UDC: 004.8:632.03
Professional paper

METHOD OF ARTIFICIAL INTELLIGENCE IN IDENTIFICATION OF PLANT DISEASES

Zani DALIPI[1*], Bilall I. SHAINI[2], Fisnik AHMETI[1*], Shpëtim REXHEPI[3]

[1]Faculty of Informatics, Mother Teresa University, Skopje, NMK*
[2]Faculty of Applied Sciences, University of Tetova, Tetova, NMK
[3]Faculty of Applied Sciences, Mother Teresa University, Skopje, NMK
[1]Corresponding author e-mail: zanoin@gmail.com*
bilall.shaini@unite.edu.mk
fisnik.ahmeti@students.unt.edu.mk
shpetim.rexhepi@unt.edu.mk

Abstract

In this paper, we explain the influence of modern technology in the identification of plant diseases. The best way nowadays to solve this problem is to use some artificial intelligence methodologies. Still, in particular, we will train Machine Learning as a field that deals with the detection of plant diseases. In this paper, we will talk about the basic principles of how artificial intelligence can be practical in agriculture, to select some problems that were once not possible due to the lack of development of technology or the lack of scientific knowledge. Through this work, a software developer can get basic knowledge to develop software or applications that will be able to identify plant diseases. The work includes the methodology of how plant diseases can be identified through any application. Our contribution is for all developers who are interested in developing technology in the field of agriculture, for this reason, we have explained the concepts of artificial intelligence and software development methodologies. Readers can have a clear overview of technology development in the agricultural sector.

Keywords: Artificial intelligence, Software in agriculture, Transfer-learning, CNN architecture, Optimizing weights.

Introduction

The purpose of this topic is to give researchers and developers a clear overview of the methodologies for the development of technology or software in the field of agriculture that will help certain people involved in agriculture, as well as learn about many new concepts such as machine learning, Deep learning, how to work with such systems and many other details that I will explain below.

First, we will explain the concepts, i.e. the basic things we need to understand for the development of a successful machine system, as concepts we explain AI as a whole or a historical explanation of AI, then an explanation of the branches of AI i.e. tools such as machine learning and deep learning delves deeper into Deep learning by explaining the vision in computers and humans also explaining how neurons work in humans and machines, and briefly explaining the components of a CNN architecture also for TensorFlow and TensorFlow-hub.

The second chapter first explains how the development of a machine/deep system goes, then explains the types of data as well as the approaches to collecting that data, after getting to know what data we have and what approaches we can use depending on how they are organized or how we will organize the data we have to decide which of the 3 approaches we will use whether supervised or unsupervised approaches. In my case, I used supervised because that's how data is organized.

The third chapter explains the training process where we have 2 approaches: training from the beginning or using transfer learning. To explain it better we show transfer-learning where we

https://doi.org/10.62792/ut.jnsm.v9.i17-18.p2838

have to follow some rules that define the approaches that should be taken depending on the data we possess.
The fourth chapter explains how the CNN architecture used in Inception v3 works in more detail than what is explained in the first chapter where we will explain all the components that make up the Inception v3 module example: flattened layer, Fully-connected layer also for different functions which we can use for output, in this chapter we will explain the problems that arise during training and how to find those problems, problems like over-fitting and under-fitting.
In the fifth chapter we return to the basic structure of a neural network which was explained in the first chapter, in this chapter we will go into detail about a neural network, in which a mathematical formula is performed to get the result of the next neuron, which are that weights, what rules we need to follow to define these weights. This approach is called forward propagation. Algorithm for optimizing weights and measuring loss for each neuron.

1. Concepts

1.1 AI vs ML vs DL?: In some cases, human knowledge may not be enough, so in the 50's man began to study expert systems, with these systems intended to overcome various problems that will replace human expertise or help him in certain tasks, such systems are called knowledge-based systems.
Real-world Artificial Intelligence (AI) solutions need different data management tools, set to different scales. AI enables the machine to understand, make interactions, and communicate with people in a human way. Machine Learning (ML) can be described as an AI application that provides systems and the ability to automatically learn and improve from experiences without clear programming. In In-Depth Learning (DL), the computer will receive data or learn from voice, images, or text and perform the action accordingly. There are many examples of DL applications currently becoming popular: airspace and defense, driving, medical automation, industrial, and electronics. The comparison between ML and DL can be made on the following basis: data dependencies, hardware dependencies, problem-solving approach, feature engineering, execution time, and interpretability. ML-based virtual assistants help people on many platforms, such as: mobile applications, multi-command smartphones, and smart speakers [7].

1.1.2 Artificial Intelligence (AI): As a concept first mentioned in 1956 and is the father of ML and DL. What is AI? Is a technique or algorithm that allows a machine to mimic human behavior. AI also allows the machine to learn from previous events. Examples are Alexa or similar systems, as well as other tools such as machine learning, Deep Learning, perhaps reinforcement learning, and others.
There are many different definitions of artificial intelligence but one on which different experts in the field agree is that first artificial intelligence studies the way people think (to understand what intelligence is), and secondly it deals with the representation and repetition of these processes in machines.
Artificial Intelligence (AI) has the definite goal of understanding intelligence and building intelligent systems. However, the methods and formalisms used on the way to this goal are not firmly set, which has resulted in AI consisting of a multitude of sub-disciplines today. The difficulty in an introductory AI course lies in conveying as many branches as possible without losing too much depth and precision [1].
A popular example of artificial intelligence is Deep Blue, a chess program developed by researchers from an IBM team that defeated chess champion Garry Kasparov. We know that chess is a game where intelligent people can win.

An interesting test to determine if a computer exhibits intelligent behavior was designed by Alan Turing, called the Turing test.
In this test, a human examiner communicates with a computer and a person who does not see them, according to this test, a computer can be considered intelligent only when the human examiner cannot determine who is a computer and who is a human. This term is successful if the person guesses with 50% certainty to determine which terminal is controlled by the person and which by the machine.
Artificial intelligence includes many sub-areas (Machine learning, Computer Vision, Deep Learning, Neural Networks Robotics, Speech recognition, and Natural language processing) The following is an image from the AI tree:

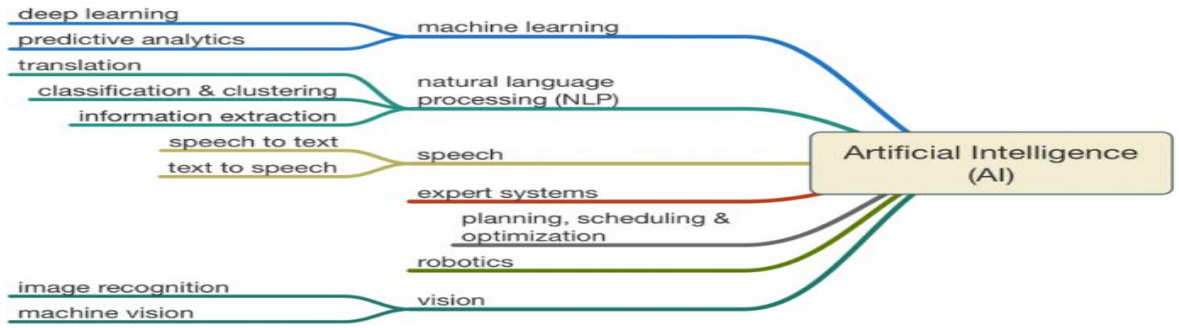


Figure 1. AI tree and their branches

Nowadays, many systems have been developed that can be used to say that the throng test is successful: a system has been developed in China, i.e. a robot with the appearance of a man who tells the news.
This system was developed in this way using the GAN (generative adversarial network) and was trained on real news videos, so the robot learned how to speak without being recognized, it also learned various facial movements like a human.
Other similar systems: are Google Duplex, Siri, Alexa, and Sophia (Robot).
However, even if a lot of success is achieved with AI, not everything can be achieved using AI systems, so there is a rule that reads "A system is considered to be successfully implemented if that system can give us a response faster than humans or equally that is, to give us an answer in one second.

1.1.3 Machine Learning (ML): Machine learning is a discipline focused on two interrelated questions: How can one construct computer systems that automatically improve through experience? And what are the fundamental statistical computational-information-theoretic laws that govern all learning systems, including computers, humans, and organizations? The study of machine learning is important both for addressing these fundamental scientific and engineering questions and for the highly practical computer software it has produced and fielded across many applications [2].
As a concept occurred in 1990, as I said machine learning is under a set of artificial intelligence, unlike AI which had Hard-coded algorithms or fixed algorithms, systems with limited capabilities were not good enough to process images and classify them. ML as an improvement of AI with more flexible algorithms allowed developers not only to monitor human behavior but also enable machines to learn just like people in certain areas who can complete a given task. It is also important to note that machine learning uses more statistics or data processing much faster than humans. As an example, we can mention the classification of e-mail whether it is spam or not spam has no human interaction the system itself classifies them.

As it is written above, machine learning is an improvement of AI in terms of a tool, machine learning is essentially a system (software) that is created from data, i.e. learns from data certain paths and gives us an answer using input (A) gives us response output (B) [9].
Definition of machine learning: Tom Mitchell calls a program learns from experience (E) by performing a task (T) and performance (P), which after completing a task (T) by measuring performance (P) you gain experience (E).
Example with playing lady:
E = gaming experience a lady gains a lot of gaming experience
T = tasks from playing lady
P = the probability that the program will win when playing with another player.
Another example of machine learning can be said to be Google example when searching a site example we will take AliExpress when searching for certain products (T) then on the home page it shows us more such products (E) or products in the same category, machine learning processes data and returns the answer whether that customer is interested in a particular product [9].
Abduction is, along with induction, a synthetic form of reasoning whereby it generates, in its explanations, new information not hitherto contained in the current theory with which the reasoning is performed. As such, it has a natural relation to learning, and in particular, to knowledge-intensive learning, where the new information generated aims to complete, at least partially, the current knowledge (or model) of the problem domain as described in the given theory. Early uses of abduction in the context of machine learning concentrated on how education can be used as a theory revision operator for identifying where the current theory could be revised to accommodate the new learning data [3].

1.1.4 Deep Learning (DL): Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains such as drug discovery and genomics. Deep learning discovers intricate structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech, and audio, whereas recurrent nets have shone light on sequential data such as text and speech [4].
Deep learning allows computational models of multiple processing layers to learn and represent data with multiple levels of abstraction mimicking how the brain perceives and understands multimodal information, thus implicitly capturing intricate structures of large-scale data. Deep learning is a rich family of methods, encompassing neural networks, hierarchical probabilistic models, and a variety of unsupervised and supervised feature learning algorithms [11].
A system that can be said to be the brain in a system processes data using ANN (Artificial Neural Networks) that work in a similar way to neural networks in the human brain, this system is under a set of artificial intelligence. Deep learning means entering the inside of a machine system that is simpler in structure i.e. has fewer layers compared to Deep Learning.
There are many systems created with this system in recent years because nowadays it is very easy to access data as well as fast processing of that data using GPUs or lately TPUs and efficient algorithms.
Applications created with DL: Self-Driving-Cars, NLP (natural language processing), Google Translate camera, chat-bots, etc.
AI VS ML VS DL - artificial intelligence is the big picture from which many systems are then created like ML and DL, ML which has many tools is a system that learns from data or previous

experience. Then there is DL which, as the name implies, processes internal data using Artificial Neural Networks which were created in the 80's but with the advent of DL became very popular. Speaking of Deep Learning (brain), to process data there must be input from somewhere for this purpose, computer vision (Computer Vision) which can be categorized as the human eye.
The most common form of machine learning, deep or not, is supervised learning. Imagine that we want to build a system that can classify images as containing, say, a house, a car, a person, or a pet. We first collect a large data set of images of houses, cars, people, and pets, each labeled with its category. During training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category [4].
DL extends classical ML by adding more “depth” (complexity) into the model as well as transforming the data using various functions that allow data representation hierarchically, through several levels of abstraction [5]. A strong advantage of DL is feature learning, i.e. the automatic feature extraction from raw data, with features from higher levels of the hierarchy being formed by the composition of lower-level features [6].

1.2 Computer Vision: " If we want machines to learn, we must first teach them to see," but the question is, what is computer vision?
This field is very important and also complicated because man since the 50's began to examine the human eye, which is a complex process; the evaluation of the human eye is very advanced in terms of Computer vision.
The approach to computer vision is best characterized as inverse computer graphics. In computer graphics, the world is represented in sufficient detail so that the image-forming process can be numerically simulated to generate synthetic television images; in the inverse, perceived television pictures (from a real TV camera) are analyzed to compute detailed geometric models [12].
The best visualization tool is the human eye, a tool used by the brain to sense and see the world, but can that level be achieved with computers?
The beginnings to create a system that simulates the human brain prompted the initial development of neural networks. In 1943, McCulloch and Pitts tried to understand how the brain could produce highly complex patterns using interconnected basic cells, called neurons [16]. The McCulloch and Pitts model of a neuron called the MCP model, has made an important contribution to the development of artificial neural networks. A series of major contributions in this field are shown in the above table [17], including LeNet [18] and Short-Long-Term Memory [19], leading to the "era of deep learning". One of the most fundamental advances in deep learning came in 2006, when Hinton et al. [20] introduced the Deep Belief Network, with multiple layers of constrained Boltzmann machines, greedily training one layer at a time in an unsupervised manner.

Table 1: Important milestones in the history of neural networks and machine learning, leading up to the era of deep learning [17]

Contribution	Contributor, year
MCP model, regarded as the ancestor of the Artificial Neural Network	McCulloch & Pitts, 1943
Hebbian learning rule	Hebb, 1949
First perceptron	Rosenblatt, 1958
Backpropagation	Werbos, 1974
Neocognitron, regarded as the ancestor of the Convolutional Neural Network	Fukushima, 1980
Boltzmann Machine	Ackley, Hinton & Sejnowski, 1985
Restricted Boltzmann Machine (initially known as Harmonium)	Smolensky, 1986
Recurrent Neural Network	Jordan, 1986
Autoencoders	Rumelhart, Hinton & Williams, 1986

LeNet, starting the era of Convolutional Neural Networks	LeCun, 1990
LSTM	Hochreiter & Schmidhuber, 1997
Deep Belief Network, ushering the “age of deep learning”	Hinton, 2006
Deep Boltzmann Machine	Salakhutdinov & Hinton, 2009
AlexNet, starting the age of CNN used for ImageNet classification	Krizhevsky, Sutskever, & Hinton, 2012

1.3 What are neurons?

1.3.1 Neurons in the biological brain: Neurons in the biological brain there are billions (100 b) each neuron performs a specific task, i.e. the brain can be thought of as a set of sensors: visual sensor (Virtual Cortex), sensory sensor, taste sensor, etc.
Each sensor or part of the brain has its network of neurons that are connected to process a certain event or process electrical impulses for a certain action by humans.

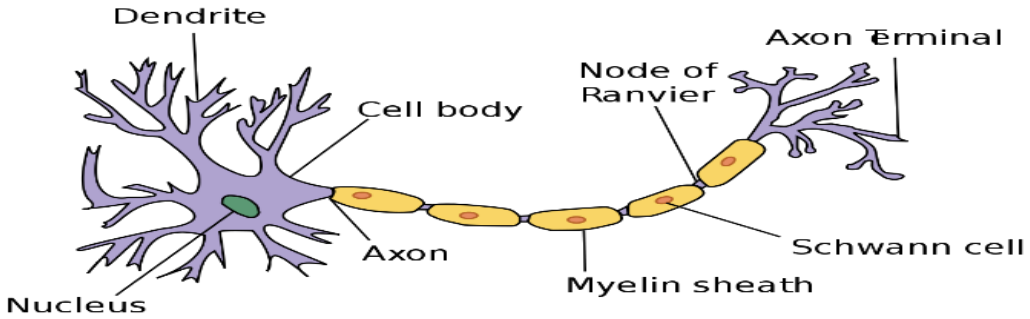


Figure 2. Picture of a neuron

The picture above shows a neuron in the brain. How does it work? A neuron works by processing electrical impulses from "Dendrite" or in other words input wires then that signal is processed inside the "Cell body" or the body of the neuron, what does it process and how does it process them? One neuron is nothing but performing some impulses (in mathematics calculations), and then that impulse is transferred using an "Axon" or output channel to the terminal where that impulse will be processed by another neuron or if it is at a high level of processing is sent to the sensor or part of the brain and that impulse is sent to the nervous system.
It should also be noted that neurons process data non-linearly, i.e. wait to receive as much of the input signals as possible to a certain degree or threshold which can stimulate a complex eclectic impulse, when that threshold is full that neuron will release it. This signal to another terminal is called "fire" in English when the threshold level is reached in the part of artificial neural networks. This process is called activation of the result obtained from the previous neuron and the weights associated with that neuron where that result is activated. Using one of the activation functions (sigmoid, SoftMax, relog, etc.), with a simple example threshold-from can be explained when we fill a glass with water, water will not flow out of the glass until it is filled.
ANN (artificial neural networks) was invented similarly. Back in the 80s, one example in which ANN was used was the grayscale number identification system, then neural networks were not so popular because they required large processing as we know at that time computers were very slow.

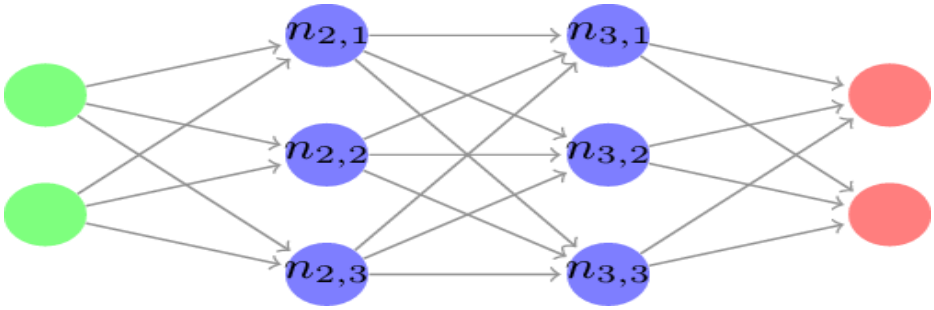


Figure 3. Basic neural network architecture

Θ^{1} = [w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23}], Input (x) = [input_1, input_2]

1.3.2 The structure of artificial neural networks: The structure of artificial neural networks is similar to that of neurons in the human brain, where we have input (Dendrite), hidden layers (Cell body), and output (Axon).

Notation:

- Θ^{j} Theta (weights) - a matrix used to multiply a layer by this matrix and get a new layer or activation. j - An indicator in which that matrix with weights is between the layers. Calculating the size of the matrix: $S_{j+1} * (S_j + 1) = 2 + 1 * 2 + 1 + 1 = 3$ x 4 in the input we add bias. Also, the names of these weights in some literature are named W (weights) W_{ij}, i - which neuron (node) from this layer is connected to j - which neuron (node) from the next layer. Example $W^{1}{}_{2}$

For weights (Theta) the parameters can also be thought of as an indicator that tells us the strength of the signal relative to the neuron that is associated with that weight in the other layer.
Question: Why do we connect all the nodes to each other? Doesn't that cause a lot of operations to be performed?
We can also create our own connection to the neurons but the way the neurons are connected is easy for the computer to encode, another drawback is that if the neurons are not connected properly we can lose important information from processing, maybe in the first layers we will have more operations to perform depending on the weight parameters, but as we know our neural network learns by multiplying the input by the weights and getting activation in the second layer so some of these weights can also have value 0 or close to 0, these values automatically do not contribute anything to the network of neurons because those signals do not pass, they do not pass because these neurons wait for a certain threshold to be satisfied for a certain signal (neuron) to be activated.

- $n^{j,i}$ - the result obtained or in neural networks called activation, i - is which neuron in that layer is j, i.e. we index the neurons for example layer (j) - 2, neuron (i) - 1.

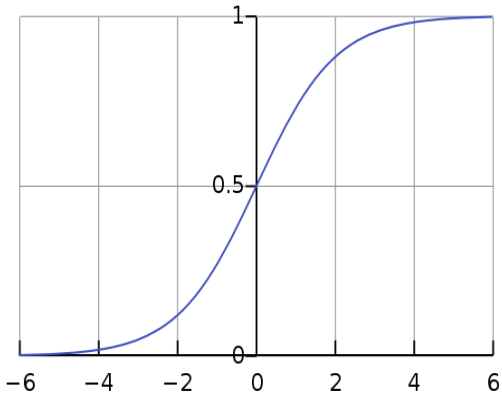
$n^{(2)}{}_{0} = 1$ (BIAS which is always equal to 1 in Neural network) -> $z^{(2)}{}_{0}$
$n^{(2)}{}_{1} = g\,(w^{(1)}{}_{11} * input\_1 + w^{(1)}{}_{21} * input\_2) + bias$ -> $z^{(2)}{}_{1}$ - (Vectorization of the result)
$n^{(2)}{}_{2} = g\,(\Theta^{(1)}{}_{21} * input\_1 + w^{(1)}{}_{22} * input\_2) + bias$ -> $z^{(2)}{}_{2}$ - (Vectorization of the result)
$n^{(2)}{}_{3} = g\,(+ w^{(1)}{}_{13} * input\_1 + w^{(1)}{}_{23} * input\_2) + bias$ -> $z^{(2)}{}_{3}$ - (Vectorization of the result)

Formula can be simple: **X = W * I**
X - Combination of all signals received as a matrix for the next layer
W - Is a matrix with weights, I - a matrix of input parameters
If we do not vectorize the results obtained then the activation function (sigmoid, SoftMax, rELU) will be performed on all values of the X matrix, this problem is easily avoided by already having results i.e. we do not combine different signals from different neurons, it is done with the previous formula X = W * **I.**
Formula: **O = sigmoid (X)** - The O notation contains all the values from the last layer.

We will vectorize all these processes, i.e. we will represent them as a vector, we will use z notation

X (z ^{2})=[z_0^2 z_1^2 z_2^2 z_3^2]

hΘ (x) = n $^{(3)}{}_{1}$ = g (Θ $^{(2)}{}_{10}$ n $^{(2)}{}_{0}$ + Θ $^{(2)}{}_{11}$ n $^{(2)}{}_{1}$ + Θ $^{(2)}{}_{12}$ n $^{(2)}{}_{2}$ + Θ $^{(2)}{}_{13}$ n $^{(2)}{}_{3}$) -> z ^{3} - this is for the final result although in the picture above we have another hidden layer which is calculated just like the first layer of the hidden layer of the grid.
g (S (X)) - is a Sigmoid function for activating a neuron.



$S(x) = \frac{1}{1+e^{-x}}$

Figure 4. Graphic representation of sigmoid activation function and formula

1.4 What is CNN (Convolutional Neural Network): At present CNN has proven to be excellent in a wide range of tasks in terms of computer vision, its high performance ease of understanding, and ease of training are the two factors CNN has been using in recent years. CNN focuses on the inputs that will consist of images. This focuses on the architecture that will best fit the need to deal with the appropriate type of data [10].

1.4.1 CNN's architecture: CNN's architecture has 2 parts:
(1) Convolutional base (feature Extractor) - these are the first layers of the architecture which include layers for convolution, rELU, and pooling layers. The layers in this section are used to find the basic characteristics of the images and configure the parameters (weights) for the best possible results.
(2) Classifier - this layer is usually composed of flattened, fully-connected layers and an activation function e.g. sigmoid, SoftMax, the purpose of this part of CNN architecture is to classify images based on the previously obtained characteristics of images from the first part of CNN, these layers or more precisely flatten I fully-connected-layers are directly connected to all the activations (neurons) from the previous layer, the layers that are in the first part.
These Deep Learning models can automatically learn the hierarchical representation of features, which means that the features calculated in the first layer are basic features that can be used in various image classification problems.
Convolution - a mathematical operation that processes one function into another to obtain a sketch for the next layer.
Explain the simple structure of CNN architecture:

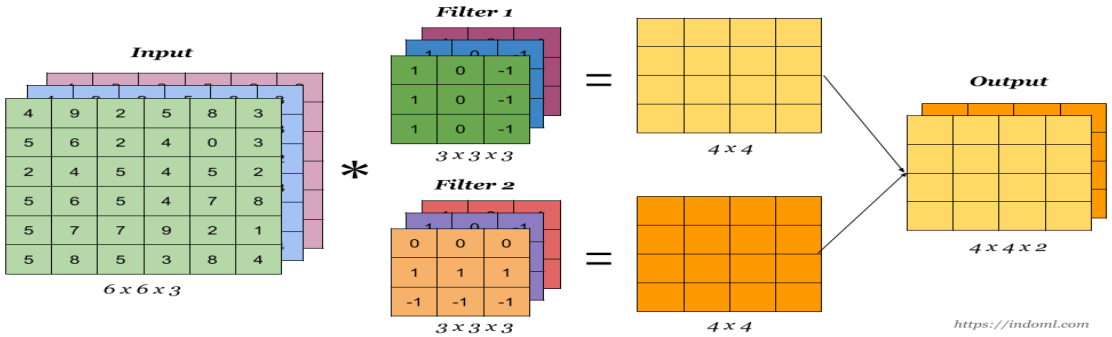


Figure 5. Explanation of CNN convolution with kernel filters

CNN Explanation: We will explain the basic structure of this architecture where we have input, hidden layers and output.
Input - depending on what kind of image it is, whether it is RGB or grayscale, the channels or layers of an image are determined, for example in RGB we have 3 layers (channels) and in grayscale, we have 1. So as input, we have an image with dimensions of 6x6x3, i.e. RGB image is followed by hidden or inner layers, these layers perform various mathematical operations until we get **a tensor** (matrix with dimensions 1 xn).
In the first layer, we need to set the hyperparameters as the size of the **kernel filter (f) -** filter (weights) which is a simple matrix that can be of different sizes most used 1 x 1, 3 x 3, 5 x 5 matrix… i.e. functionality on this filter it is the same as looking at a picture with a magnifying glass to reveal the details in that picture, different hyperparameters should also be used to find different edges or find an object in the picture.
Stride (s) - is a parameter used to determine how many pixels the kernel filter (magnifying glass) moves or how many pixels it looks at with a magnifying glass.
Padding (p) - very rarely used, we add a pudding if we want the image not to be drastically reduced during CONVOLVE.
Number of filters (matrix depth) (nF) - the number of kernel filters on which the image will be multiplied.
Also, the number of channels in the kernel filter and the image must be the same.
MAXPOOL - the purpose of this technique is to reduce the image by a factor of 2, for example, if we use a 2x2 matrix for maxpool then only the largest number is taken from the 4 numbers that will be processed.
RELU (rectified linear unit) - nonlinear activation function used after each layer, with this function we convert the negative values into 0 the other values of the function remain the same.

2 Data collection different approaches to organizing training

All machine systems as the first condition that must be met are to collect the necessary data.
It is important to note that in machine learning we have 3 different approaches to organizing data:

2.1 Supervised Learning : Supervised learning is the most important methodology in machine learning and also has a central importance in multimedia data processing [13].
The idea of this process is to organize the data in classes (nodes) by labeling them and expecting a result from organized data, i.e. we process the input and we know the output result, we have a short relation between the input and the output.
In my case I use this approach from where I have organized the data by classes for 11 diseases of apples, peaches, grapes, and cherries.

Of all these different plants I focused more on apples because in that part I can collect more data in the future and improve that model, at the time I started the project it was too late to collect data in conditions where the application will be used and I failed to collect enough data to improve the model for use in real conditions, I use other data for other diseases from a database created by **Plant Village** with more than 32000 images for many types of diseases, from this database I use only part of the data by creating 11 classes per 1000 images.
Problem: The images in this database are in a controlled environment, i.e. when identifying there will be many problems such as light problems, the background would also affect the angle. How to overcome these problems, these problems can be overcome with more images in real environments or by using more advanced techniques in Computer Vision such as:
Objects Localization - localize an object by creating a frame and convolving only that frame by ignoring the background.
Segmentation - this technique divides the image into multiple regions, i.e. identifying different objects and ignoring the background.
This approach has 2 sub-approaches: Linear regression and logistic regression (classification)
Linear regression - is an approach where we expect a result continuously, we process entry into a continuous function. An example of linear regression is a company where we have many products that are similar in their features, but we want to predict how many of those products will be sold in the next 3 months.
On the other hand, another approach is logistic regression*;* in this approach we expect a discrete result.
On the other hand, multi-class classification means classification into multiple classes and gives us probability for all classes. It is understood which class is more likely that the program thinks it is most likely in the input or in other words the binary classification process is processed multiple times for each class.
In multi class classification or **one vs all** for each class a graph is created where an example for the first class when we create the graph only for that class is positive i.e. 1 (one) the others are negative, then it is created for the second class and so on, after processing all the results that algorithm returns the value that is most likely to be in that input.
Formula to predict that u (level) is part of a class, h (x) - is input this formula for each input will give us a probability
(0 to 1)

$h^{(0)}_\theta(x) = P(y = 0 \mid x; \theta)$
$h^{(1)}_\theta(x) = P(y = 1 \mid x; \theta)$
…
$h^{(n)}_\theta(x) = P(y = n \mid x; \theta)$

To find where it is most likely we use the following formula:
Prediction = $\max_i (h^{(i)}_\theta(x))$

2.2 Unsupervised Learning: On the other hand, in this process, there is a database that is not grouped into classes and we do not know the result, i.e. we have not defined it.
This type of process is expected to be used more in 2019 by combining Reinforcement Learning.
Example of unsupervised learning: an example is *Google News* which organizes the data in clusters, the clusters are organized based on some data example: in CNN news in my text I have written "Tornado in America" also another portal writes with the same text (“The tornado in America") this algorithm (clustering algorithm) will organize them in different news blocks.

2.3 Data Collection: Before explaining the steps required to collect data, it is important to explain the types of data and how they are collected.
We have Structured as data types Data (Spreadsheet data) and Unstructured Data (Image, Audio).
We have several ways of collecting data but the most common are the following 3:
Manual-Labeling - an approach that requires an expert piece of data in my project is collected with this method of data collection.
Data from websites - an example from the internet you can find a lot of data in certain areas or from different websites like Kaggle where there are many databases for different problems and they are free.
Some of my data was downloaded from Kaggle where there was a competition to create a plant disease identification system using the Plant Village database.
Data from Partner - receiving data from a company for which we are solving a certain problem.
When collecting data, it is important to have a large amount of data because under-fitting databases can have an under-fitting process which can be explained as follows: when training data it is important to split the data into training and validation data where it is important to reduce the loss with each iteration, in the case of under-fitting it means that the training data loss decreases which is good but the validation data increases.
Another problem we can have gaps in the database if the database is a spreadsheet where a way out needs to be found to fill those gaps.
When we talk about spreadsheet databases, there can also be a problem with values, i.e. as an example we will take the sale of a house where we have attributes such as: size, number of rooms, year as inputs and as output we have the price of that house which as price there may be values that are not realistic example price 0.01 MKD.

2.3.1 Labeling: This process takes the most time and is one of the most important steps because all the different data (images) in our case must be labeled or divided into classes all different diseases, in this process there must also be an expert in the field who will labels the data in this process we have bad and good labeling:
Bad labeling - the expert only identifies diseases in each category of plants without adding additional information about the disease.
Good labeling - the expert gives us more information about the diseases, where the expert, in addition to the disease, identifies the affected regions of the plants. As I said, this process is long-term and an expensive option, so most of the databases are poorly labeled.

2.3.2 Increasing the number of data (Data Augmentation): In this step it is important to increase the number of copies (images) using a technique called in English: Augmentation technique which is used most often by the developers of such Deep learning systems because as mentioned above good labeling is a laborious process that takes a lot in a row and of course time costs.
Augmentation - its purpose is to increase the number of samples or images by performing various geometric transformations such as: resizing, crop, image rotation, also changes in contrast, color, brightness, etc.
On the other hand, image pre-processing which is used to normalize the data in the database. During normalization, 2 techniques are performed such as changing the size of the input images to the size affected for CNN layers.
Example from Data Augmentation uses the Augmentator library which has many options for arguing an image so that with a small number of images we can increase that number by adding different affects to each of the images. For that purpose, I created a Python script for each of the folders:

Figure 6.Copy of Data Augmentation

3 Retraining a Model (Transfer Learning)

After completing the first data collection process, we need to train that model where we have 2 possibilities:

1. Training from the beginning (to select the appropriate parameters (weights, filters))
2. Transfer-Learning

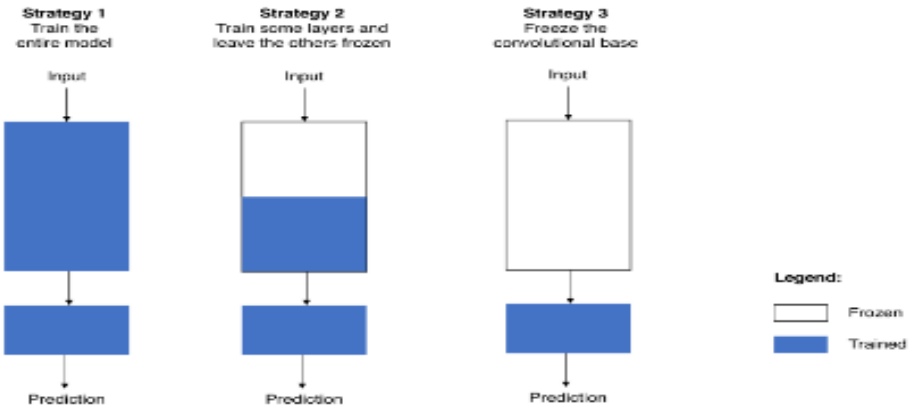
In this case we have a small amount of data so instead of training from the beginning or finding the appropriate parameters for our model we can use an already trained model with all the parameters and replace only the last layers of that model using a technique called Transfer Learning. Thanks to very large databases such as ImageNet on which different models have been created using different CNN architectures such as Inceptionv3, MobileNet, ResNet, LeNet, etc.

Transfer methods are closely related to the machine learning algorithms used to learn the tasks and can often be considered simply extensions of the algorithms [14].

- **Transfer learning -** what is Transfer Learning and how will we train our model. Transfer learning is a process that reduces our training time by using an already trained model like Inceptionv3 on the ImageNet database. This process is shown in more detail below:
  We use a transfer learning technique which means we start training our model from a previously trained model that is used for a similar purpose because training from the beginning requires a lot of time as well as GPU power.

In Computer Vision, transfer learning is usually expressed through the use of pre-trained models, a pre-trained model is a model that has been trained in a large database (ImageNet, MNIST, CIFAR- 10, MS-COCO etc.) which solved a problem similar to the one we want to solve, we can use an example for other problems but we will have to train the model and feature extractor and the classification part from the beginning. With transfer learning we take the shell of an architecture that gives a good result in classifying a particular problem.

3.2 Transfer Learning Options:



1. Model training from the beginning - In this case we use only the model shell (Pre-trained model) and train according to our data, which means model learning, we will need to have a lot of data as well as test resources and training (GPU, TPU).

2. Training a few layers (50%) and freezing the other layers - As was mentioned in the CNN section, the lower levels learn the general characteristics; in contrast the higher levels find the specific characteristics of the problem.
In this strategy we will use some layers from the first part of the architecture, we leave the others frozen, i.e. under frozen it is understood that they cannot be changed; we leave those parameters as they are. For example, if we have little data and a large number of parameters then we can leave more layers freezing to avoid overfitting.
On the other hand, if we have a large database, that number of parameters may be small and underfitting may occur, so we will use several layers from the first part (Convolutional base) for that data to be processed properly.

3. Freeze the first part of CNN - In this case we freeze the first part and just give it data from the database to replace only the second part to classify our data, this approach is good if we have a small database (class <1000 images) or we do not have the resources (GPU), if we try with the CPU it will take a lot of time to train the model.
In the first 2 strategies care should be taken in setting the learning rate parameters, it is a hyper-parameter that controls the frequency at which to replace the values of the weights in the neural network. Setting this parameter is recommended to be small because problems can occur when optimizing one of the data optimization algorithms (gradient descent).

Transfer learning process (rules to follow):
1. Selecting a pre-trained model - nowadays we have many models that we can access such as Inceptionv3, VGG16, MobileNet, and ResNet.
2. Classification of the problem on a similar problem, ie in what conditions, does the image classify small things, etc. - in this step we should pay attention to the conditions we have, for example large on the base, similarity of our problem with the module we use.
3. Fine-Tune selected model - in this step we decide which of the strategies listed above we will use for our case depending on what we have.

4 INCEPTION V3

In some of the older or basic CNN models we needed the size of the kerner filter after each layer of CONVOLUTION we had to choose the size of that filter or if we want a pooling layer at some point, it is not the same in the Inceptionv3 model in this model all the most used dimensions of the kernel filter above in the image are shown if one layer is processed.
By choosing the size of the kernel filter in previous CNN architectures in different cases we could lose significant information in one image, for example if we use a small kernel then we will find things that are local or within that kernel filter, on the other hand using a large kernel filter we will find the global details in one image so that's why the Inception model uses them all.

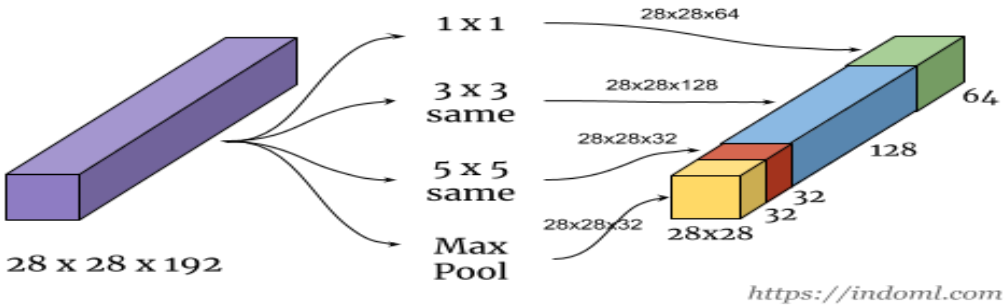


Figure 8. Basic Inception module how it works

We convolve the whole matrix with a 1x1 kernel filter we get a block of size 28x28 with a number of filters 64, then with 3x3 we get 28x28x128, then 5x5x32, in maxpooling we add padding to get the same size of that block of filters we get 28x28x32.
We have input 28x28x192 as output we get 28x28x256.
The problem in this case is Computational cost, i.e. a large number of processes when using 5x5 or 3x3 if we calculate at 5x5 kernel with 32 filters required processing is 120 M.
That number can be reduced up to 10 times which we will explain in a picture with the Inception module.
First, we will explain why 1x1 is used before 5x5 or 3x3, if we use 5x5 directly it takes a lot of processing up to 120 m just for this kernel filter, to reduce that number as the first convolution with 1x1 convolution data: f = 16, s = 1, nF = 1x1x192, as output we get 28x28x16 which has a much smaller volume than 5x5, then we convolve it with 5x5 the same.
1x1 - also called bottleneck layer this can be explained with a bottle, the part where the bottle is narrowest can be called a bottleneck i.e. before the water flows the upper part to have a smaller amount of leakage.
Result of the necessary processing: 28x28x16 x 1x1x192 = 2.4M for the first layer, for the second layer 28x28x32 5x5x32 = 10.0 M which means from 120 M to 12.4 M.
Explanation of the second image or part of the Inception model, suppose that as input we have activation (result) from a previous layer Activation size 28x28x192, I previously explained one layer in the case 1x1 with 5x5 where 1x1 in this case has 16 kernel filters and 5x5 has 32 kernel filters.
The same process is repeated in the case of 3x3 only. We have a different number of kernel filters for 3x1 96 and for 3x3 we have 128.
It is also convoluted with 1x1x64 where there is no need to convolve twice with 1x1 as it was with 3x3 and 5x5.
The last convolution is MAXPOOLING layer, we add a padding to have the same output with the input in terms of dimensions p = same, just like the previous cases with 5x5 and 3x3 here we will convolve with 1x1 because at maxpool we get the same size as the input (1x1) the same so there are a large number of filters (192), after convolving an example we get 28x28x32 it is

done using nF = 32 filters in dimension f = 1x1x192, the last layer of a CONVNET is Channel Concat means adding the number of filters (channels), as output we get 28x28x256 same case as in the first picture.
But even though the number of processes was reduced by 10 times, that model was very deep, i.e. it had many layers, with overfitting often appearing.
That's why the researchers who designed Inception want to improve that model by reducing the layers inside by designing Inception v2 and Inception v3 at the same time.
What has changed since Inception V1:?

- Factoring 5x5 Conv into two 3x3 conv to improve computing speed, even if this seems illogical, but 5x5 requires 2.78 times more computing power than 3x3.

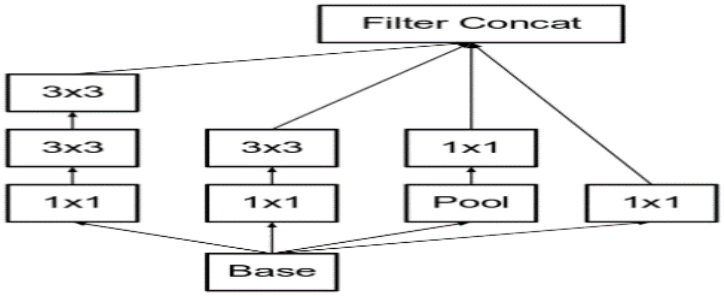


Figure 9. Inception v2 first version

Then they added two more filters after 3x3 by noticing a 33% performance improvement, they added two parallel kernel filters after 3x3 by doing this once for 1x3 then for 3x1 for output.

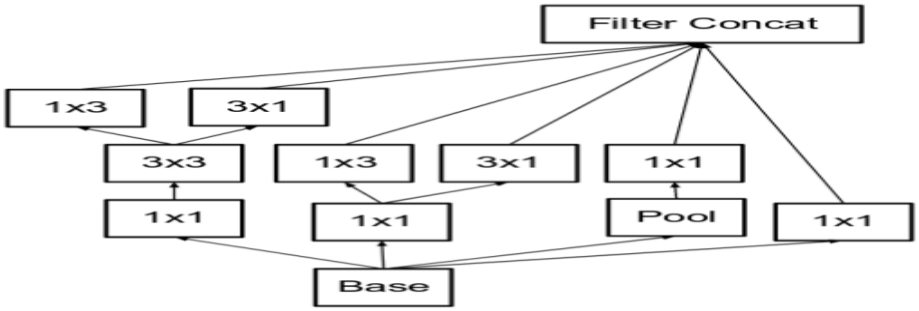


Fig 10. Inception v2 second version

The actual Inception v3 architecture is similar to v2 the only difference is that after each kernel filter (conv) we add batch normalization - a parameter that will normalize the values in the matrix obtained in a certain range as the final result improves.
Also add a few other techniques to improve that model:

- RMSProp optimizer
- Batch normalization
- Label Smoothing - as a regulating component that is added to the tooth measuring function.
  With this technique we will overcome the problem of overfitting by the fact that this technique does not allow the network to be very reliable in classifying certain classes (we do not get results like 98% plus but it is good because we know that in our case it does not occur overfitting).

How to create this model, this model is created with a set of multiple modules only the input has different blocks and the output where we have different blocks such as: flatten, Fully-connected, SoftMax.

It is also different that we have other branches in the model that can be classified from the inner layer before the model reaches the final process, i.e. that branch creates Fully-connected layers also SoftMax. These branches overcome the problem of overfitting.
Flatten - is a process by which at some point after a few layers (hidden layers) we need to reduce that image to *n x 1* size affected for the SoftMax output, followed by Fully-connected.
Fully Connected - is a layer that is composed of *n x 1* or scalar in TensorFlow is called tensor, we can have several such nodes which are composed of a number of neurons that are connected to each other one neuron from FC1 connects with all of FC2 Also with this layer some of the neurons that have a weak connection are rejected.
The task of this layer is to classify depending on how many classes we have in my case 12 and in ImageNet there are 1000.
SoftMax - a function used to convert the results of the classification into probabilities, i.e. if we have a result for 1option 2.0 for 2option 1.0 and for 3option 0.1 to convert them into probabilities for example 0.7 for 1option, 0.25 for 2option, 0.05 for 3 options in order to get numbers from 0 to 1.
When to use SoftMax? SoftMax is most often used when we want to classify into several classes, i.e. the results of SoftMax activation depend on each other, on the other hand sigmoid is used when as output we want only one result that is not dependent on others and the result is between 0 and 1.
When to use SoftMax when sigmoid? Both can be used in multi-class or binary classification but when we talk about efficiency sigmoid is used more in binary on the other hand SoftMax is used in multi-class classification.
In the retraining process of the model - it is important to specify some parameters such as learning rate which is used to define how big our step will be (meaning setting the weights) to the goal to reach Convergence; this term is when the lowest point of a gradient descent will be reached whether it was local or global.
Another important parameter is also defining how much of the images (data) will be used for training, how much for testing and how much for validation in my case I decided on the following parameters:
Learning rate 0.001
train_batch_size 96 images per iteration for testing and validation I used a value of -1 which means to test and validate on all images normally this technique is more efficient but requires more training time.
Python retrain.py

```
--image_dir =. / training_data
--tfhub_module = https://tfhub.dev/google/imagenet/Inceptionv3/classification/1
--bottleneck_dir = / PlantDiseaseFinal / bottlenecks
--how_many_training_steps = 10000
--saved_model_dir = / PlantDiseaseFinal / models /
--summaries_dir = / PlantDiseaseFinal / training_summaries /
--output_graph = /PlantDiseaseFinal/retrained_graph.pb
--output_labels = /PlantDiseaseFinal/retrained_labels.txt
--final_tensor_name = final_result
--learning_rate = 0.001
--train_batch_size = 96
```

With the above commands we will train only the last layer of the CNN architecture trained on the ImageNet database which as I mentioned has over 1000 classes. We use the parameters of that model only when the last layer of data (Disease Identification) is added which will enable good classification for fast execution time.

The ImageNet database does not include any of these types of plant diseases that we train, but it does allow us to classify other objects. In my case the last layer will have 11 classes.
Bottleneck are created for that purpose if I want to train again on the same data next time there will be no need to re-read the images on which the text file bottleneck was created, i.e. those images will be laminated for reuse and faster processing.
They are also useful because during the training process one image is used many times, which saves us time because during testing or validation it has created bottleneck files (transfer-values). This access is enabled because we do not touch the previous layers, we use the parameters and hyper-parameters of the model, and we can cache and reuse the data.
After completing the process of creating bottleneck files, the training of the last one (classification layer) begins.
In this project, 10,000 iterations were performed. In each iteration, the algorithm selects 10 images at random from the training data, finds the corresponding bottleneck from the cache and uses them in the last layer to classify.
This classification is then compared to current labels and the results of this comparison are used to update the weights of the last layer through the backpropagation process algorithm.
As we mention, the model goes through 10,000 iterations and after each iteration we get data on the **accuracy of the training,** i.e. it shows us the percentage of how many of the images used in a batch were successfully identified on their classes and labeled on that class: average accuracy 97-98%
Another concept on which we receive data and is important is **accuracy in validation,** i.e. validation or testing means using data that is not processed in the training process and we want to try how our model relates to data that it has not seen.
Validation average - 95 - 96% of the graph, but tested in application on average 70-80%
Cross-Entropy (Cost Function) - is an identifier that shows us how the learning process goes, this process shows us what is the difference between the input and the true value, which means that the input is from the test / validation set and the true value is processed in the training set (labels) it is important that the smaller the number we get the less loss we have which means it is better.
Minimum in my case: 0.070542
In Keras API we define optimizer and loss functions with which after each iteration we will optimize the weights and measure the loss of the model, in terminology these two things cover a wide range of thinking but in Keras it is very simple with just one line of code will we optimize weights, but in any case, we first need to understand how these functions work in terminology to optimize them in code.
The picture above shows the progress of the model precision during training. The X axis is for the number of iterations and the U axis for the accuracy of the model.
We have 2 lines, the orange line shows us the accuracy in training and the blue shows us the accuracy in validation, why is it necessary to follow this process?
In my case, the process goes as it should, it can be said that it is one of the best cases. There are cases when the line or the training process increases all the time but the validation accuracy decreases, that process is called "overfitting".
Final test accuracy = 98.1% (N = 1253)

4.1 Overfitting and Underfitting: As we mention 2 problems can occur during training - overfitting and under-fitting, it is important to be able to eliminate these 2 problems by well defining the data, creating an architecture that is appropriate for the data we have available and also well-defined parameters.

Under-fitting - a problem that occurs when the model is not effective in training, this can happen when the model is not powerful, i.e. the architecture is not suitable for the data we have, or is not trained long enough to learn all the paths, i.e. we have a small number of neurons.
This problem in classification is also called high bias - is the difference between the result obtained and the actual result and is therefore called under-fitting because the input is far from the true value.
On the other hand, overfitting occurs when we train a lot of time and will learn paths that we will not generalize in testing.
In the diagram above we can see how the validation loss in the first iterations is drastically reduced or normal reduction because the goal is to reach the smallest loss, but in the diagram, we can see that after a few iterations the training loss decreases normally, but the loss of validation increases this is called overfitting, why does this happen? Because the model specializes in classifying training data, it teaches paths in detail and we will not have a good result in testing or validation, or it can be said that it memorizes test data but does not generalize validation data.
Example: testing 3 models of different sizes this is for a small number of inputs:
1. Small model 3x DenseLayers (feature extractor), 1 x classification (sigmoid) (4 neurons, rELU)
2. Medium model 3x DensLayers (feature extractor), 1 x classification (sigmoid) (16 neurons, rELU)
3. Big model 3x DenseLayers (feature extractor), 1 x classification (sigmoid) (512 neurons, rELU)
In the picture we can see that the test loss in the medium and large model decreases faster than the small model, but unlike the validation test data in the large model the validation jumps up because the more neurons we have the easier overfitting can occur. Of the training data by the fact that the model will not work properly when validating that training data. But at the same time if we have a few neurons the model may not learn the tracks well in training and not be effective in validation is called under-fitting, it is always good to find balance.
Overcoming Fitting:

- Increasing the number of data
- Data Augmentation
- Batch normalization (To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation)
- Using efficient architecture
- Adding regularization (defining weights to move in a small range of numbers example -0.5 to 0.5)
- Add Dropout

The most commonly used technique to overcome overfitting is regulation. With this technique we penalize theta parameters so that they can be smaller to get a simpler line (hypothesis).

5. Backpropagation and Forward Propagation algorithms

The whole process of a neural network is "Enter multiply by Weights, add BIAS, activate at some point"

5.1 Forward Propagation: We explained this process indirectly when we talk about neural networks where we had input or activation from one neuron by calculating the neuron and we get the activation for the next neuron, the same is repeated with forward propagation.

 # 1
$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} * a^{(1)}$
$z^{2} = [z_0^2\ z_1^2\ z_2^2\ z_3^2\ ]$these values are obtained from neuron activation calculations example
($n^{(2)}_1 = g\ (\Theta^{(1)}_{10} x0 + \Theta^{(1)}_{11} x1 + \Theta^{(1)}_{12} x2)$)
$a^{(2)} = g\ (z^{(2)})$ (add bias $a^{(2)}_0$).
 # 2
$z^{(3)} = \Theta^{(2)} * a^{(2)}$
$a^{(3)} = g\ (z^{(3)})$ (add bias $a^{(3)}_0$).
 # 3
$z^{(4)} = \Theta^{(3)} * a^{(3)}$
$a^{(4)} = h\Theta\ (x) = g\ (z^{(4)})$
Notation:
$a^{(l)}$ - means this is the activation or in English "Fire" of a neuron in random order.
$\Theta^{(j)}$ - with this symbol we mean the parameters (weights) that are used to activate neurons.
$z^{(n)}$ - vectorization of calculations between input and weights for easier access when using the activation function (sigmoid or softmax).

Let's explain the process above:
The processor above is if we have only one example of training then I will explain it in the simplest way
First of all, in that network of neurons we need to apply forward propagation which will get a certain value for the hypothesis (line) which we will compare with the desired output or the actual value.
a (1) is the first layer of the network i.e. the input used notation for the input is (x), then we will calculate z (2) which is calculated by activating a neuron a (1) multiplied by the weights (the size of the weights is obtained in the following way if for example the input has 3 neurons and the other layer we need to get has 3 neurons then the size of the matrix for the weights will be 3 + 1 X 3 = 4 x 3), a (2) i.e. the second layer of the neural network where depending on which function we use whether sigmoid or softmax we perform that function on z (2) on the results obtained from a (1) * $\Theta^{(1)}$ and we get that layer a (2) we add a bias which always has a value of 1.
The same process is in all other hidden layers until we get to the exit from the hidden layer in the output layer.
That layer is equal to the output or value of the hypothesis calculation function.
After the obtained result, we use the backpropagation algorithm to estimate the loss and optimize the weights.

5.2 Backpropagation & Gradient Descent: Backpropagation is a way to open a wide variety of neural networks. It has triumphed all over the world and proved capabilities of the new information processing technology based on neural networks [15].
Before explaining how the backpropagation algorithm works, we first need to have an idea of how the Gradient Descent works.
Gradient descent is a method used to minimize the loss; the loss is equal to the true value in the base with the value we get from the training of the neural network.
The simplest Gradient Descent can be explained as follows:
We will take it as an example if we are in the mountains, we have visibility only a few meters, our goal is to reach the lowest point, i.e. at home, how will we assess which path we should follow to get home, we will normally go down step by step until we reach the lowest point, the

same thing happens and the gradient descent by performing certain mathematical functions approaches the goal, the goal is to reduce the loss in small steps towards the goal.
In this case we have only one minimum, which in neural networks is called the global minimum.
The part described above for GD applies if we have only one function, i.e. for one neuron (or in neural networks (nodes)).
But what if we have more functions (nodes) to perform?
In that case we should be careful to perform all the functions and find which of the n-minimum is the best or which of the minimums has a smaller loss, in this case we have more local minimums.
We use the learning rate to determine the size of the step we want to take to the goal, in this case we need to be careful in deciding the size of that learning rate because if that value is too high we can jump from mountain to mountain and to never find the minimum, it is also not okay for that minimum to be very small because performing a gradient descent will take a long time to reach the goal.
The back-propagation algorithm is similar to the forward algorithm but the execution of the functions is performed from right to left.
Backpropagation is a terminology in neural networks that is used to minimize loss (Cost Function).
$\min \Theta J(\Theta)$ - we want to find the minimum of the function J (Θ) by optimizing the parameters for each iteration, i.e. optimizing theta (weights) parameters.

1. Let's calculate the loss (Cost Function)
2. To calculate partial derivatives with the loss (partial derivative)

How this algorithm works:

The name itself tells us that we start from the end or the last layer (output layer) and iterates in each node (nodes) by calculating the derivatives of the loss function $\boldsymbol{\delta^{(l)}_j}$ where we compare two values or the value obtained (from forward propagation) and the desired value, when we subtract them we will find **error** ($\delta^{(l)}_j$) in that neuron this only applies to the last layer.
The formula for finding that error is simple ($\delta^{(L)} = a^{(L)} - y^{(i)}$) example: $\delta^{(4)} = a^{(4)}_1 - y^{(i)}$)
$a^{(l)}_j$ - activation of j (neuron) in l (layer).
To represent Error, we use the following notation $\delta^{(l)}_j$ we do this for $a^{(l)}_j$ (in each layer and in each neuron).
The purpose of the backpropagation algorithm is to compare the result obtained by forward propagation with the actual result we expect from y (1). After the obtained result, the algorithm proceeds in reverse from forward propagation by calculating the derivatives of each neuron.
Derivate - is the tangent to a point in a graph or model.
Formula:
$\delta^{(l)}_j = \Theta^{(j)}_I * \delta^{(l)}_j$ -> the error is equal to all the weights associated with that neuron multiplied by the neuron from which the example comes: we want to calculate for $\delta^{(3)}_2$ neurons that we assume is located in the last hidden layer and the output layer has only one result $\delta^{(4)}_{1.,}$ the only neuron that connects to that neuron.
$\Delta^{(3)}_2 = \Theta^{(3)}_{1,2 \text{ (from which neuron to which neuron)}} * \delta^{(4)}_1$
But what if we just calculate the neuron derivatives without optimizing the weights to improve the result, how do we optimize them, and are there any weight optimization rules and which weights have the most impact on the neural network?
As I said when we go back to the neural network we will also have to optimize the weights at the same time we will have to use a formula to find those neurons that have the largest contribution in that connection, for example if we have w11 = 6.0 and w21 = 3.0 then how we will divide the error, using the following formulas:

W11 = w11 / w11 + w21 = 6/6 + 3 = 6/9 = 2/3 - we will optimize this neuron by 2/3 of the whole E in the last layer of the hidden layer.
W21 = w21 / w21 + w11 = 3/3 + 6 = 3/9 = 1/3 - we will optimize this neuron for 1/3 of the whole E in the last layer of the hidden layer.
Other weights in the interior are different.
Important things to keep in mind when implementing the BackProp algorithm:

1. The value of the weights must be greater than 0 because if it is zero then this algorithm will iterate but without result i.e. we will get the same result in each iteration.
2. In $\delta^{(1)}$ this algorithm does not apply because it does not make sense to change the input values.
3. We can never be sure that this algorithm works properly even though we get results that look fine, i.e. the loss is reduced but we can have bugs that can be overcome by using the gradient checking technique (Gradient Checking)
4. After performing this algorithm, it optimizes the values of the weights by reducing the "ERROR" in the next step.
5. Both algorithms are executed for one example, i.e. a pair of (x (i), y (i)) we read that data with a for loop on the training data and we iterate on all pairs of data.

Conclusion

To learn something from the beginning, we must explain the basic concepts of this work. In the first chapter, we learned the main concepts that are the introductory part of the methodology of learning about the impact of artificial intelligence in finding plant diseases.
To develop efficient software, you need to collect data. We learn the types of data as well as the approaches to collecting that data, after learning what data we have and what approaches we can use depending on how they are organized or how we will organize the data we have to decide which of the 3 approaches we will use whether supervised or unsupervised approaches. In my case, I used supervised because that's how data is organized. After collecting the data, the training phase begins from the beginning or using transfer learning. For this reason, we explained transfer learning where we must follow some rules that determine the approaches to be taken depending on the data we have. We also explained the problems that arise during training and how to find those problems, problems as overfitting and underfitting. We finished this paper with the basic structure of a neural network, we explained details inside a neural network, which mathematical formula is performed to get the result of the next neuron, what are those weights, and what rules should we follow to determine these weights. This approach is called forward propagation. Algorithm for optimizing weights and measuring loss for each neuron. The work is based on an experimental application to detect plant diseases. The database is taken from https://www.kaggle.com/ to experiment with the application.

References

[1] Wolfgang Ertel, Introduction to Artificial Intelligence Second Edition 2017
[2] Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." Science 349.6245 (2015): 255-260
[3] Sammut, Claude, and Geoffrey I. Webb, eds. Encyclopedia of machine learning. Springer Science & Business Media, 2011.
[4] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.
[5] Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural Networks 61, 85–117.
[6] LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444.

[7] Abdulrahman Yarali, Applications of Artificial Intelligence, ML, and DL p.279 - 297, 10.1002 / 9781119685265.ch16 2022
[8] Mohanty, Sharada P., David P. Hughes, and Marcel Salathé. "Using deep learning for image-based plant disease detection." Frontiers in plant science 7 (2016): 1419.
[9] Tom Michell Jaime Carbonell Ryszard Michalski, An overview of Machine Learning, 1983
[10] Keiron O'Shea and Ryan Nash, An Introduction to Convolutional Neural Networks, arXiv:1511.08458v2 [cs.NE] 2 Dec 2015 p.3
[11] Voulodimos, Athanasios, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. "Deep learning for computer vision: A brief review." Computational intelligence and neuroscience 2018
[12] Baumgart, Bruce G. "A polyhedron representation for computer vision." *Proceedings of the May 19-22, 1975, national computer conference and exposition*. 1975.
[13] Cunningham, P., Cord, M., Delany, S.J. Supervised Learning. In: Cord, M., Cunningham, P. (eds) Machine Learning Techniques for Multimedia. Cognitive Technologies. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-75171-7_2, 2008
[14] Lisa Torrey and Jude Shavlik, Transfer Learning, USA University of Wisconsin, Madison WI 2009
[15] Shun-ichi Amari, Backpropagation and stochastic gradient descent method Author links open overlay panel, September 1992
[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biology, vol. 5, no. 4, pp. 115–133, 1943.
[17] Nikolaos Doulamis, Anastasios Doulamis and Eftychios Protopapadakis, Deep Learning for Computer Vision: A Brief Review, Published 01 Feb 2018
[18] Y. LeCun, B. Boser, J. Denker et al., "Handwritten digit recognition with a back-propagation network," in Advances in Neural Information Processing Systems 2 (NIPS*89), D. Touretzky, Ed., Denver, CO, USA, 1990.
[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[20] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527–1554, 2006.