:

$$D_C = (\left\lfloor \frac{n_h + 2p - f}{s} +1 \right\rfloor , \left\lfloor \frac{n_w + 2p - f}{s} +1 \right\rfloor) \tag{2.8}$$

where $\lfloor a \rfloor$denotes floor function of a.

The pooling operation, on the other hand, is a subsampling process that reduces the image to height n_x and width n_y keeping the depth n_c intact. To do this, first the image must be divided into subareas of size (n_{dx}, n_{dy}) to calculate the pixel that represents each subarea. For this reason, n_{dx} and n_{dy} must be multiples of n_x and n_y, respectively (Fig. 2.22). One possibility is to use the average value of all pixels in a subarea:

$$\phi_{\text{avg}}\left(I\right)_{x,y,c} = \left\lfloor \frac{1}{n_{dx} \times n_{dy}} \sum_{i=1}^{n_{dx}} \sum_{j=1}^{n_{dy}} I_{x+i-1,y+j-1,c} \right\rfloor \tag{2.9}$$

Another solution is to simply use the cell of maximum intensity:

$$\phi_{max}(I)_{x,y,c} = max(I_{i,j,c}) \tag{2.10}$$

where $i \in [1, ..., n_{dx})$ and $j \in [1, ..., n_{dy})$.

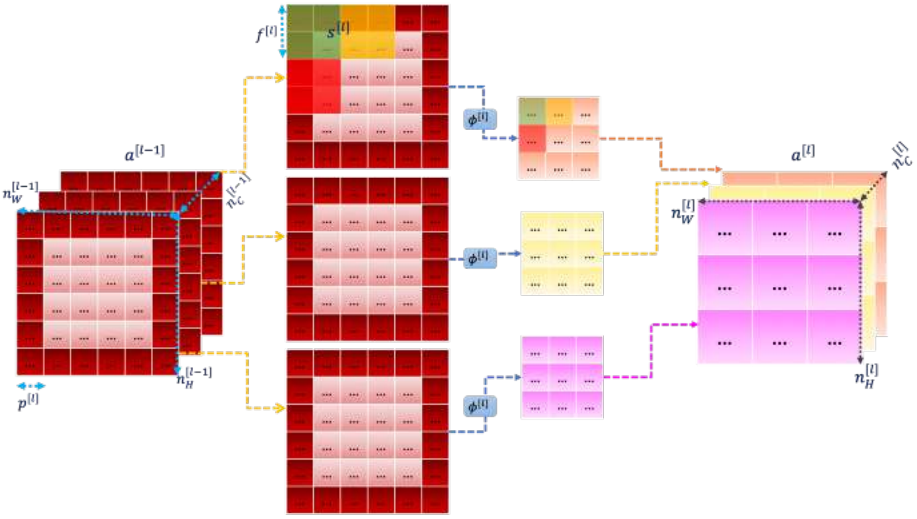


Figure 2.22: Example of pooling operation using $s = 2$ [16]

In this sense, some authors believe that using strides $s > 1$ convolution products is more beneficial than using pooling to obtain good generative models [46].

2.4.3 Layers of CNN

As explained above, a CNN iterates convolution/maxpooling operations to obtain feature maps of different size and scale representations. More precisely, each layer may consists of the following:

- A *convolutional* layer followed by an activation function $g(\cdot)$
- A *pooling* layer
- A fully connected ANN layer comprised of multiple perceptrons (see 2.2.1)

For a single layer λ, the parameters are defined as follows:

- $I^{[\lambda-1]}$ is the feature input of size $(n_h^{[\lambda-1]}, n_w^{[\lambda-1]}, n_c^{[\lambda-1]})$,where $I^{[0]}$ is the original image
- $p^{[\lambda]}$ is the padding
- $s^{[\lambda]}$ is the stride
- $\phi^{[\lambda]}$ is the activation function
- $F^{(n)}$ is the filter(kernel) of size $D_F = (f^{[\lambda-1]}, f^{[\lambda-1]}, n_c^{[\lambda-1]})$, where $\forall n \in [1, 2, ..., n_c^{[l]}]$

- $b_n^{[\lambda]}$ is the bias of n^{th} convolution
- $a^{[\lambda]}$ is the output of size $(n_h^{[\lambda]}, n_w^{[\lambda]}, n_c^{[\lambda]})$
- x and y are the positions in the output, where $x \in [1, 2, ..., n_h^{[\lambda]}]$ and $y \in [1, 2, ..., n_w^{[\lambda]}]$

With these definitions, the convolution/activation operation at layer λ is computed as:

$$C(I, F^{(n)})_{x,y} = \phi^{[\lambda]}(\sum_{i=1}^{n_h^{[\lambda-1]}} \sum_{j=1}^{n_w^{[\lambda-1]}} \sum_{k=1}^{n_d^{[\lambda-1]}} F_{i,j,k}^{(n)} I_{x+i-1,y+j-1,k}^{[\lambda-1]} + b_n^{[\lambda]}) \tag{2.11}$$

Thus, the feature map of each layer is a tensor of the form:

$$a^{[\lambda]} = [\phi^{[\lambda]}(C(I^{[\lambda-1]}, F^{(1)})); \phi^{[\lambda]}(C(I^{[\lambda-1]}, F^{(2)})); ...; \phi(C(I^{[\lambda-1]}, F^{(n_c^{[\lambda]})}))] \tag{2.12}$$

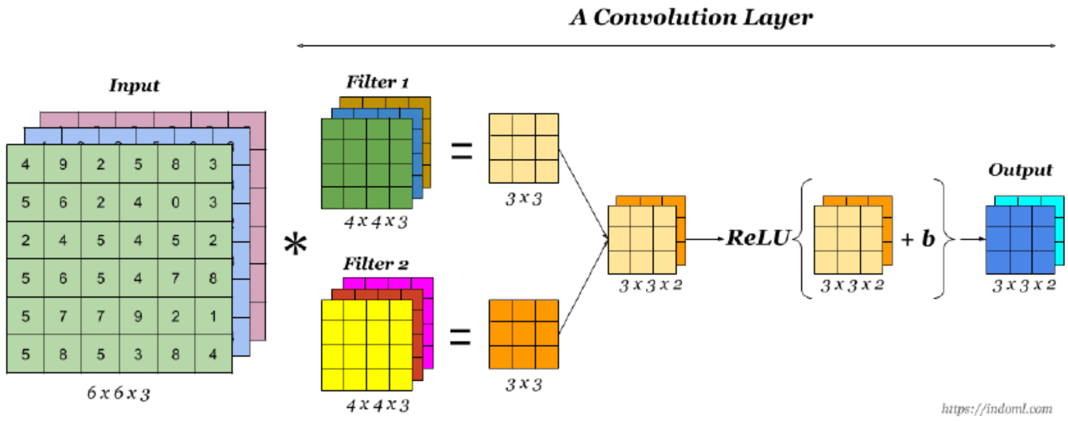
The dimensions of $I^{[\lambda]}$ are the following:

$$D(I^{[\lambda]}) = (n_h^{[\lambda]}, n_w^{[\lambda]}, n_c^{[\lambda]}) \tag{2.13}$$

The height and width are similar to equation [2.8]:

$$n_{h/w}^{[\lambda]} = \left\lfloor \frac{n_{h/w}^{[\lambda-1]} + 2p^{[\lambda]} - f^{[\lambda]}}{s^{[\lambda]}} + 1 \right\rfloor \tag{2.14}$$

Where $n_c^{[\lambda]}$ is equal to the number of filters applied in layer $\lambda - 1$ (Fig. 2.23).



4	9	2	5	8	3
5	6	2	4	0	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

Figure 2.23: Convolutional layer [49]

After convolving the image with the kernel filter, the CNN uses one of the pooling functions ϕ described above:

$$P(a^{[\lambda-1]}) = \phi^{[\lambda]}(a^{[\lambda-1]}) \tag{2.15}$$

Where ϕ is the pooling function and the dimensions are the same as in the previous layer with the only exception that $n_c^{[\lambda]} = n_c^{[\lambda-1]}$.

Typically, the pooling is applied with a stride of 2 in 2×2 patches. The average pooling produces a smooth downsampled image. Meanwhile, the max pooling is convenient for detecting sharp features, which is the reason it is more commonly used.

The output layers of a CNN are made of a classical ANN structure of three hidden layers (see subsection 2.2.1). However, the network has to make the corresponding transition from the three-dimensional structure. Firstly, the tensor of layer $\lambda - 1$ is flattened into a one-dimensional vector with the following dimensions:

$$D(a^{[\lambda-1]}) = (n_h^{[\lambda-1]} \times n_w^{[\lambda-1]} \times n_d^{[\lambda-1]}, 1) \tag{2.16}$$

Consequently, the following calculation is performed:

$$z_i^{[\lambda^*]} = \sum_{j=1}^{n_{\lambda^*-1}} w_{i,j}^{[\lambda^*]} a_j^{[\lambda^*-1]} + b_i^{[\lambda^*]} \rightarrow a_i^{[\lambda^*]} = \phi^{[\lambda^*]}(z_i^{[\lambda^*]}) \tag{2.17}$$

where λ^* denotes a one-dimensional layer, $z_i^{[\lambda^*]}$ is the weighted sum of the inputs of the i-th perceptron in layerλ^*, n_{λ^*-1} is the number of perceptron outputs of the precedent layer, and $w_{i,j}^{[\lambda^*]}$ the weights between the node i in the layer λ and the node j in the layer $\lambda - 1$.

2.4.4 Variations of CNN architecture

2.4.4.1 R-CNN

CNNs represent a good starting point for the feature extraction and characterization of images. But their behavior is much different from human perception. As explained previously, there is a trade-off between the image generalization and object location associated to the depth of the layers. Another limitation is that changing the spatial location of the objects in images may adversely affect the predictions.

For detecting and classifying several objects in the image, R-CNN (*Region-based Convolutional Neural Networks*) are generally used (Fig. 2.24) [50].

These networks work with the concept of RoI (*Regions-of-Interest*), which are frames that the network marks as potentially likely to contain an object. They are calculated within the image using an algorithm called *selective search*. Each frame is also known as a *bounding box*. The core idea consists in training the network on images that contain bounding boxes.

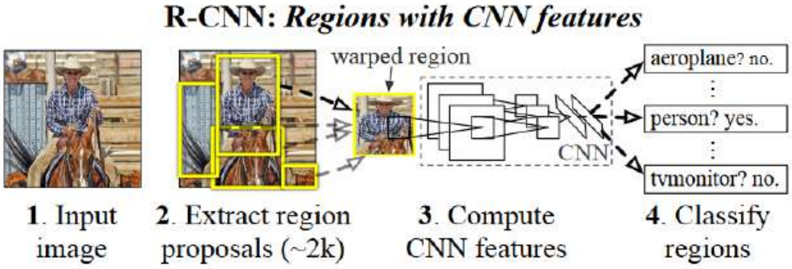


Figure 2.24: R-CNN algorithm [50]

The method starts by proposing and extracting 2000 RoIs of different sizes and locations. Starting with the initial segmentation, the goal of the selective search is to perform subsegmentations recursively combining similar regions (in term of object structures) (Fig. 2.25).

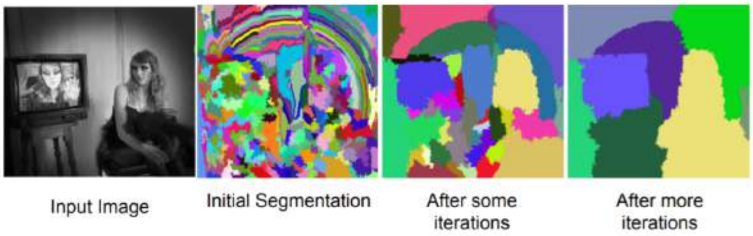


Figure 2.25: Example of recursive steps in the selective search algorithm [50]

Each proposed region is then warped to a fixed size, and passed through a CNN to determine the feature maps of each RoI. The ANN architecture at the output layers is preceded by a class predictor and a regressor. The class predictor is an SVM classifier (*Support Vector Machine*) which predicts the scores for each of the classes, whereas the regressor has four real-valued outputs: the x and y coordinates of the bounding boxes' leftmost upper point, their height and width.

2.4.4.2 Fast R-CNN

Basic R-CNN is very slow at training because the network has to learn the parameters from 2000 feature maps corresponding to each one of the RoIs, which makes its application infeasible in practice [51]. Fast R-CNN (Fig. 2.26) is the first attempt to increase the computational efficiency of R-CNN.

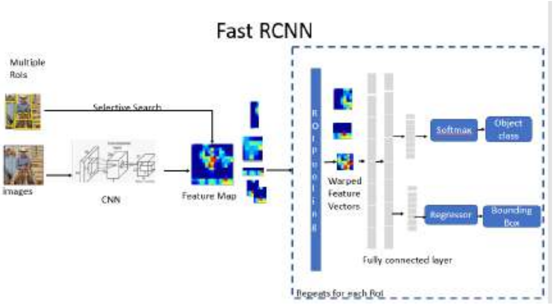


Figure 2.26: Structure of the Fast-RCNN network

In Fast R-CNN, the image is processed with several convolutional and max pooling layers to produce a feature map. Afterwards, selective search generates a number of object proposals from the feature map, each of which is warped into a fixed-length feature vector. Every feature vector is then fed into a sequence of fully connected layers that finally give two output layers: one that presents the probability estimates for every detected object instance and the values denoting its bounding box.

2.4.4.3 Faster R-CNN and Mask R-CNN

Both R-CNN and Fast R-CNN use selective search to produce region proposals. In Fast R-CNN, this process generally takes about two seconds per image, which is considerably better compared to R-CNN. However, the real-life applications require even shorter processing times.

In addition, the selective search algorithm does not have the ability to detect objects within regions since it simply combines the pixels into regions based on the low-level features that they form [52]. Faster R-CNN uses *Region Proposal Network* (RPN) instead of selective search [53]. The idea behind RPN is to provide a mechanism for learning the detection of objects.

In Faster R-CNN, the feature map is obtained by several convolutional and pooling layers, as in the previous implementations. Subsequently, the map is divided into windows each of which is fed to the RPN. RPN maps each window to a lower-dimensional feature and passes it to two fully connected layers that generate k anchor boxes of different shape and sizes for each window (Fig. 2.27). For every anchor the RPN predicts two things: the probability of the anchor containing an object (regardless of the class the object belongs to), and the bounding box for adjusting the anchor to fit the object.

In this sense, only the anchors of positive detections, with non-repeated objects, are then passed to the input of the RoI-pooling layer, which has the same functionality as in the Fast R-CNN architecture. This also applies to the fully-connected ANN output layer which is responsible for predicting the class of selected objects and the bounding box location in the original image.

As a result of this upgrade, Fast R-CNN the speed of object prediction has increased by 10 times in relation to Fast-RCNN.

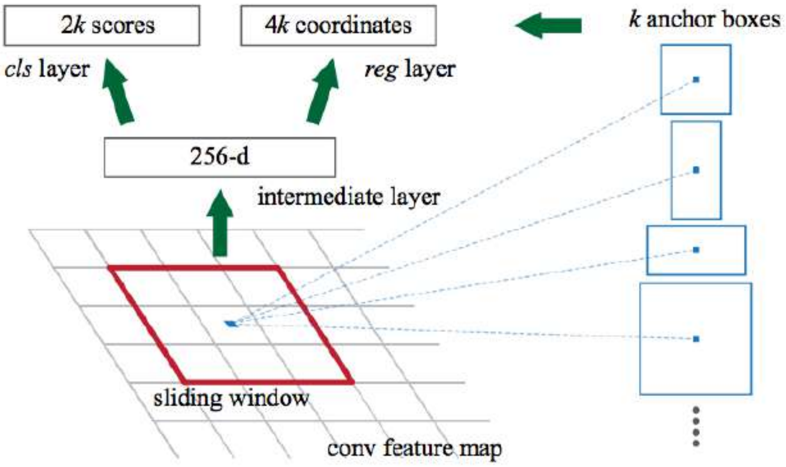


Figure 2.27: Sliding a window over the feature map in an RPN network [53]

Mask R-CNN is the last advancement of the computer vision technology [54]. It is the natural extension of Faster R-CNN since the underlying architecture is the same. The main advantage of Mask R-CNN consists in its ability to provide masks for each object on top of the classification and bounding box.

Image segmentation can be presented in two main types: semantic and instance segmentation. Semantic segmentation classifies multiple objects of the same category as one object, i.e., they are highlighted in the image with one color and name. Meanwhile, instance segmentation is a more complicated task. The latter classifies each object in an image as a separate instance, simultaneously not differentiating those that fall under the same category. The goal of Mask R-CNN is to implement this technology, where each pixel is assigned a category.

Additionally, Mask R-CNN is capable of pixel-to-pixel alignment between input and output due to using *RoIAlign* layer instead of *RoIPool* as in Fast R-CNN and Faster R-CNN. *RoIPool* lacked in high pixel accuracy since it operated on a grid of sub-windows that didn't precisely fit on RoIs. Although, it didn't affect the overall classification precision, it was unfit for instance segmentation. As a result, the researchers devised *RoIAlign* that is capable to create a more accurate grid of windows on the image, and, as a consequence, achieve pixel accuracy.

In conclusion, Mask R-CNN is the state-of-the-art of computer vision that is capable of correctly detecting objects in images in a short period of time. Thus, it was decided to employ this technology for the purpose of this project.

2.5 Drones

Drone is an aerial vehicle with no pilot, crew or passengers on board controlled remotely or through a predetermined mission program. Also referred to as Unmanned Aerial Vehicle (UAV), drones are part of Unmanned Aerial System (UAS) that also include the team managing flight controls and the connection equipment. If the flight control is carried out through a remote controller, this is a case of Remotely Piloted Aerial System (RPAS). Whereas, if the mission has already been installed in the hardware, it is an Autonomous Operation (AO) [55].

Drones are used for multiple purposes that, among others, include recreational, surveillance, military, competitive and agricultural. Their application in agriculture is of particular interest for this project and, perhaps, the most beneficial for humanity.

2.5.1 Drones in agriculture

Agricultural UAVs can monitor crop fields, plant seeds, and spray fertilizers and pesticides. Until recently such tasks were developed by expensive heavy machinery (airplanes and cars) that require considerable funds for the fuel. In this sense, UAV technology has been an important breakthrough in this field since it has reduced the cost of these tasks and has streamlined the involved processes. Accordingly, the use of UAVs in agriculture is gaining popularity every year.

As an example, in 2019 Switzerland has become the first European nation to employ UAVs for spraying operations even though this activity was banned in European Union in 2009 [56]. Now, up to 60 vineyards of Aargau, Zurich and Thurgau use this technology.

Drones can fly at low speeds which is ideal for *precision farming* [57]. The term refers to a management approach based on real-time observations of crops. It allows the farmers to achieve higher crop yields while reducing the costs. Moreover, some experts argue that drones can perform certain tasks 5 times faster than the heavy machinery, while reducing working hours for farmers, contributing to the quality of crop yields and minimizing environmental impact [57]. According to [58] the use of UAVs in the agriculture sector includes:

1. **Soil and field analysis:** While flying over fields, UAV can get three-dimensional image maps from video cameras, and use image computer analysis to determine, for example, irregularities in the crop field or its water needs.
2. **Planting:** By adding customized mechanical elements, UAVs can carry pods with seeds and nutrients that can be thrown away from the air to sow the ground.
3. **Crop spraying:** By employing lasers and ultrasonic sensors UAVs can adapt its height to the ideal distance for spraying the chemicals or water the plants.

4. **Monitoring:** Vast crop fields require monitoring for ensuring the yields. Previously, farmers used expensive methods based on the analysis of images obtained either by a satellite or manned aircraft. Both approaches have considerable drawbacks. For instance, satellite images are expensive. However, drones can provide high-altitude images in real-time and at a reduced cost [59].
5. **Thermal imaging:** Drones can capture infrared radiation emitted by the crops to identify problems caused by climate changes, weeds, pests and diseases, improper irrigation and others [60]. An example of such an image is represented in Fig. 2.28.

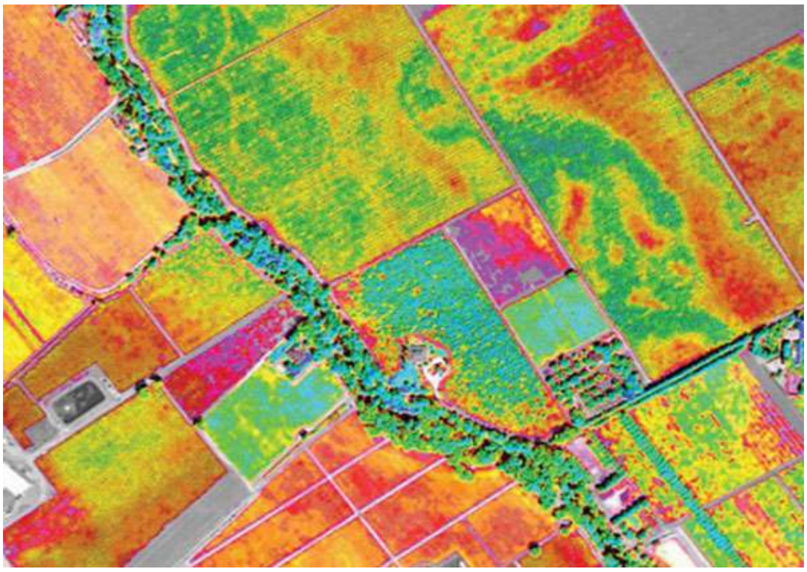


Figure 2.28: Image of infrared radiation emitted by crops [60]

Today, UAV images are combined with image analysis using Machine Learning techniques. In [61], UAS performs an agricultural mapping in Dubai by means of neural networks used for image vision analysis. More precisely, Pytorch is used to train a neural network to identify ghaf trees and date palms. The results showed that this method can efficiently provide up-to-date quality data.

2.5.2 Regulation

UAV flights are regulated in Spain by both EASA (European Aviation Safety Agency) and AESA (Agencia Estatal de Seguridad Area) [62, 63]. EASA is in charge of establishing the main flight regulations for UAVs for the 31 members of the European Union, while AESA ensures compliance with these regulations in Spain.

Currently, there are two main regulation documents of reference written by the EASA that set these rules in Europe - *Execution Regulation (UE) 2019/945* and *Execution Regulation (UE) 2019/947*. The first specifies the following

1. Types of UAVs whose design, production, and maintaining are subject to certification
2. Requisites for design and fabrication of UAVs
3. Rules for UAV operators
4. Commercialization rules

The second is related to the degrees and expertise of the pilots required to manage UAVs in the different sectors. In addition, it defines the following categories:

- **Free:** Low risk, no authorization or declaration necessary.
- **Specific:** Medium risk, a declaration or authorization is necessary in standard scenarios.
- **Certified:** High risk, regulated in a way similar to traditional manned aviation.

2.6 Alternative to UAV and Deep Learning

As mentioned in the first chapter, another solution for monitoring the crop fields can be an IoT system similar to the one developed in [2]. A “smart platform” comprised of a single sensor node, a gateway, a server application and a web page would be able to provide the most recent data about the state of crops and actuate the irrigation valves, thus completely eliminating the necessity for people to be transported to the crops during the monitoring process. In addition, the platform would help to detect other problems such as pests and bad weather conditions.

3 Data Preparation and Programming Mask R-CNN

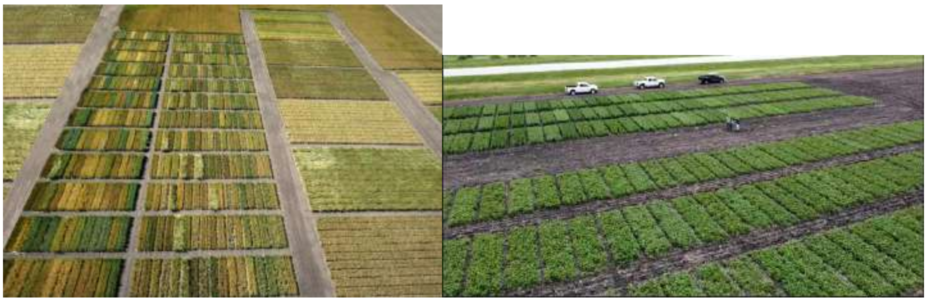
3.1 Dataset preparation

Neural networks are some of the most data-intensive classification models used in the field of ML. They are part of supervised ML algorithms that, in general, require big amounts of data for the training and estimating the precision accuracy with reduced variance. Furthermore, the learning does not stop when the model is applied in practice and new data obtained in real tests is used to improve the results. As such, preparing the data is a key point in the development of supervised ML algorithms.

Creating the necessary dataset required a study of the potato crop to understand when exactly to take pictures. This process also involved a thorough familiarization with the functioning and programming of the employed UAV to avoid any kind of damage to the drone or other objects. Taking pictures was a continuous process that required certain planning. Furthermore, every data sample utilized in the training had to go through the process of labeling and separation into different datasets.

3.1.1 Image acquisition and drone usage

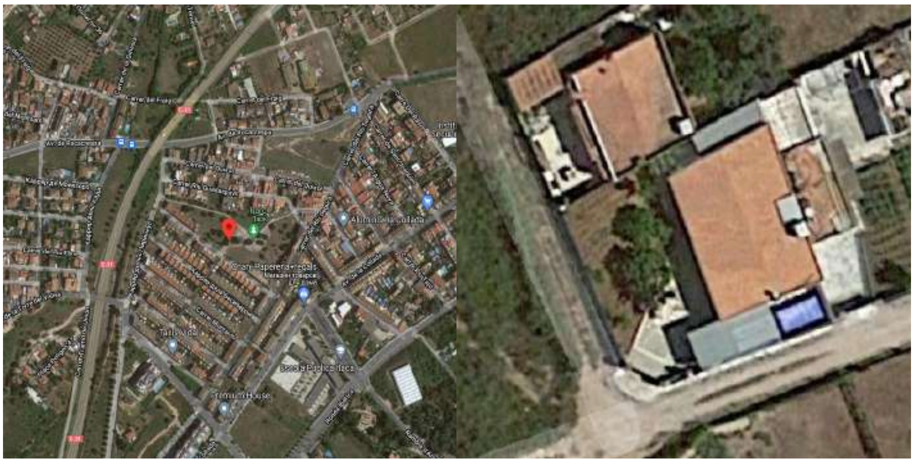
The challenge of this work consisted in training a neural network to analyze UAV images of crop fields, detect the crops present on them and determine their growth stage. Accordingly, photographs had to be taken with the help of a UAV in different conditions and growth stages of the plant. To achieve a proper generalization of the model, the dataset had to consist of photo samples from a wide variety of positions, including different heights, light conditions, and in different days to catch the features of the various growth stages.



(a) Wheat crops [64] **(b)** Soybean crops [65]

Figure 3.1: UAV pictures of various crop fields

However, there was an issue that had to do with flight regulations. Flying UAVs and taking pictures of private crop fields, similar to the ones in Fig. 3.1, is illegal without permission. After several attempts to find support from farmers willing to contribute to the goal, in the end it was only possible to take photographs of potato plants in a small orchard located on the outskirts of Vilanova i la Geltrú. More precisely, in La Collada, a north-west neighborhood area in Vilanova (Fig. 3.2).



(a) Collada neighborhood **(b)** Crop field

Figure 3.2: Terrestrial conditions

In addition, this site presents certain difficulties not only due to the small cultivated area or weather flight conditions, but also to the surroundings. With many small landowners and a highway nearby the task was complicated even further. Therefore, the use of the UAV has been kept to a minimum by performing fewer short duration flights with a maximum height of 3m, complementing aerial photographs with pictures occasionally taken with a phone camera.

Assuming these limitations, flights were configured with the *Mission Planner* open-source software with a total of 7 waypoints, that established the take-off, landing and various points for taking images. *Mission planner* can act as a ground control station for planes, copters, rovers and a wide variety of radio-controlled vehicles [9]. The waypoints are point references in terms of GPS coordinates to which aerial maneuvers can be assigned. Maneuvers such as taking-off, landing and loiter among others. It also allows to control the gimball to focus the camera on a certain part of the crop field.

Initially, a total of 5 flights have been conceived. However, due to the time and weather limitations, only three were carried out. In each flight the aircraft takes off at WP1 and the controller adjusts the gimball to focus the camera on WP2 (Fig. 3.3). Afterwards, the aircraft moves to WP3 where it enters the loiter mode. The loiter time has been configured between 15 seconds a 2 minutes for different flights. The same is repeated for WP4, WP5 and WP6. After the loiter time of WP6 has ended, the aircraft moves to WP7 where it lands. This mission was sent from a computer to the ArduPilot controller through the USB connection.

The video camera model is a GoPro Hero 4, which can be controlled from a mobile phone or any other hardware. It is easy to immediately download the files from the camera to later use them for training the Mask R-CNN. In this case, the mobile phone was connected to the GoPro via wireless WiFi connection and the pictures were taken at different moments with different zoom and aspect ratio options [66].

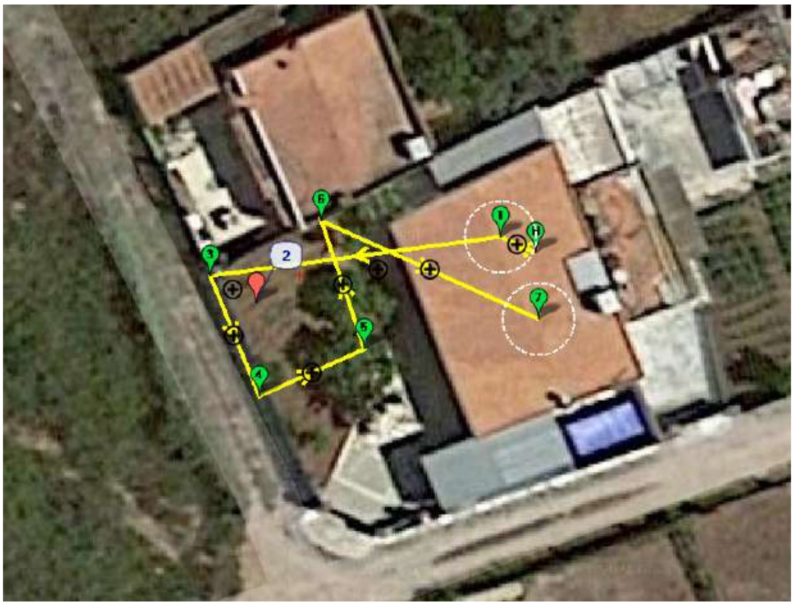
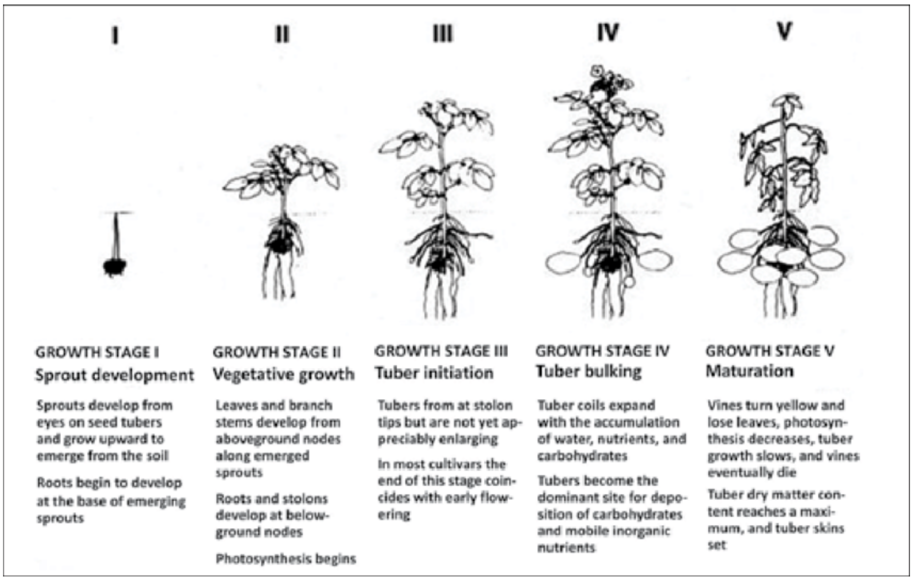


Figure 3.3: Programmed mission

3.1.2 Experiment configuration

3.1.2.1 Potato growth

Class labels are set according to the different growth stages of the potato plant.



GROWTH STAGE I
Sprout development

Sprouts develop from eyes on seed tubers and grow upward to emerge from the soil

Roots begin to develop at the base of emerging sprouts

GROWTH STAGE II
Vegetative growth

Leaves and branch stems develop from aboveground nodes along emerged sprouts

Roots and stolons develop at below-ground nodes

Photosynthesis begins

GROWTH STAGE III
Tuber initiation

Tubers from at stolon tips but are not yet appreciably enlarging

In most cultivars the end of this stage coincides with early flowering

GROWTH STAGE IV
Tuber bulking

Tuber coils expand with the accumulation of water, nutrients, and carbohydrates

Tubers become the dominant site for deposition of carbohydrates and mobile inorganic nutrients

GROWTH STAGE V
Maturation

Vines turn yellow and lose leaves, photosynthesis decreases, tuber growth slows, and vines eventually die

Tuber dry matter content reaches a maximum, and tuber skins set

Figure 3.4: Different growth stages of potato [67]

The potato plant has a relatively short life, going from 80 to 150 days since sowing to maturity, with differences among species. Its stages are often described in terms of tuberization and tuber development (Fig. 3.4). The tuber cycle is characterized by a period of initiation and growth, followed by a dormant period and finally the shoots sprouting in the next vegetative generation [67, 68].

After planting a potato, the growth initiation, preceded by the dormancy, is accompanied with substantial increases in cellular metabolism. This stage finishes with shoots appearing from the eyes of the primary tuber until they start appearing on the surface. During the growth, all vegetative parts (leaves, branches, and stolons) are formed. These two stages last from 30 to 70 days depending on several factors among which are: planting date, temperature soil and other environmental factors such as age of tubers and particular characteristics of crop conditions.

Approximately 30 to 60 days after planting the seed, the tuber formation begins. The tubers appear from lateral underground shoots developing at the base of the main stem kept underground, which turn into stolons due to diagravitropical growth. When conditions are favorable for tuber initiation, the elongation of stolons stops and cells located in the pith and cortex of the apical region first enlarge and then divide longitudinally. The combination of these processes results in swelling of the subapical part of the stolon.

During the enlargement of stolons of the third stage, tubers become the major sink for the potato plant, storing massive amounts of carbohydrates (mainly starch) and also significant amounts of protein. Their overall metabolic activity decreases and they behave as storage sinks until the plant saturation is reached. At this point, the plant leaves turn yellow and then fall. Photosynthesis and tuber growth slows, and vines eventually die. The dry matter content of the tuber reaches a maximum and the tuber skin sets. This is when the potatoes can be harvested.

3.1.3 Labeling

Taking into account how stages develop in this plant, it seems reasonable to set five different classes:

1. Sprout development (pat1)

2. Vegetative growth (pat2)

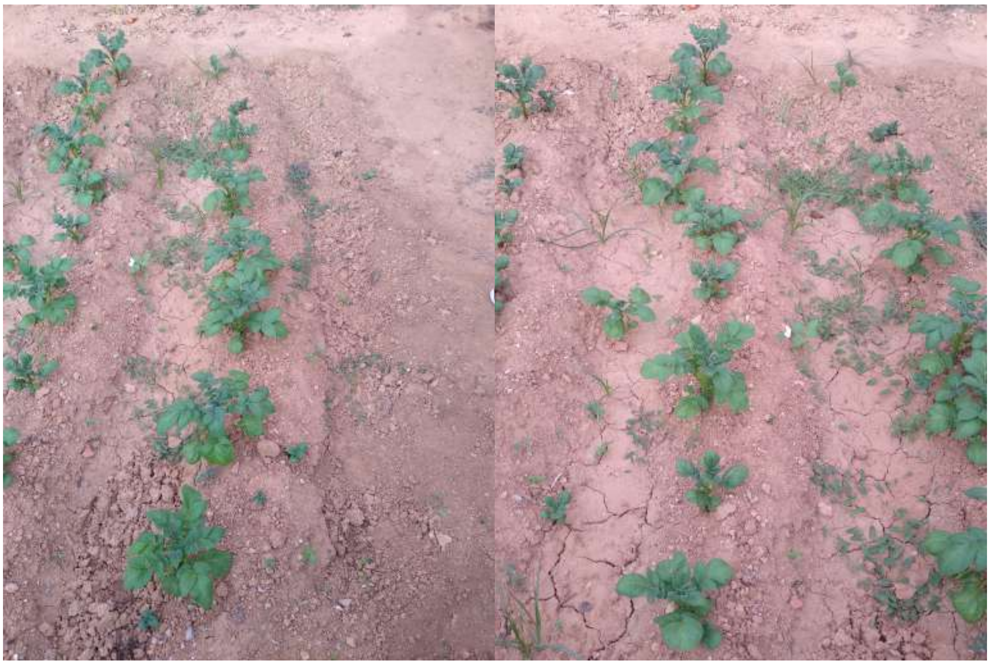
3. Tuber initiation (pat3)

4. Tuber bulking (pat4)

5. Maturation (pat 5)

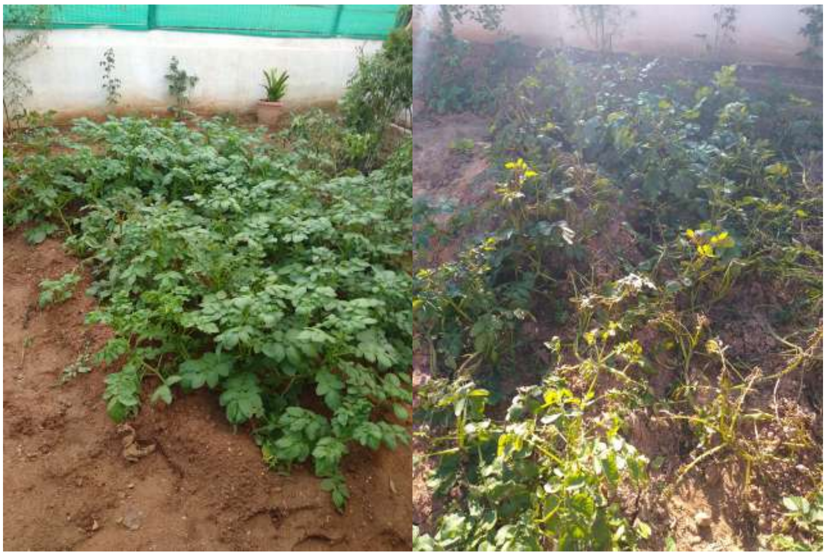
Fig. 3.5 and 3.6 show aerial and ground pictures obtained during different stages of the potato growth which have been labeled accordingly, so that Mask-RCNN can be trained. Separate crops can easily be identified by the human eye in the first three stages. The first stage lacks visible signs of growth, whereas the second and third differ in height and the amount of branches and leaves. The fourth is the highest of all with occasional flowers on top typical of the bulking stage. Finally, the fifth shows clear signs of maturation since the branches loose their rigidity.

It is clear that potato crop fields in stage 4 are too thick with leaves to detect separate crops (Sub-Fig. 3.6a). Likewise, in stage 5 the crops are too wilted to be identified separately as well (Sub-Fig.3.6b). Thus, such pictures only seem useful for evaluating the stage of growth of entire fields or rows of crops.



(a) First example of stages 1, 2 and 3 **(b)** Second example of stages 1, 2 and 3

Figure 3.5: Samples of the acquired dataset (I)



(a) Stage 4 **(b)** Stage 5

Figure 3.6: Samples of the acquired dataset (II)

Fig. 3.7 shows an example of picture labeling with plants from stages 2 and 3. Labeling is a crucial part of supervised ML. The models learn by comparing the predicted masks with those created manually. For manual labeling of picture samples, the open source software *Labelme* from GitHub was used [5]. Labelme is an interactive tool that allows polygons to be created and assigned to a group class. When this is done, a *json* file containing the list with the pointed shapes, the coordinate points of their masks and the category they belong to is created for each one of the images.

Accordingly, each instance has a list of hand-picked "points" that surround each object. Fig. 3.7 is created by using the data from a json file in a Python script, presented in subsection 3.2.1. This is a monotonous and laborious task needed so that Mask R-CNN could learn to recognize potato crops.

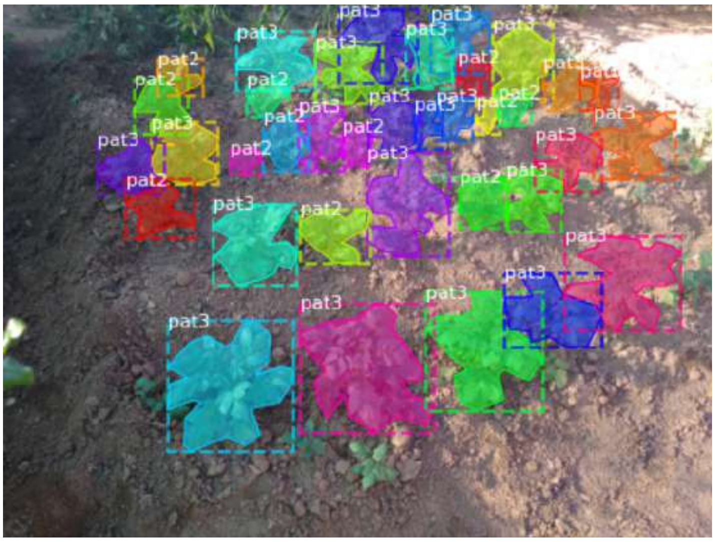


Figure 3.7: Labels created in *LabelMe*

Before the labels are made, the peculiarities of the potato growth had to be identified from the image to correctly interpret the stage of each crop. Ideally, this is a task that should be carried out by expert potato farmers. However, the complexity of the project would be unnecessarily increased along with its economic cost. For this reason, estimating potato growth is subject to bias and error as shown in the results (see chapter 4).

3.1.4 Dataset split

Generally, supervised learning requires the data to be split into two categories [69]:

1. **Train dataset** used for determining the convolutional filters, and weights, and biases of the different output layers.
2. **Test dataset** used during the inference stage to estimate the model accuracy.

Using all samples for both the training and test would lead to very optimistic detections because the model losses generalization while training on the same dataset.

To converge to the true accuracy, the samples in the training must be different to the test set.

Thus, 80/20% set splits are generally used to emulate a real experiment. In general, the true estimated accuracy is much lower to that estimated from the training set, because the model needs to learn from new samples. This is the reason why supervised learning requires huge datasets. When large datasets are not available, test-split leads to wrong estimates and other approaches such as *k-fold cross validation* should be used instead [70].

Additionally, the model needs a way of knowing whether it is "doing good" during the training stage and modify the training process or even stopping it when it's not the case. To avoid the latter, it is important to detect situations when the model starts to loose accuracy on the test dataset. When that happens, the recent calculated weights should be omitted and the learning rate η modified to calculate new weights. In ML this process is referred too as "regularization".

To provide regularization, a third set, known as *validation* is prepared from the input data. When splitting a dataset, a common approach is to use 75% (training), 15% (test) and 10% (validation). During the training, the network estimates intra-accuracies and decides whether to continue with the current updated weights or change the learning rate η. Thus, for this work a 75/15/10 split is used, which resulted in 150 images for the training dataset, 30 for test and 20 for validation. The images were taken in different resolutions some of which are 1920×1080 and 1280×960. It should also be noted that for the reasons discussed in subsection 4.5, not all the obtained images were utilized for the training.

3.2 Implementation of Mask R-CNN

Due to the big amounts of data necessary to learn different patterns, supervised learning employs considerable computational resources and processing power. The CPU hardware does not suit the required tasks since a simple ALU (Arithmetical Logic Unit) structure is not capable of configuring millions of parameters in a relatively short period of time. Instead, GPUs (Graphics Processing Unit) are used for performing such computations tasks. Very briefly, GPUs can parallelize batch processes of CNNs and complete numerous tasks simultaneously. In that regard, the convolutional products do not depend on each other and the feature maps can be obtained much faster.

GPUs contain thousands of *CUDA* processors that require large amounts of VRAM (Video Random Access Memory) during the training. This hardware is expensive and can easily cost more than several thousands of dollars [71], a price which is out of reach for this project.

Another possibility is to use cloud-base alternatives existing today worldwide. Companies such as Google, Microsoft and Amazon provide web-based frameworks to train

Deep Learning models either for free under certain restrictions or for fees providing better batch services, memory and GPU capacity.

The notebook environment developed by Google known as *Colab* allows to create Linux-based virtual machines, which can be configured to work with remote CPUs, GPUs or TPUs. Both GPUs and TPUs meet the hardware requirements for training Mask R-CNN. Colab servers also provide 12GB of RAM (Random Access Memory) to write Python code in a *Jupyter*-like interface. Considering all the above-given features, it was decided to use *Colab* for training the Mask R-CNN model.

However, this decision was limited in availability. Although the GPU environment can be used for free, depending on the utilized resources, Colab can limit the execution time of Python scripts up to several hours, which is clearly insufficient to train the models. In addition, the option of upgrading to *Collab premium* paying a fee of 10$/month is not available in Spain since September of 2021.

3.2.1 Python script

As mentioned previously, the used python script is an implementation of Mask R-CNN Python 3, *Keras* and *TensorFlow* [4]. It was written in multiple code cells in Colab that successively run one after another. Accordingly, each cell will be commented to understand the training process.

The script starts with mounting a directory in Google Drive. It is required for saving the progress and accessing the logs and data samples. Once executed, the script asks for permission after which it is possible to upload the necessary files. This step is represented by the next code box:

```
from google.colab import drive
drive.mount('/content/drive/')
```

Consequently, the folder with Mask R-CNN implementation has to be downloaded from GitHub:

```
1 !git clone https://github.com/akTwelve/Mask_RCNN
2 %cd Mask_RCNN/
3 !pip3 install -r requirements.txt
4 %cd ../../
```

With *!git clone* the script copies the “Mask R-CNN” folder in the GitHub page. The *akTwelve* is simply a branch of the *Matterport* page that solves issues that come up if Colab is used instead of a personal computer. The third line downloads some of the necessary libraries such as *NumPy*, *TensorFlow*, *ImgAug* and others. Some libraries such as *os*, *sys*, *random*, *math*, and others are predownloaded in any python development environment.

It is necessary to import the libraries used for the training and assign a name when it is convenient:

```
import os
import sys
import random
import math
import re
import time
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
import json
import imgaug
```

The configuration class that establishes the parameters necessary for the training is imported along with other files that contain the required functions:

```
ROOT_DIR = os.path.abspath(
          "/content/drive/MyDrive/TFG/Mask_RCNN/")
sys.path.append(ROOT_DIR)
from mrcnn.config import Config
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
from mrcnn.model import log
```

The *os.path.abspath()* function determines the absolute path of the root directory of the project, while *sys.path.append()* helps to find the local version of the library.

The *os.path.join()* function comes in useful for creating other directories such as *MODEL_DIR*, that will contain model parameters, and *COCO_MODEL_PATH* that will contain the pre-trained weights from the *COCO* dataset.

```
MODEL_DIR = os.path.join(ROOT_DIR, "logs")
COCO_MODEL_PATH = os.path.join(
          ROOT_DIR, "mask_rcnn_coco.h5")
if not os.path.exists(COCO_MODEL_PATH):
          utils.download_trained_weights(COCO_MODEL_PATH)
```

These COCO weights will be used only once - in the beginning, before any training has started.

The dataset directory is established on Drive in which the data is kept in a split manner described previously:

```
DATA_DIR = "/content/drive/MyDrive/TFG/FOTOS/LabelMe_JSON"
DATASET_TRAIN_DIR = os.path.join(DATA_DIR, "train")
DATASET_VAL_DIR = os.path.join(DATA_DIR, "val")
DATASET_TEST_DIR = os.path.join(DATA_DIR, "test")
```

Next, the configuration is set in the *PotatoConfig* class, which is a child class of the *Config*:

```
class PotatoConfig(Config):
        NAME = "potatos"
        IMAGES_PER_GPU = 1
        NUM_CLASSES = 1 + 3
        STEPS_PER_EPOCH = 70
        DETECTION_MIN_CONFIDENCE = 0.9
        USE_MINI_MASK = False
        IMAGE_SHAPE = [1024,1024,3]
        LEARNING_RATE = 0.0005

config = PotatoConfig()
```

The *Config* class has many parameters. Nonetheless, the ones that were overridden above have the most importance. To better understand some of them, the following definitions should be considered:

- Batch size - the number of samples that is fed to the neural network after which it updates the weights and biases.
- Epoch - the process in which the neural network is going through the entire dataset.

If there is a dataset of 50 samples (e.g., images) with a batch size of 10, it means that after one epoch the model parameters (weights and biases) were updated 5 times. Consequently, the parameters can be defined:

- NAME: Required for definition for correctly loading the dataset.
- IMAGES_PER_GPU: Number of images to train with on each GPU. The VRAM of the available GPU does not allow more than one image. Moreover, since Colab only allows to use one GPU, this number is also equal to the batch size.
- NUM_CLASSES: Number of classes defined during the labeling process. In Mask R-CNN, each region is defined as either pertaining to the background or to a user-defined class. Although there are more stage growths, for reasons that were discussed previously, the amount of possible classes is set to three plus one background.
- STEPS_PER_EPOCH: Number of images fed to the network. It can vary across the training process since more images can be acquired with time.

- DETECTION_MIN_CONFIDENCE: Defines the threshold for filtering out low confidence boxes.

- USE_MINI_MASK: Used for controlling the memory load created by the object instances, If enabled, the load is reduced. However, while testing, the masks could be correctly represented only if the value was *False*.

- IMAGE_SHAPE: The Matterport implementation of R-CNN is trained on squared images. If the fed images are not squared, they are padded with zeros to make them squared.

- LEARNING_RATE: The *README.MD* file recommends to use a learning rate less than 0.02. Accordingly, the final value used for this project has been obtained after careful testing with the error function.

A new class called *PotatoDataset* is created. It inherits from *utils.Dataset* class and defines new functions:

```
1  class PotatoDataset(utils.Dataset):
2      def load_dataset(self,dataset_dir):
3          self.add_class('potatos', 1, 'pat1')
4          self.add_class('potatos', 2, 'pat2')
5          self.add_class('potatos', 3, 'pat3')
6          for i, filename in enumerate(os.listdir(dataset_dir)):
7              annotation_file = os.path.join(dataset_dir,
8                                              filename.replace('.jpg','.json'))
9              if '.jpg' in filename and os.path.isfile(annotation_file):
10                     self.add_image('potatos',
11                                    image_id=i,
12                                    path=os.path.join(dataset_dir, filename),
13                                    annotation=annotation_file)
```

In the first place, the *load_dataset()* function adds the classes defined by the user (lines 3-8). Then, the *annotation_file* that contains the masks is defined for each image (lines 9-13). If the file that contains "*.jpg*" possesses the according annotation file, the *add_image* function appends the information about the image into the dataset (lines 10-13).

The next function is *extract_masks()*:

```
 1
 2     def extract_masks(self ,filename):
 3         json_file = os.path.join(filename)
 4         with open(json_file) as f:
 5             img_anns = json.load(f)
 6         n_masks = 0
 7         for anno in img_anns['shapes']:
 8             if anno['label']=='pat1'     or\
 9             anno['label']=='pat2'    or\
10             anno['label']=='pat3'    or\
11             anno['label']=='pat4'    or\
12             anno['label']=='pat5':
13                 n_masks+=1
14
15         masks   =   np.zeros([img_anns['imageHeight'],
16                                 img_anns['imageWidth'], n_masks],
17                                 dtype='uint8')
18         classes = []
19         i=0
20         for anno in img_anns['shapes']:
21             if anno['label']=='pat1' or anno['label']=='pat2' \
22             or anno['label']=='pat3' or anno['label']=='pat4' \
23             or anno['label']=='pat5':
24                 if anno['shape_type']=='polygon':
25                     mask = np.zeros([img_anns['imageHeight'],
26                                     img_anns['imageWidth']], dtype=np.uint8)
27                     cv2.fillPoly(mask, np.array([anno['points']],
28                                                 dtype=np.int32), 1)
29                     masks[:,:,i]=mask
30                     classes.append(self.class_names.index(anno['label']))
31                     i+=1
32         return masks,classes
```

The function calculates the amount of classes detected in the json file of an image (lines 7-13). In lines 15 - 17, the *zeros* of *NumPy* is used to create a three-dimensional matrix with the height and width of the image and the depth equal to the amount object instances. Afterwards, the *cv2.fillPolly* function is used to fill the previously created matrix with the masks' point values (lines 25-29) and the *append()* to save the masks' classes.

The two remaining classes of *PotatoDataset* are *load_mask()* and *image_reference()*:

```
    def load_mask(self, image_id):
        info = self.image_info[image_id]
        path = info['annotation']
        masks, classes = self.extract_masks(path)
        return masks, np.asarray(classes, dtype='int32')
    def image_reference(self, image_id):
        info = self.image_info[image_id]
        return info['path']
```

The *load_mask()* loads the masks of a particular image while *image_reference()* serves for obtaining the path to the image.

Once the PotatoDataset class has been created, the dataset directories can be prepared:

```
dataset_train = PotatoDataset()
dataset_train.load_dataset(DATASET_TRAIN_DIR)
dataset_train.prepare()

dataset_val = PotatoDataset()
dataset_val.load_dataset(DATASET_VAL_DIR)
dataset_val.prepare()

dataset_test = otatoDataset()
dataset_test.load_dataset(DATASET_TEST_DIR)
dataset_test.prepare()
```

The *prepare()* function prepares the data by defining certain parameters such as the number of instance classes, their identification numbers, names, number of images and others.

Once the configuration is set, the necessary classes are created and the dataset is prepared, the model can be established:

```
model = modellib.MaskRCNN(mode="training", config=config, model_dir=MODEL_DIR)
```

Where the mode can be either “training or “inference”. Consequently, the model can load the weights:

```
init_with = "last"

if init_with == "coco":
    model.load_weights(COCO_MODEL_PATH, by_name=True,
                       exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
                                "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":
    model.load_weights(model.find_last(),by_name=True)
```

As mentioned previously, if the model is only starting to be developed for the first time, the COCO pre-trained weights are used, in which case *init_with* should be *“coco”*. Otherwise, if the training already has at least one fully completed epoch, *“init_with”* should be *“last”*.

Finally, the training can start:

```
Potato_augmentation = imgaug.augmenters.Sometimes(0.5,
    [imgaug.augmenters.geometric.Affine(rotate=(-360,360))])

model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=150,
            layers='heads',augmentation = Potato_augmentation)
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE/10,
            epochs=150,
            layers="all")
```

At the beginning, the dataset is augmented artificially by rotating the images (lines 1-2). The images are rotated around the center of the image, where *0.5* is used to indicate how often it happens (once every two images, in this case) and *(-360,360)* is the expected value range for rotation.

The model is trained with the *train()* function (lines 4 and 8). The first and second argument are the training and validation directory respectively. The third is the learning rate defined in *config* previously. The epochs, i.e., the total number of images is the fourth. The *"layers"* argument indicates what layers are trained exactly. The creators of this implementation recommend to start the process by training only the randomly initialized layers (the ones that don't have pre-trained weights from COCO). Thus, the *"heads"* is passed as a parameter. Afterwards, the training is happening at a smaller learning rate to fine tune all the layers. Accordingly, *"all"* is passed as a parameter.

4 Results

4.1 Overview

The CNN models obtained after multiple training epochs can be evaluated in several ways: loss plots and confusion matrix. Accordingly, several loss plots will be presented along with the confusion matrix for the final epoch. Additionally, the evolution of the obtained model will be shown by performing the detection on several images with parameters obtained in different epochs. To sum up, the obtained results will be discussed.

4.2 Training evaluation

The Python scripts used in this work permit to keep track of the training stage by storing the loss function parameters of the gradient descent. Unlike the classical NN, the loss function of Mask R-CNN contains three parameters: classification loss ($\mathcal{L}_{class}$), the bounding box loss ($\mathcal{L}_{box}$) and the mask loss ($\mathcal{L}_{mask}$). The first two are derived from the RPN class predictor and box regressor (see subsection 2.4.4.3). $\mathcal{L}_{class}$ represents the loss cost of detecting an object, whereas $\mathcal{L}_{box}$ is calculated only over detected objects and represents the difference offset between the box predicted by Mask R-CNN and original box. $\mathcal{L}_{mask}$ is associated to the fully connected part of the network and is computed as an average binary entropy loss obtained by applying a pixel-per-pixel sigmoid with the *k-th* class associated to the original ground-truth box.

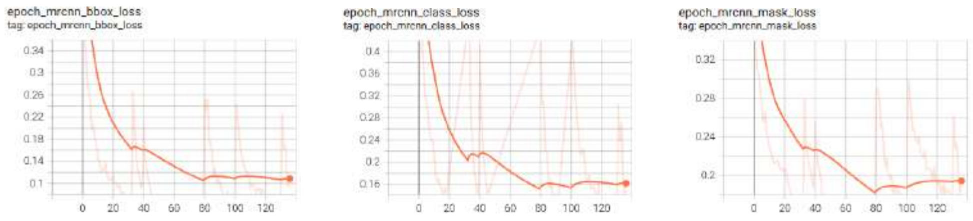
The plots that characterize these losses are obtained with the help of the *TensorBoard* visualization kit that is a part of the *TensorFlow* library [72]. *TensorBoard* is integrated within Google Colab, which is very convenient for the testing process. The following code has been used to obtain the plots:

```
import tensorflow.compat.v1 as tf
%load_ext tensorboard
sess = tf.Session()
path_logs = "/content/drive/MyDrive/TFG/Mask_RCNN/logs/potatos20211126T2055/"
file_writer = tf.summary.FileWriter(path_logs, sess.graph)
%tensorboard --logdir /content/drive/MyDrive/TFG/Mask_RCNN/logs/potatos20211126T2055/
```

In general, all three parameters have a decaying profile with discontinuities associated to the minimization of the overall loss through gradient descent, which is calculated as:

$$\mathcal{L}_{total} = \mathcal{L}_{class} + \mathcal{L}_{box} + \mathcal{L}_{mask}$$

If the loss suddenly starts to increase, it's a sign that some parameters have to be changed or the data have to be updated. The obtained plots are represented in Fig. 4.1:



(a) Bounding box loss plot **(b)** Class loss plot **(c)** Mask loss plot

Figure 4.1: Loss plots

As shown, the network learns the most in the range of 1 - 30 epochs. Next, the loss reduction slows down reaching its minimal value around epoch 80. Finally, the network starts to fluctuate gradually reaching its plateau loss value after the epoch 120. After the epoch 80, the loss starts to increase slightly. Although it might seem as an indication to change something, the overall precision is still higher after the epoch 120 as it will be seen in the confusion matrices. However, it's hard to ignore the fact that the network is learning poorly after the epoch 80. As a result, the decision was taken to continue the training by adding more data samples as they were acquired and a total of 137 epochs was completed in the end.

4.3 Inference

Inference is normally evaluated by computing the confusion matrix. Confusion matrices present the most clear way to understand how well the network performs assigning the different classes of the experiment. Inference is calculated by passing the images from the test set and calculating the outputs. First, it is necessary to create a new configuration for the model:

```
class InferenceConfig(PotatoConfig):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
inference_config = InferenceConfig()

model = modellib.MaskRCNN(mode = "inference",config = inference_config,model_dir=MODEL_DIR)
model.load_weights(model_path, by_name=True)
```

The operating mode is "inference" because the model is now supposed to simply detect object instances instead of performing forward and backpropagation. For this step, the direction to the recently obtained model parameters is found with *model.find_last()*. If there is a need to use the previously obtained parameters, the *model_path* can be adjusted. Combining the above-given code with the script from [73], the confusion matrix is obtained:

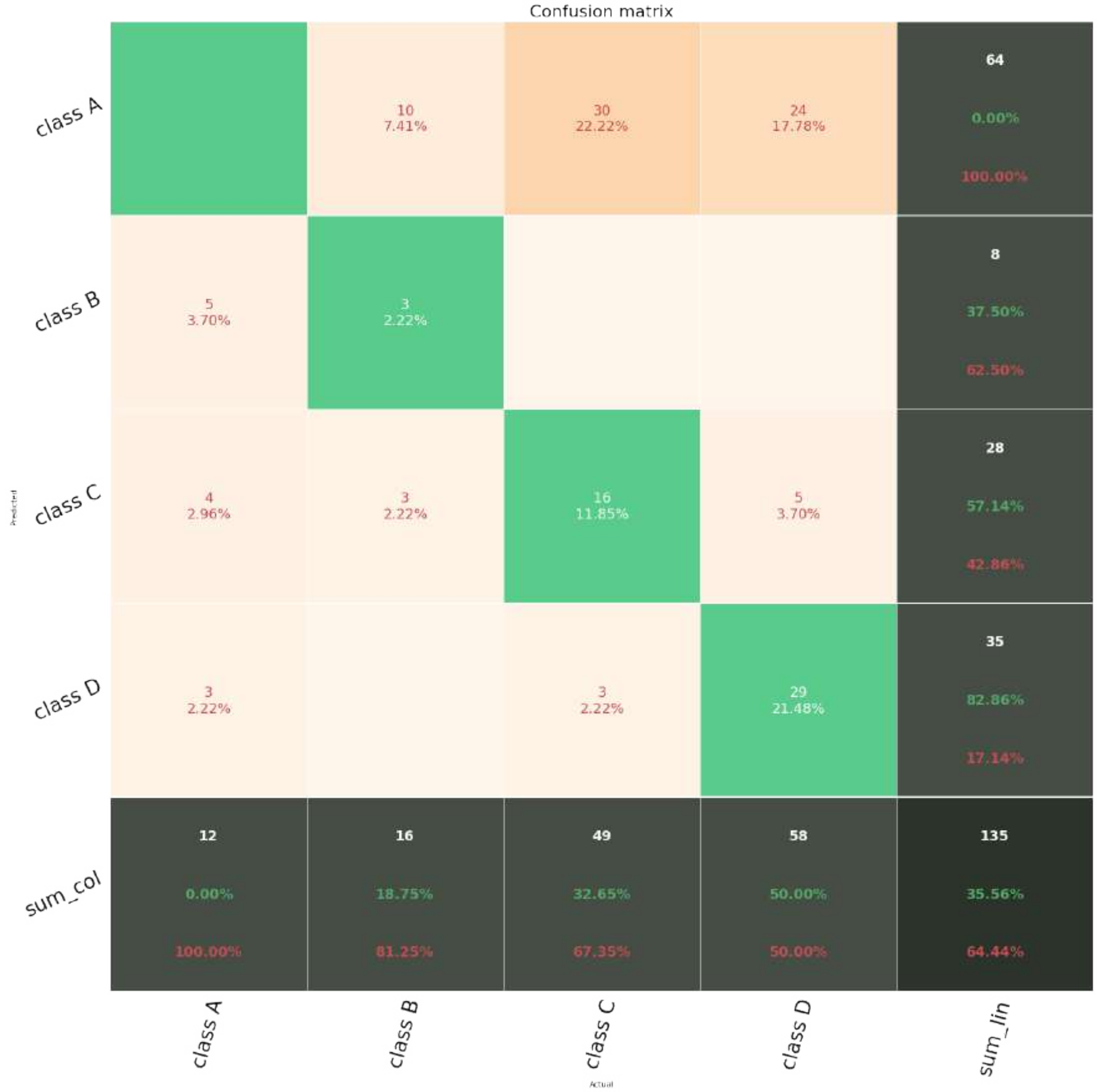
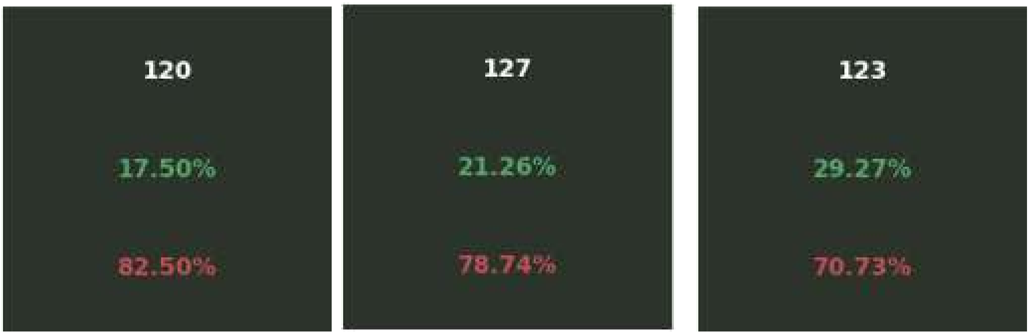


Figure 4.2: Final confusion matrix

Fig. 4.2 shows the accuracy results after 137 epochs of training. As mentioned previously (see subsection 3.1.3), potatoes can be identified separately only in the first three classes, which makes them the only classes that can be used during the labeling process. Hence, classes B to D corresponds to these three stages of potato growth. Class A is the background, which is not associated to any potato class.

In the matrix, the rows correspond to the network predictions and columns correspond to the true instance classes. The main diagonal in green specifies the correct classifications of each class. As such, the first column (left) indicates background areas which have been predicted to contain potatoes at some growth stage (from B to D), whereas the first row (top) indicates the potato plants that have been identified as the background. In a similar way, the cells not contained in the main diagonal specify the mistakes made by the network. Global results are presented at the bottom right corner of the matrix.

Fig. 4.3 shows that between epoch 40 and epoch 120, the model gained more than 10 % of precision. The last trained epoch in Fig. 4.2 shows 35.56 % of precision and detects more object instances.



120

17.50%

82.50%

(**a**) Epoch 40

127

21.26%

78.74%

(**b**) Epoch 80

123

29.27%

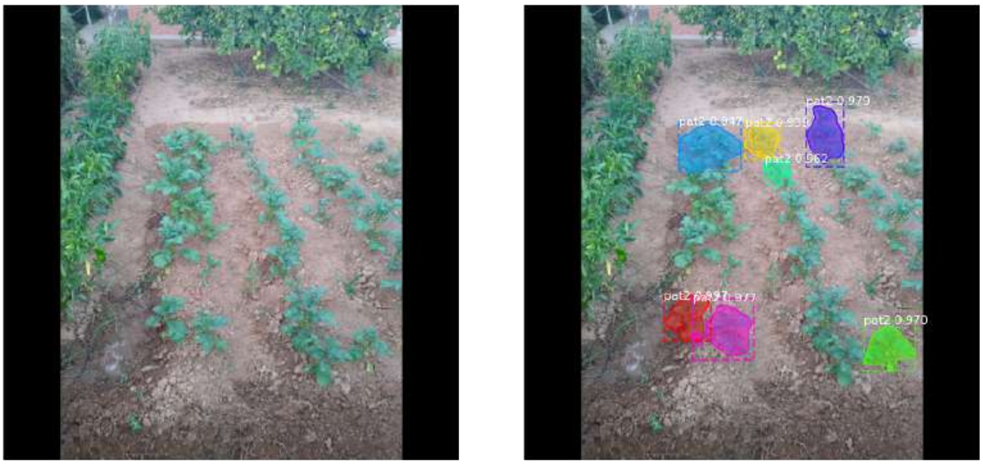
70.73%

(**c**) Epoch 120

Figure 4.3: Overall confusion matrix precision for different epochs

4.4 Mask examples

Another way to illustrate the results consists in showing how the network predicts masks over images. Fig. 4.4 shows the progression of Mask R-CNN detecting the plants at epochs 40, 80, 120, and 137.



(a) Epoch 40

(b) Epoch 80

Figure 4.4: Inference examples in the early stages of the training

It can be observed that the number of detected masks increases with training. At epoch 40 (Fig. 4.4a) the model has not yet gained enough confidence to identify any object, whereas at epoch 80 (Fig. 4.4b) it identified correctly some plants with class *pat2*.

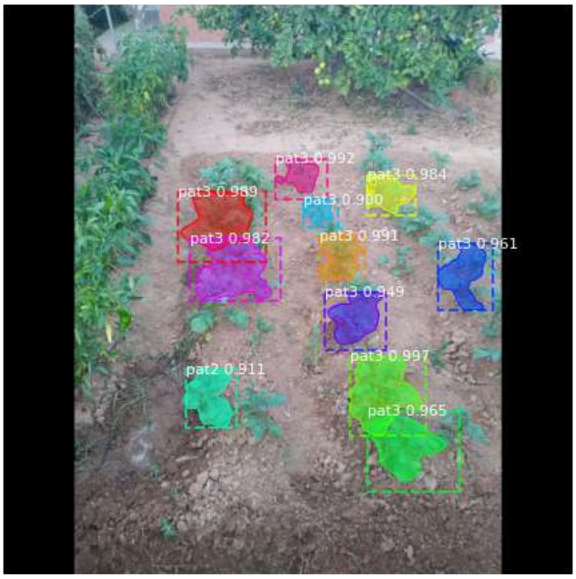
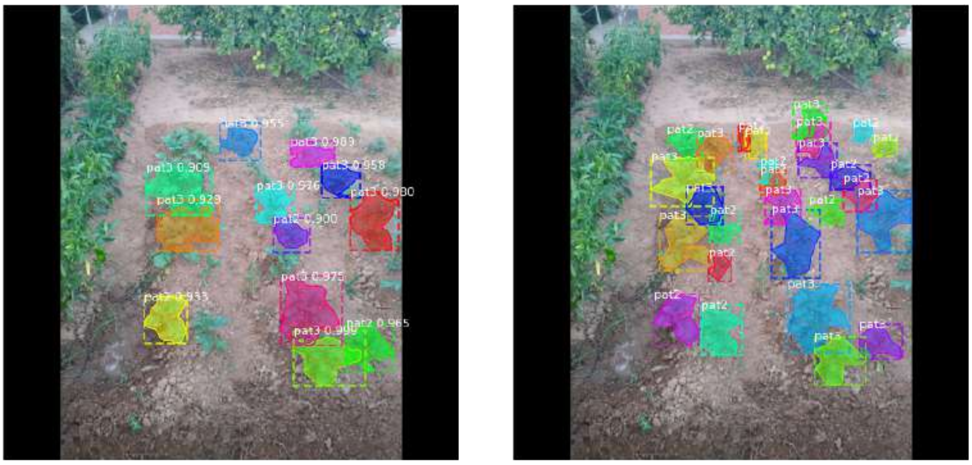


Figure 4.5: Inference at Epoch 120

By epoch 120 (Fig. 4.5) the model becomes significantly more confident in detecting crops than in the early stages. There are less masks that contain several crops and the overall class precision has increased as well. It should also be noted that the model does not confuse potato crops with other greenery.



(a) Epoch 137 **(b)** Real masks

Figure 4.6: Comparison between the real masks and final inference

By the end of the training the model is capable of detecting most of the plants in the same image (Fig. 4.6). Although the accuracy could be higher, the model manages to locate and predict multiple plants in the image despite the complexity of shapes and the number of instances.

To perform the inferences, the configuration has to be adjusted as in the subsection 4.3. First, the real masks are obtained for an image:

```
image_id = 13
original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    modellib.load_image_gt(dataset_test, inference_config,
                           image_id)

log("original_image", original_image)
log("image_meta", image_meta)
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)

visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id,
                            dataset_train.class_names, figsize=(8, 8))
```

Afterwards, the inference is launched with the method *model.detect:*

```
results = model.detect([original_image], verbose=1)

r = results[0]
visualize.display_instances(original_image, r['rois'], r['masks'], r['class_ids'],
                            dataset_val.class_names, r['scores'], ax=get_ax())
print(r['class_ids'].shape)
print(r["class_ids"])
```

The first index of the tuple *results* contain RoIs, masks, class_ids and scores for each of the detected objects in the image.

4.5 Discussion

In general, the model shows a good prediction capability for tuber initiation (Stage 3), which accuracy is 83%. Unfortunately, the performance is worse with other classes. There may be several reasons for this. Some may be due to the unforeseen circumstances and the complexity of the conceived approach and objectives, and others may be due to the lack of a correct planning attributed to little experience of both the author and director of this work.

Collecting the data and preparing it correctly is crucial for obtaining higher model precision. Crops had to be observed every week, perhaps even every day, to take multiple pictures across the entire span of the project. In practice, the crop field couldn't be supervised and monitored often enough and the pictures were taken only when the circumstances would allow it.

Additionally, the growth process is sporadic at the early stages. Some plants would grow twice as fast as others, depending on earth, climate and watering conditions. In connection with that, the time to take photos at the early stages (Stage I and II) was missed, which may have conditioned the capability of the model to predict these stages. As a result, the network was trained more on some instances classes than others, as it has happened with class D.

The crops have complex shapes and irregularities when observed from close distances. The labels of certain crops would contain more than 40 points, which is considerably more than what would be required for a crop field, a car or a person. As a result, the model lacked confidence to identify crops correctly and instead identified them as part of the background. Of the 135 object instances that were present in the pictures, 64 were identified as background. Additionally, potato crops in the stage of vegetative growth often look similar to those of the tuber initiation. Sometimes the plant is in a period between the two stages, which makes it hard to identify even for a farmer.

However, even the obtained dataset could demonstrate a considerably better performance if it was not for the processing limitations. Although hundreds of images were taken and labeled across the span of the project, training with them all at the same time proved impossible due to Colab restrictions. First, 200 epochs were trained across the span of several weeks with approximately 70 photographs per epoch per GPU. The training could happen only for a couple hours a day before it would be abruptly finished. The model showed good detection capabilities but the classes were identified incorrectly in many cases. Consequently, it was decided to gather more data, adjust the labels and start the training again. Unfortunately, during the mid-December Google posed considerable restrictions to unpaid accounts. With a paid subscription not being allowed in Spain, the training process was rendered virtually impossible. There was no available computer with the necessary processing power and RAM either. In consequence, the training stopped after the epoch 137.

Some networks are trained for hundreds of epochs with hundreds of steps per epoch to obtain good results [74]. Moreover, the projects that employ region-based neural networks typically do not require the detection of complex shapes as it was the case in this project. These CNNs show exceptional results with detecting humans, cars, fruits, nucleus in cells, and other simpler shapes. Given the complexity of the task at hand, this project's network required more training with more data per epoch. However, even the obtained results show that the network is capable of recognizing objects in an image more often than not. It improved its precision and confidence continuously with more training epochs.

5 Conclusions and Future work

5.1 Conclusions

In this project, the Deep Learning field was studied to assess its combined application with UAVs in agriculture. The neural network, which is the underlying technology of Deep Learning, was researched to understand the basis of its functioning and potential for the world of technology. Accordingly, the most suitable type of neural network and its implementation was selected to fulfill the desired objectives. The implementation, data obtainment and its processing techniques were studied as well to provide transparency as to how the neural network was trained. Additionally, the field of UAV was briefly investigated as well to provide the context of its usage and relevance for the project. A UAV designed for agricultural purposes was successfully employed for taking the pictures that after going through a preparation process were used to train the programmed neural network to be able to recognize different object instances.

The objectives were not fulfilled entirely. The trained network showed impressive results since it was capable of automatically detecting objects and classify them in many cases. The results, however, could be considerably better with more available computational power and data samples. However, this project showed that using Deep Learning and UAV together can potentially be of great benefit for the agriculture and humanity.

5.2 Future work

This work could benefit greatly if there is a possibility to continue the training of the obtained neural network. If it is clear that the model has reached high precision in detecting potatoes and can no longer be improved, other crops could be used for the training, which would allow the network to detect different kinds of crops in different growth stages. Additionally, the detection could be done in real-time if network could be implemented in the hardware of a drone. Thus, the agriculture could acquire an UAV that would be able to fly over crop fields with different kinds of crops on them, analyze the crops and send the acquired information to a server. All done simultaneously.

Bibliography

[1] Food and Agriculture Organization of the United Nations. *Food Security and Nutrition in the World Security , Improved Nutrition and Affordable Healthy Diets for All.* 2021.

[2] Andrea Saba Sasot. Study of the Development of an IoT-based sensor platform for E-Agriculture. 2020.

[3] Carlos Aparisi Cantero. Desenvolupament d ' un dron customitzat per ús en aplicacions d ' agricultura. 2021.

[4] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017. Accessed: 2021-11-27.

[5] Image polygonal annotation with python. `https://github.com/wkentaro/labelme`, 2021.

[6] Hua Zhang, Rui Zhang, Daquan Sun, Fan Yu, Zhang Gao, Shuifa Sun, and Zichang Zheng. Analyzing the pore structure of pervious concrete based on the deep learning framework of Mask R-CNN. *Construction and Building Materials*, 318:125987, feb 2022.

[7] Dangfu Yang, Xiang Wang, Haoran Zhang, Zhen yu Yin, Dong Su, and Jun Xu. A Mask R-CNN based particle identification for quantitative shape evaluation of granular materials. *Powder Technology*, 392:296–305, nov 2021.

[8] Kit de marco plegable de fibra de carbono para dron, 560mm, motor sin escobillas 40a esc 1455, hÃ©lices apm2.8 con brÃºjula, rc quadcopter. `https://es.aliexpress.com/item/1005001374620900.html?spm=a2g0s.904231-1.0.0.274263c0NaPokj.%20Consultat%20el%2005-03-2021.`, 2021. Accessed: 2021-12-10.

[9] Michael Oborne. Mission planner overview. `https://ardupilot.org/planner/docs/mission-planner-overview.html#mission-planner-overview`, 2020. Accessed: 2022-01-02.

[10] Gopro api for python. `https://github.com/KonradIT/gopro-py-api`, 2020. Accessed: 2021-12-13.

[11] The difference between artificial intelligence, machine learning and deep learning. <https://spotcloud.medium.com/la-diferencia-entre-inteligencia-artificial-machine-learning-y-deep-learning-cc415f20e63a>. Accessed: 2021-11-29.

[12] Jason Brownlee. What is Deep Learning? https://machinelearningmastery.com/what-is-deep-learning/, 2020. Accessed: 2021-12-02.

[13] Y. S. Park and S. Lek. *Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling*, volume 28. Elsevier, 2016.

[14] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[15] Jason Brownlee. Gentle introduction to the bias-variance trade-off in machine learning. https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning, 2016. Accessed 2021-12-21.

[16] Ismail Mebsout. Deep learning's mathematics. https://www.ismailmebsout.com/Convolutional%20Neural%20Network%20-%20Part%201/. Accessed: 2021-12-06.

[17] Alan Woodruff. Visual perception. https://qbi.uq.edu.au/brain/brain-functions/visual-perception. Accessed: 2021-12-06.

[18] Pragati Baheti. 12 types of neural network activation functions: How to choose? https://www.v7labs.com/blog/neural-networks-activation-functions, 2021. Accessed: 2021-12-04.

[19] Chun-Nan Chou, Chuen-Kai Shie, Fu-Chieh Chang, Jocelyn Chang, and Edward Y. Chang. Representation Learning on Large and Small Data. *Big Data Analytics for Large-Scale Multimedia Search*, pages 1–28, 2019.

[20] Kenneth Leung. The dying relu problem, clearly explained. https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24, 2021. Accessed: 2021-12-15.

[21] Daniel Burkhardt Cerigo. On why gradient descent is even needed. https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635, 2018. Accessed: 2021-12-14.

[22] Matt Mazur. A step by step backpropagation example. https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/. Accessed: 2021-12-08.

[23] IBM Cloud Education. Gradient descent. https://www.ibm.com/cloud/learn/gradient-descent, 2020. Accessed: 2021-12-08.

[24] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[25] Hanxiao Liu. Adaptive Subgradient Methods for Stochastic Optimization. 2015.

[26] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. 2012.

[27] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it. `https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it` 2018. Accessed: 2021-12-20.

[28] Alberto Romero. 5 Deep Learning Trends Leading Artificial Intelligence to the Next Stage. `https://towardsdatascience.com/5-deep-learning-trends-leading-artificial-intelligence-to-the-next-stage-11f2ef6` 2021. Accessed: 2021-12-03.

[29] Great Learning Team. Types of neural networks and definition of neural network. `https://www.mygreatlearning.com/blog/types-of-neural-networks/`, 2020. Accessed: 2021-12-03.

[30] Smaranda Belciug. Parallel versus cascaded logistic regression trained single-hidden feedforward neural network for medical data. *Expert Systems with Applications*, 170:114538, may 2021.

[31] Huile Li, Tianyu Wang, and Gang Wu. Dynamic response prediction of vehicle-bridge interaction system using feedforward neural network and deep long short-term memory network. *Structures*, 34:2415–2431, dec 2021.

[32] Maozhang Zheng, Jianjun Luo, and Zhaohui Dang. Feedforward neural network based time-varying state-transition-matrix of Tschauner-Hempel equations. *Advances in Space Research*, oct 2021.

[33] nerdthecoder. Recurrent neural networks. `https://nerdthecoder.wordpress.com/2019/02/03/recurrent-neural-net/`, 2019. Accessed: 2021-12-21.

[34] Adrian Colyer. Recurrent neural network models. `https://blog.acolyer.org/2017/03/23/recurrent-neural-network-models/`, 2017. Accessed: 2021-12-20.

[35] Shankar Ananthakrishnan. Amazon scientists applying deep neural networks to custom skills. `https://www.amazon.science/blog/amazon-scientists-applying-deep-neural-networks-to-custom-skills`, 2020. Accessed: 2021-12-13.

[36] Nasir Saleem, Jiechao Gao, Muhammad Irfan Khattak, Hafiz Tayyab Rauf, Seifedine Kadry, and Muhammad Shafi. DeepResGRU: Residual gated recurrent neural network-augmented Kalman filtering for speech enhancement and recognition. *Knowledge-Based Systems*, page 107914, dec 2021.

[37] Qiao Wang, Min Ye, Meng Wei, Gaoqi Lian, and Chenguang Wu. Co-estimation of state of charge and capacity for lithium-ion battery based on recurrent neural network and support vector machine. *Energy Reports*, 7:7323–7332, nov 2021.

[38] Jonas Van Gompel, Domenico Spina, and Chris Develder. Satellite based fault diagnosis of photovoltaic systems using recurrent neural networks. *Applied Energy*, 305:117874, jan 2022.

[39] Ben Dickson. Tesla ai chief explains why self-driving cars don't need lidar. `https://venturebeat.com/2021/07/03/tesla-ai-chief-explains-why-self-driving-cars-dont-need-lidar`, 2021. Accessed: 2021-12-14.

[40] D.T.T. Vijaya Kumar and R. Mahammad Shafi. Analysis and fast feature selection technique for real-time face detection materials using modified region optimized convolutional neural network. *Materials Today: Proceedings*, may 2021.

[41] Ahmed Sabeeh Yousif, Zaid Omar, and Usman Ullah Sheikh. An improved approach for medical image fusion using sparse representation and Siamese convolutional neural network. *Biomedical Signal Processing and Control*, 72:103357, feb 2022.

[42] Farshid Rayhan, Sajid Ahmed, Zaynab Mousavian, Dewan Md Farid, and Swakkhar Shatabda. FRnet-DTI: Deep convolutional neural network for drug-target interaction prediction. *Heliyon*, 6(3):e03444, mar 2020.

[43] Yann LeCun et al. Object recogntion with gradient-based learning. `http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf`, 1998. Accessed 2021-12-21.

[44] Cnn edge detection. `https://datahacker.rs/edge-detection`, 2018. Accessed 2021-12-17.

[45] Chirag Goyal. 20 questions to test your skills on cnn. `https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks`, 2021. Accessed 2021-12-21.

[46] CS231n. Convolutional neural networks for visual recognition. `https://cs231n.github.io/convolutional-networks/#pool`. Accessed 2021-12-18.

[47] Vgg-16 | cnn model. `https://www.geeksforgeeks.org/vgg-16-cnn-model/`, 2020.

[48] Aseem Patil and Milind Rane. Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition. *Smart Innovation, Systems and Technologies*, 195:21–30, 2021.

[49] Sabina Pokhrel. Beginners guide to convolutional neural networks. `https://towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural-networks-ae9ed58bb17d`, 2019. Accessed: 2021-12-23.

[50] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[51] Ross Girshick. Fast R-CNN. Technical report.

[52] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[53] Shilpa Ananth. Faster r-cnn for object detection. `https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46`, 2019. Accessed: 2021-12-20.

[54] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397, 2020.

[55] ICAO. *Unmanned Aircraft Systems (UAS).* 2011.

[56] Francisco Klauser and Dennis Pauschinger. Entrepreneurs of the air: Sprayer drones as mediators of volumetric agriculture. *Journal of Rural Studies*, 84:55–62, may 2021.

[57] K. R. Krishna. Drones in Agriculture. *Agricultural Drones*, (January):1–59, 2018.

[58] Michal Mazur. Six ways drones are revolutionizing agriculture. `https://www.technologyreview.com/2016/07/20/158748/six-ways-drones-are-revolutionizing-agriculture/`, 2016. Accessed: 2022 - 01 -02.

[59] David Anthony, Sebastian Elbaum, Aaron Lorenz, and Carrick Detweiler. On crop height estimation with UAVs. *IEEE International Conference on Intelligent Robots and Systems*, (Iros):4805–4812, 2014.

[60] UASLogic. Using drones for precision agriculture. `https://www.uaslogic.com/drones-for-precision-agriculture.html`, 2016. Accessed: 2022 - 01 - 03.

[61] Lala El Hoummaidi, Abdelkader Larabi, and Khan Alam. Using unmanned aerial systems and deep learning for agriculture mapping in Dubai. *Heliyon*, 7(10):e08154, oct 2021.

[62] The agency. `https://www.easa.europa.eu/the-agency/the-agency`. Accessed: 2022 - 01 -01.

[63] Â¿quÃ© es aesa? `https://www.seguridadaerea.gob.es/en/quienes-somos/que-es-aesa`. Accessed: 2022 - 01 -01.

[64] Focus crops: Bread wheat, durum wheat, and potato. https://www.solace-eu.net/de/about/focus-crops.html, 2019.

[65] Pulse and soybean variety guide. https://www.manitobapulse.ca/production/variety-evaluation-guide/, 2021. Accessed: 2022-01-03.

[66] Gopro quik: Video editor and slideshow maker. https://play.google.com/store/apps/details?id=com.gopro.smarty&hl=en_US&gl=US.

[67] Virupaksh Patil, Prashant Kawar, Sundaresha S, and Vinay Bhardwaj. Biology of Solanum tuberosum (Potato): Series of Crop Specific Biology Document. *Ministry of Environment, Forest and Climate Change*, (December):1–28, 2016.

[68] Jude E. Obidiegwu, Glenn J. Bryan, Hamlyn G. Jones, and Ankush Prashar. Coping with drought: Stress and adaptive responses in potato and perspectives for improvement. *Frontiers in Plant Science*, 6(JULY):1–23, 2015.

[69] Jason Brownlee. What is the difference between test and validation datasets? https://machinelearningmastery.com/difference-test-validation-datasets/, 2017. Accessed: 2022-01-02.

[70] Jason Brownlee. What is the difference between a parameter and a hyperparameter? https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/, 2019. Accessed: 2022-01-02.

[71] Michael Balaban. Choosing the best gpu for deep learning in 2020. https://lambdalabs.com/blog/choosing-a-gpu-for-deep-learning/, 2020. Accessed: 2022 - 01 - 02.

[72] Get started with tensorboard. https://www.tensorflow.org/tensorboard/get_started. Accessed: 2022 - 01 -01.

[73] Yassine Ait Jeddi. Confusion-matrix-for-matterport-implementation-of-mask-r-cnn. https://github.com/Altimis/Confusion-matrix-for-Mask-R-CNN. Accessed: 2022 - 01 -02.

[74] Pulkit Sharma. A practical implementation of the faster r-cnn algorithm for object detection (part 2 - with python codes). https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/, 2018.