ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE (DEI)

AUTOMATION ENGINEERING (L)

TESI DI LAUREA

in
[Visione Artificiale]

AN ANALYSIS OF TRAINING A DEEP OBJECT DETECTOR WITH SYNTHETIC IMAGES

CANDIDATO
Mattia Zoffoli

RELATORE:
Prof. Luigi Di Stefano

CORRELATORE/CORRELATORI
Ing. Pierluigi Zama Ramirez

Anno Accademico [es.:2018/19]

Sessione [II]

Contents

Abstract **5**

1 Introduction **6**

2 Machine Learning **8**

2.1 Definition and working principle 8

2.2 Typologies of machine learning problems 9

2.2.1 Supervised Learning 9

2.2.2 Unsupervised Learning 9

2.2.3 Reinforcement Learning 10

3 Machine Learning Algorithms **11**

3.1 Linear and Polynomial Regression 11

3.1.1 Cost Function 11

3.1.2 Gradient Descent 13

3.1.3 Feature Scaling 15

3.2 Logistic Regression 16

3.2.1 Binary Classification 16

3.2.2 Multiclass Classification 18

3.3 Stochastic and Mini-Batch gradient descent 19

3.4 Underfitting and Overfitting problem 20

3.5 Regularization 21

3.5.1 Regularized Logistic Regression 22

3.6 Train, Validation and Test Sets 22

3.7 Diagnosing and solving Bias and Variance errors 23

3.8 Skewed Classes: Precision, Recall and F_1 Score 24

3.9 Decision Tree 26

3.9.1 Gini Index 27

3.9.2 Chi-Square 27

3.9.3 Information Gain 28

3.9.4 Prevent Overfitting of Decision Tree 29
3.9.5 Advantages and Disadvantages of Decision tree 29
3.10 Support Vector Machine 30
3.11 K-Means Algorithm 32
3.11.1 Choice of the number of Clusters 32
3.12 Principal Component Analysis 34

4 Artificial Neural Networks 36
4.1 Definition of Artificial Neural Networks 36
4.2 Structure of Artificial Neural Networks 36
4.3 Feedforward and loss Function 37
4.4 Backpropagation 38
4.5 Softmax Function 39
4.6 Dropout 39
4.7 Convolutional Neural Networks 40
4.7.1 Architecture 40
4.7.2 Layer Types 41
4.7.3 Backpropagation 43

5 Object Detection 44
5.1 Evaluation Metrics for Object Detection 45
5.1.1 Intersection over Union 46
5.1.2 Mean Average Precision 47
5.2 Object Detection Framework 49
5.2.1 Region Proposals: from R-CNN to SSD/YOLO 49
5.3 Main Object Detection Datasets available 52

6 Experimental Results 53
6.1 Objective 53
6.2 Training Data Organization 53
6.3 Experimental Procedure 54
6.4 Results 54

7 Conclusion 58

Bibliography 60

List of Figures

2.1 Typical Machine Learning Algorithm 8

3.1 Linear Regression example 12
3.2 Gradient Descent Intuition. 13
3.3 Gradient Descent Initialization problem. 14
3.4 Feature Scaling 15
3.5 Mean Normalization 16
3.6 Most Common Activation Functions 17
3.7 Cost Function for Logistic Regression 18
3.8 Multiclass Classification with One-vs-All approach. 19
3.9 Batch, Stochastic and Mini-Batch Gradient Descent. 20
3.10 Underfitting, Optimal and Overfitting models. 20
3.11 Graphical representation of Bias and Variance relationship. 21
3.12 Cost Function as function of the complexity of the model. 23
3.13 Cost Function as function of the number of training examples. 24
3.14 Possible prediction outcomes (TP, TN, FP, FN). 25
3.15 Decision Tree logic scheme[2]. 26
3.16 Decision Tree problem example. 26
3.17 Gini Index calculation example. 27
3.18 Chi-Square calculation example. 28
3.19 Information Gain calculation example. 28
3.20 Support Vector Machine working principle. 30
3.21 SVM: higher margin or better performance?. 30
3.22 SVM: managing outsiders. 31
3.23 SVM: Kernel Function. 31
3.24 Clustering Algorithm working principle. 32
3.25 Elbow Method example. 33
3.26 Evaluation of the effectiveness of Clustering process. 33
3.27 Principal Component Analysis 2D intuition. 34

3.28 Principal Component Analysis example of Variance plot. 35

4.1 Artificial Neural Networks base structure 37
4.2 Feedforward propagation of inputs. 38
4.3 Feedforward and Backpropagation in a single Neuron. 39
4.4 Dropout graphical representation. 40
4.5 Digital representation of images. 40
4.6 Structure comparison between FC-ANN and CNN. 41
4.7 Convolution Layer working principle. 42
4.8 Max-Pooling technique example. 43
4.9 Complete Convolutional Neural Network example. 43

5.1 Difference between Image Classification and Object Detection[3]. . . . 44
5.2 Ground Truth Bounding Boxes example. 45
5.3 Bounding box coordinates example (1024x768 pixels image). 45
5.4 Intersection over Union (IoU) definition and examples. 46
5.5 Precision-Recall relationship. 47
5.6 Average Precision (AP) Representation example. 47
5.7 Mean Average Precision (mAP) example. 48
5.8 Example of mAP results some CNNs on COCO dataset. 48
5.9 R-CNN architecture[5]. 49
5.10 Fast R-CNN. 50
5.11 Faster R-CNN architecture. 50
5.12 Comparison between different algorithms'speeds. 51
5.13 Fast R-CNN. 51

6.1 AP Experimental Results. 55
6.2 IoU Experimental Results. 55
6.3 mAP and IoU numerical results. 55
6.4 Visualization result example (1). 56
6.5 Visualization result example (2). 57

Abstract

L'obiettivo di questa tesi è di analizzare i risultati ottenuti da una **rete neurale** (Single Shot Detector) allenata per il **riconoscimento di oggetti**, utilizzando immagini **reali** (KITTI) e **sintetiche** (Playing For Benchmarks).
In caso di risultati positivi, questo particolare esperimento nel campo della **visione artificiale**, può semplificare l'allenamento di reti neurali, in quanto la creazione di set di immagini etichettate (**dataset**), risulterebbe operativamente molto più rapida a partire da immagini sintetiche, piuttosto che da immagini reali.
La creazione manuale di dataset richiede la supervisione umana per la generazione di **bounding boxes** che identifichino gli oggetti presenti nelle immagini, mentre la creazione di dataset a partire da immagini sintetiche risulta molto più semplice, avendo l'accesso al codice sorgente dell'immagine, che contiene già al suo interno le informazioni sulla posizione degli oggetti generati.
Per questo motivo, in applicazioni dove i dataset disponibili sono molto piccoli o inesistenti, la generazione di un dataset digitale potrebbe rivelarsi conveninente rispetto alla creazione o estensione di un dataset costituito da immagini reali.

Chapter 1

Introduction

The first chapters of this thesis will analyze deeply the most important machine learning algorithms developed and employed in a potentially limitless number of applications, starting from **linear** and **logistic regression**, which prove particularly useful to understand the problems that can be solved with machine learning.

The second topic will be **artificial neural networks**, which represent the most common non-linear machine learning algorithm used when facing complex problems such as **image processing**, **speech recognition** and generally **forecasting** applications in a countless number of fields.

The third part of the thesis will introduce the problem and techniques of **object detection**, which is an important branch of **machine vision**, and it will be applied in the experimental part.

In particular, object detection is performed employing **convolutional neural networks**, which process images in a non-linear way, through the process of **deep learning**, where the network learns from some *labeled* training examples (images with bounding boxes that identify the presence of objects).
Many different kinds of convolutional neural networks optimized for solving different problems (good accuracy or fast detection) are already available as open-source software, however the problem that can slow down the training is sometimes the absence or scarcity of labeled images (**dataset** of training images) from which the network can learn.
In such applications, the effort required to create or expand an already existing dataset by taking new images and manually draw some bounding boxes to identify objects can be many times more onerous than the actual training of the network.
In order to avoid this bottleneck in training neural networks for object recognition, an idea would be to employ synthetic images, and generate automatically many labeled images, given the advantage of knowing the source code of the images. As

a matter of fact, the source code generates the objects, therefore it automatically knows their position inside of the image.
The problem of employing synthetic images, is that they do not look exactly as real images, especially because the network is extremely sensible even to small variations of the image, which whould be difficult to detect for human eyes.
The experimental part of this thesis was done in order to evaluate the possibility of employing synthetic datasets to train a network that will be able to detect objects in real images.

Chapter 2

Machine Learning

2.1 Definition and working principle

A rigorous definition of Machine Learning states that: "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[14]
As an example we can analyze a software playing chess:

- E → experience of playing chess games.
- T → task of playing chess.
- P → probability that the program will win the next game.

⇒ if the program is driven by a machine learning algorithm, P (winning probability) will increase proportionally with E (experience).[12]

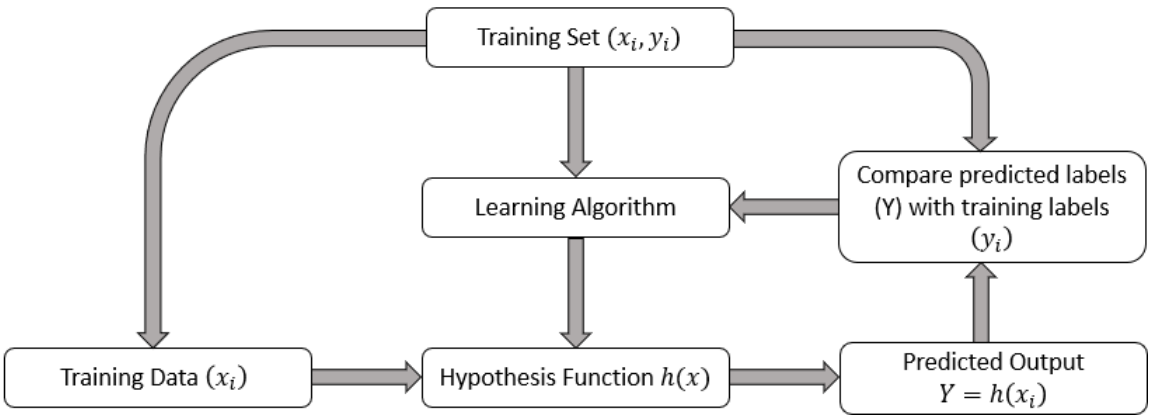


Figure 2.1: Typical Machine Learning Algorithm

Looking at figure 2.1, it is possible to understand the operation of a typical machine learning algorithm.

The input data is made by a **training set** with data already classified according to some specific features (**labels**).
The algorithm will continuously make an *hypothesis* of a function *h(x)* able to connect all the training data to their labels, in order to create a predicting **model** that can be applied also to new (previously unseen) data.
During every **step** of the computing process, the hypothesis of the connecting function will be corrected by the algorithm, and this process will stop after performing a predefined number of steps.

2.2 Typologies of machine learning problems

2.2.1 Supervised Learning

Supervised learning happens when input variables (x) and output variable(s) (Y) are given, and an algorithm is used to learn the mapping function h(x) from the input to the output ($Y = h(x)$)

This process is called supervised learning because it presents some similarities with a teacher supervising the learning process. The correct answers are known and the algorithm *iteratively* makes predictions on the training data and is corrected by the teacher. Learning stops when the students achieve an acceptable level, and from that point on, they will be able to solve on their own the same problem with different (previously unseen) data.

The algorithms analyzed until now and most of the present practical applications, follow a **Supervised Learning** process.

Supervised learning problems can be grouped into:

- **Classification** problems, where the output variable is a category (a *class*), that can assume discrete values, such as detecting spam emails (each element can be classified as spam or not).

- **Regression** problems, where the output variable is a real value, which can assume continuous values, such as predicting the future number of visitors of a website.

2.2.2 Unsupervised Learning

Usupervised Learning is characterized by the absence of output variables (Y).
The learning process is more complex, because the goal of unsupervised learning is to find the underlying structure of the data, starting from *raw* (unlabeled) data.

Recalling the previous example, unlike supervised learning there are no correct answers and there is no teacher. Students are left on their own to discover and organize the answers.

Unsupervised learning problems can be grouped into:

- **Clustering** problems, where the data must be grouped into *clusters*, such as grouping customers by purchasing behaviour.
- **Association** problems, where the aim is to find some rules that describe the majority of the data available, such as finding two objects that are usually bought together on an online platform.

2.2.3 Reinforcement Learning

Another field of application of machine learning is **Reinforcement Learning**, which develops algorithms continuously learning and improving the actions they perform on the environment (e.g. algorithm that learns how to drive a car).
This process is different from both supervised and unsupervised learning problems, because it is applied when there is a goal to reach (which means it is not completely unsupervised), but there are no labeled training samples (which means the problem is also not supervised).
This kind of approach is needed in fields like learning to drive a car (or to fly a plane) autonomously. In this case the system must decide which action to take on the environment, and has the information coming from sensors as input data.
Reinforcement learning algorithms should always realize a trade-off between **exploration** (improving themselves by a trial and error process) and **exploitation** (performing the task with their current ability).

The object detection problem, which is the final part of this thesis, is of supervised learning type, hence from this point onward, mainly Supervised Learning problems and algorithms will be analyzed.

Chapter 3

Machine Learning Algorithms

3.1 Linear and Polynomial Regression

3.1.1 Cost Function

In the case of a supervised learning Regression problem, to measure the accuracy of the hypothesis function h(x) proposed, at every step the **Cost Function** is computed, representing the *error* that the current predicting function is making with respect to the labels of the training data.
The cost function $J(\theta)$ must calculate the mean value of the error, made by h(x) on all m examples belonging to the training set ((x_i, y_i) with $i = [1, 2...m]$), and its expression is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{3.1}$$

By elevating to the power of 2 the single elements of the cost function, it is possible to weight the errors, giving more importance (higher negative contribution) to the elements which are farther from the prediction made.

In equation 3.1, x_i is a vector containing all input features values for one training sample, and y_i is its output label, while θ represents the weight to give to every input feature to obtain a prediction.

The example of figure 3.1, gives a graphical representation of the concepts explained above, in a simple problem of predicting the profit of a restaurant, given the population of a city in which it will be opened.

In this case the length of x vector is simply 1, because there is only one input feature (city population), and we are considering a *linear* function ($Y = \theta_1 x + \theta_0$) as hypothesis, so $\theta(\theta_0, \theta_1)$ will have length 2.
This problem is solved with an algorithm of **Linear Regression**, because the function used to approximate is linear, and the predicted value is a real value (a price).

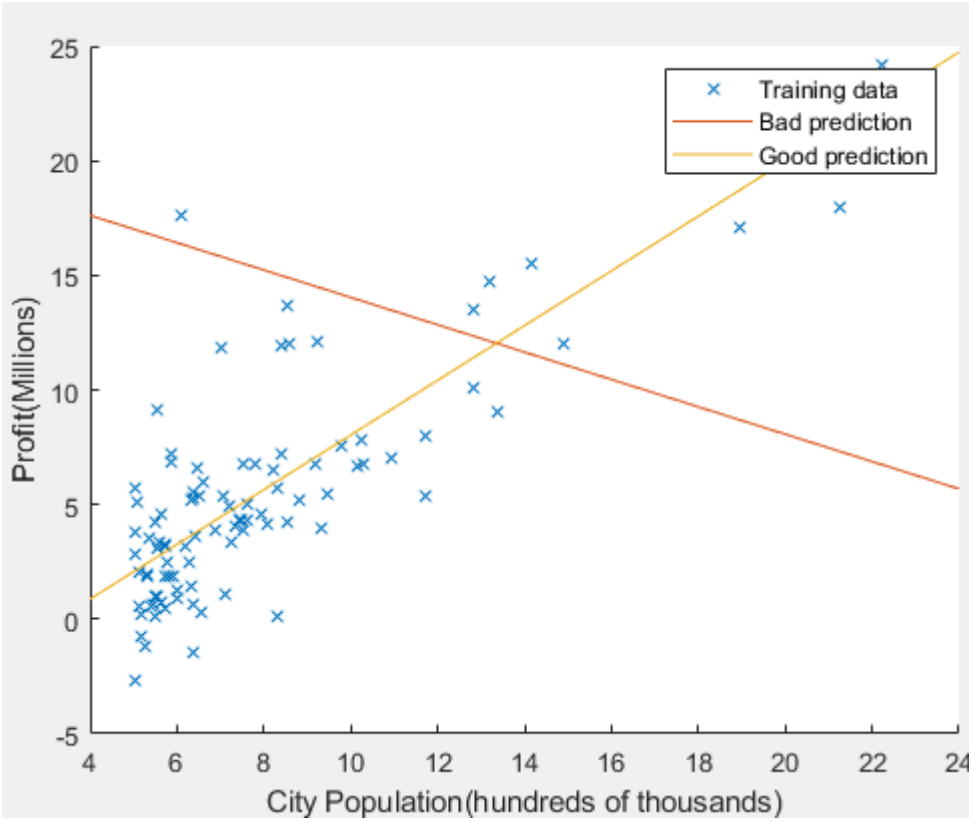


Figure 3.1: Linear Regression example

The data provided can also contain more than one feature that contribute to the predicted output, so for a number n of features, the hypothesis function becomes:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_n x_n \tag{3.2}$$

In this expression x_0 is always 1, and it is the **bias** term of the model proposed, representing the *offset* of the model function.

Polynomial Regression

Sometimes a linear function can be insufficient to model the data provided, and a higher-degree *(polynomial)* function may be required.
In this case, the higher-degree terms are included into the hypothesis function as new features (e.g. $x_{New} = x_1^2$), thus increasing the degree of the regression model, entails the addition of some new features, and the hypothesis function will keep the structure of formula 3.2.

3.1.2 Gradient Descent

In order to optimize the hypothesis function $h(x)$, the values of θ must be updated at every step, shifting in the right direction to make every hypothesis better than the previous one, as shown graphically in figure 3.2.

Gradient Descent is a technique that updates the current values of θ following

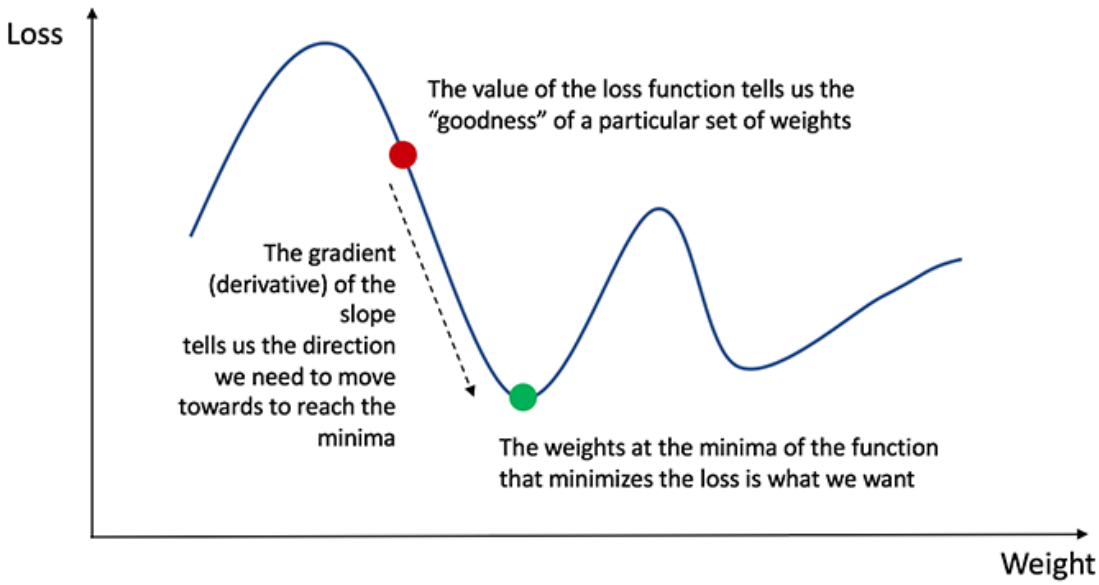


Figure 3.2: Gradient Descent Intuition.

the gradient of the cost function, in order to minimize (get to the local minimum of) the cost function ($J(\theta)$).

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} J(\theta_i) \tag{3.3}$$

Repeating iteratively the updates of formula 3.3 for all the n values of θ, the cost function will eventually reach its *local* minimum, and the hypothesis function will be the best model to represent the training data analyzed, given the particular initialization considered.

The initialization chosen in the beginning for the values of θ can sometimes change the outcome of gradient descent, because, with different initialization, a different local minimum can be reached, as shown in figure 3.3.

Therefore, sometimes the same process can be repeated many times with different initialization, keeping only the best results obtained.

The parameter α represents the **learning rate** of every iteration, and should be chosen making a compromise between small values (slow gradient descent) and big values (risk of no convergence to a minimum value of the cost function).

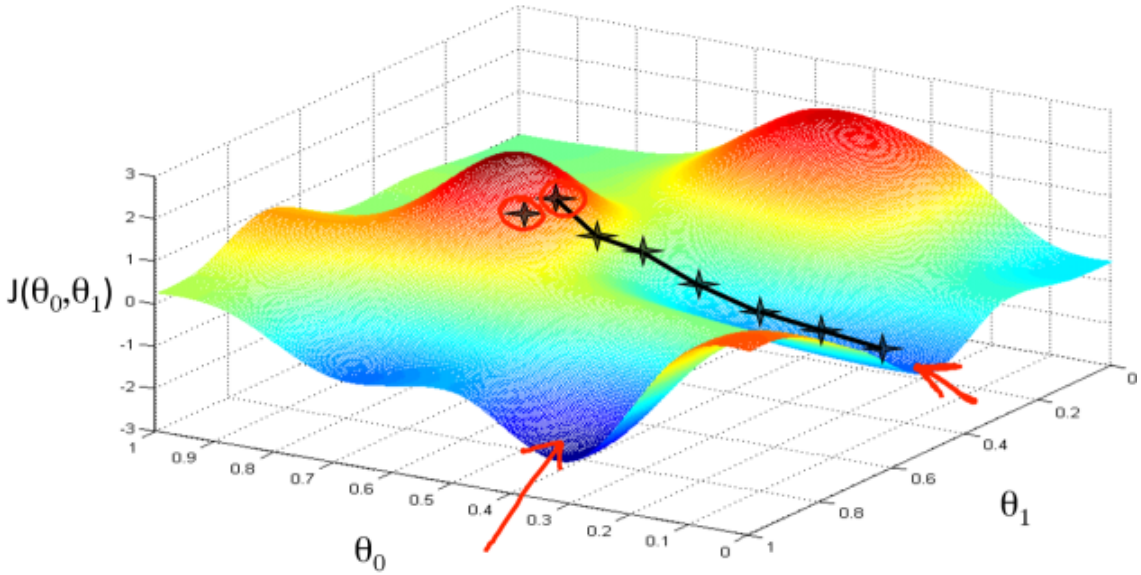


Figure 3.3: Gradient Descent Initialization problem.

In the following calculations, it is shown how it is possible to analytically compute the gradient of the cost function $J(\theta)$.

$$
\begin{aligned}
\frac{\delta}{\delta\theta_j}J(\theta) &= \frac{\delta}{\delta\theta_j}\frac{1}{2}(h_\theta(x)-y)^2 \\
&= 2\cdot\frac{1}{2}(h_\theta(x)-y)\cdot\frac{\delta}{\delta\theta_j}(h_\theta(x)-y) \\
&= (h_\theta(x)-y)\cdot\frac{\delta}{\delta\theta_j}(\sum_{i=0}^{n}\theta_i x_i - y) \\
&= (h_\theta(x)-y)x_j
\end{aligned}
\tag{3.4}
$$

Therefore formula 3.3 can be updated to apply it to real cases without being forced to compute any derivative term:

$$
\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i)-y_i)x_{ij}) \tag{3.5}
$$

3.1.3 Feature Scaling

Sometimes the data provided may be unbalanced: one feature may have a really wide range (e.g. square meters of a house), while another one has a smaller one (e.g. numbers of room in a house).
If the objective is to predict the price of the house, the feature with the wider range weights a lot more on every gradient descent step, and the process will be less effective, as visualized in figure 3.4.

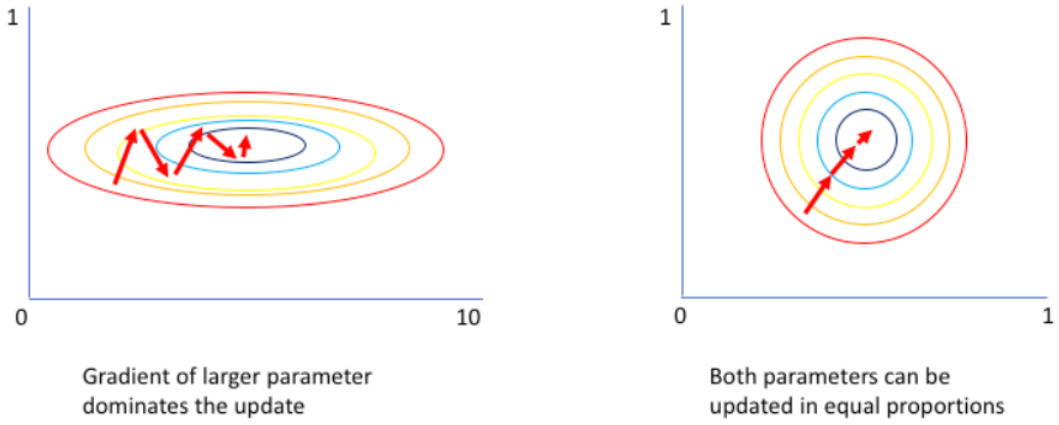


Figure 3.4: Feature Scaling

In order to solve this problem the features should be evenly scaled using **feature scaling** techniques.

Normalization

Normalization is a feature scaling technique that keeps the shape of the data while changing their range in a more balanced way.
Min-max Normalization rescales all features of the training data to fit the interval $[0, 1]$, according to formula 3.6, while **Mean Normalization** rescales them to fit into $[-0.5, 0.5]$, according to formula 3.7.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.6}$$

$$x' = \frac{x - x_{mean}}{x_{max} - x_{min}} \tag{3.7}$$

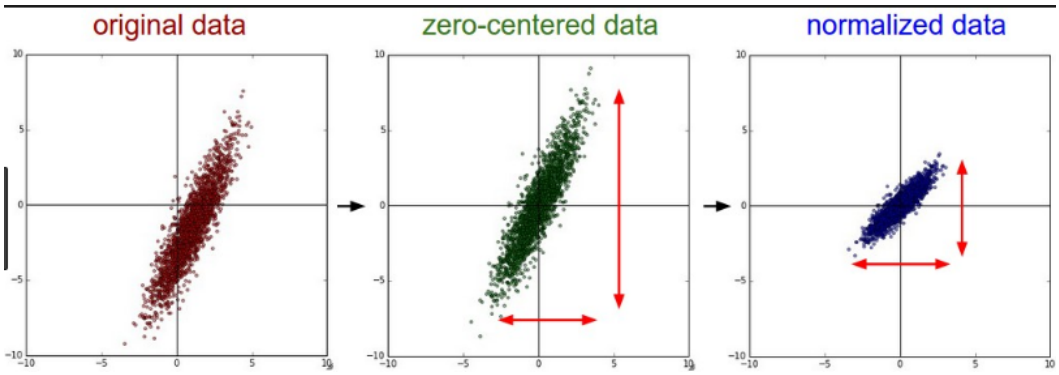


Figure 3.5: Mean Normalization

Standardization

Standardization is a feature scaling technique that makes the values of each feature have mean value equal to zero and variance equal to 1, following formula 3.8 (where σ represents the standard deviation).

$$x' = \frac{x - x_{mean}}{\sigma} \tag{3.8}$$

3.2 Logistic Regression

3.2.1 Binary Classification

A classification problem is similar to regression problem, but the accepted values for the labels ($y = h(x)$) are a small number of discrete values (classes).
At first the case of a simple problem with two possible classes (1 or 0) will be analyzed. These problems are called **Binary Classification**.

Activation Functions

In this case, it is convenient to transform $h_\theta(x)$ such that $0 \leq h_\theta(x) \leq 1$. In this situation the function $h_theta(x)$ will calculate the probability that the output is 1. There are many **activation functions** ($g(z)|z = \theta^T x$) with different characteristics, that are used for this process, as shown in figure 3.6.

The **Sigmoid** function was the first to be used, but it is computationally expensive, and it is not zero-centered, which means that some directions of gradient update are not possible using it.
Therefore $tanh(x)$ was used to solve these problems, but the two functions analyzed

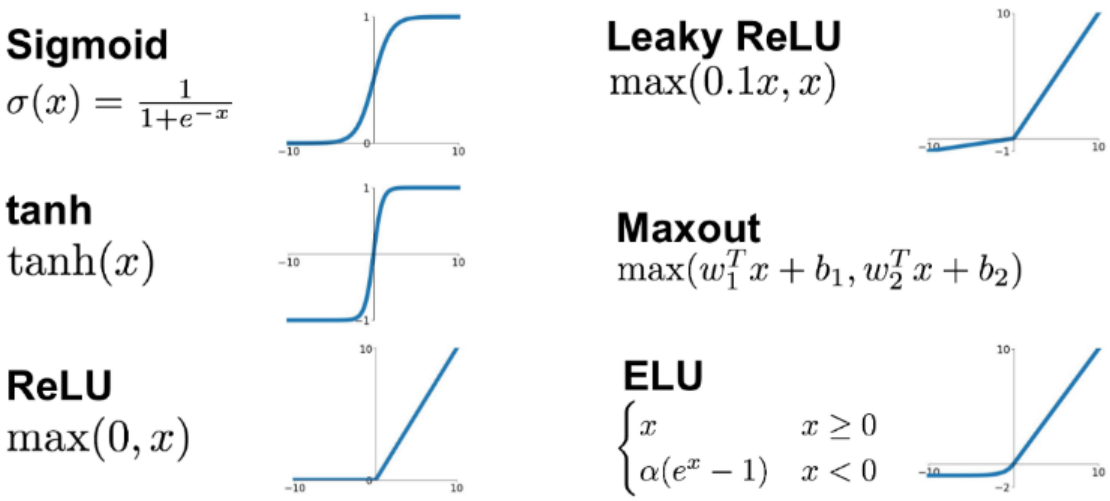


Figure 3.6: Most Common Activation Functions

until now are not used anymore, because they saturate really quickly, and this creates problem in neural networks (some neurons are deactivated, and the gradient is *killed*).

The basic function used nowadays is **ReLU** (**R**ectified **L**inear **U**nit), because it is computationally efficient and solves the bigger problems of its predecessors. **Leaky ReLU** and other more complex kinds of ReLU are computationally less expensive, but they are sometimes used to make the function become zero-centered, otherwise some bad initializations would make it impossible to get a solution.

It is important to note that for machine learning applications the sigmoid function is still used when the algorithm does not employ neural networks, therefore in this section all examples will use it as activation function.

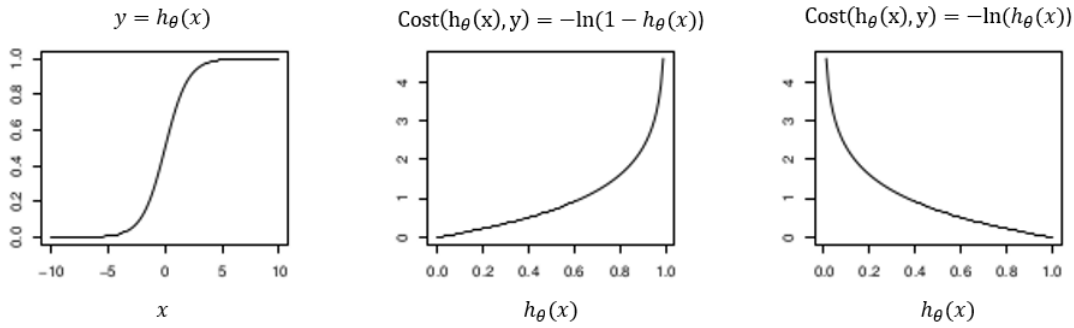
Double Class Classification Cost Function

The cost function used in Classification problems cannot be the same as with regression algorithms, because the sigmoid function get close to 1 and 0 really fast, therefore the cost function must requilibrate this effect.

Formula 3.9, together with figure 3.7, shows how to obtain the equilibrium necessary: the closer the hypothesis $h_\theta(x)$ gets to the correct prediction, the smaller will be the cost function ($Cost(h_\theta(x), y)$), following a logaritmic proportionality, while if the prediction is wrong, the cost function will grow exponentially.

$$
\begin{aligned}
Cost(h_\theta(x), y) &= -\ln(h_\theta(x)) && if y = 1 \\
Cost(h_\theta(x), y) &= -\ln(1 - h_\theta(x)) && if y = 0
\end{aligned}
\tag{3.9}
$$

To avoid the condition on y, the two cost functions of equation 3.9 can be compressed



(a) *Sigmoid function.* **(b)** *Cost for* $y = 0$. **(c)** *Cost for* $y = 1$.

Figure 3.7: Cost Function for Logistic Regression

into 3.10, and the total cost function over all the training examples will be the mean value, expressed by equation 3.11.

$$Cost(h_\theta(x), y) = -y \ln h_\theta(x) - (1 - y) \ln 1 - h_\theta(x) \tag{3.10}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \ln h_\theta(x_i) + (1 - y_i) \ln 1 - h_\theta(x) \tag{3.11}$$

Consequently, applying the general rule for updating the parameters (θ) in gradient descent (equation 3.3), it is possible to obtain equation 3.12, which remains the same as in linear regression (equation 3.5

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_{ij}) \tag{3.12}$$

3.2.2 Multiclass Classification

If the *classes* (labels) of the input data are more than two ($y = 0, 1, ...n$), it is still possible to apply the same algorithms used for binary classification by the **One-vs-All** approach, shown in figure 3.8.

One binary classificator will be trained for each class, predicting whether each element belongs or not (binary choice) to the class considered.

When all binary classificators have been trained in such way, the algorithm will predict an unseen data belongs to the class connected to the classificator that scores higher on that input data.

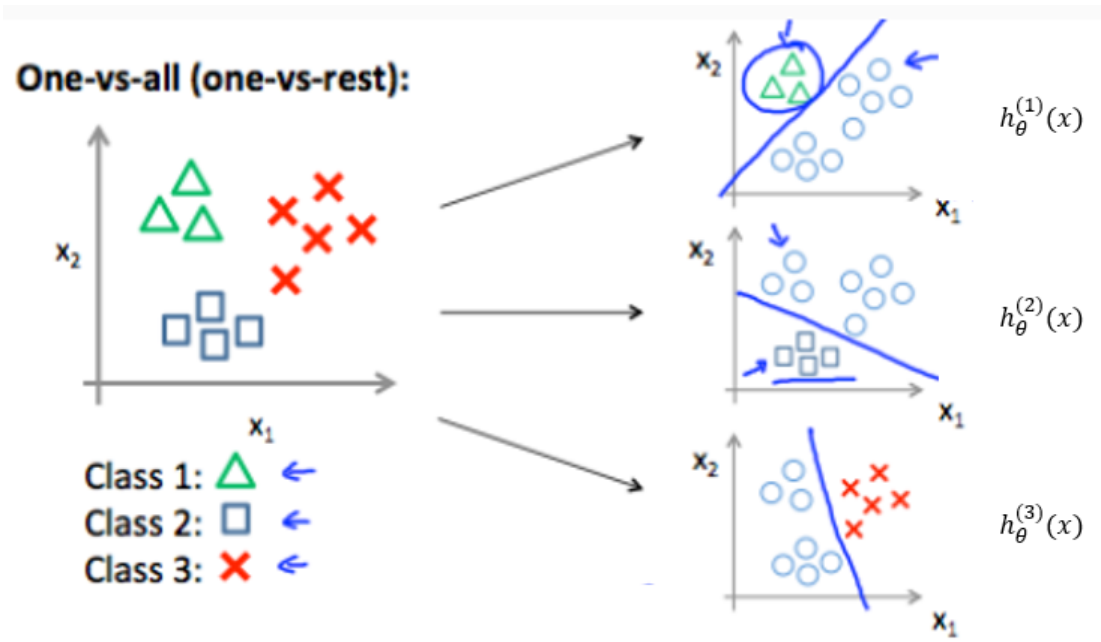


Figure 3.8: Multiclass Classification with One-vs-All approach.

3.3 Stochastic and Mini-Batch gradient descent

The algorithm of gradient descent presented before is not computationally efficient, because at every step of the iteration, it needs to sum up the contribution to the gradient of all training examples. To make the computation more efficient, Stochastic and mini-batch gradient descent were invented, and their differencies can be visualized in figure 3.9

Stochastic gradient descent, manages to speed up the computation by considering only one training sample for every iteration, and updating the parameters (θ) accordingly.

However, Stochastic gradient descent has extremely low accuracy, thus **Mini-Batch gradient descent** is often used instead, because it creates a good tradeoff between accuracy and speed. In mini-batch gradient descent, for every step of the optimization a group (*batch*) of training examples is considered and the gradient is computed accordingly to update the values of the parameters (θ).

In this case the optimal dimension of the batch depends on the size of the training set, therefore it must be chosen by experience, or by trial and error, increasing gradually the size of the batches until a satisfying accuracy is reached.

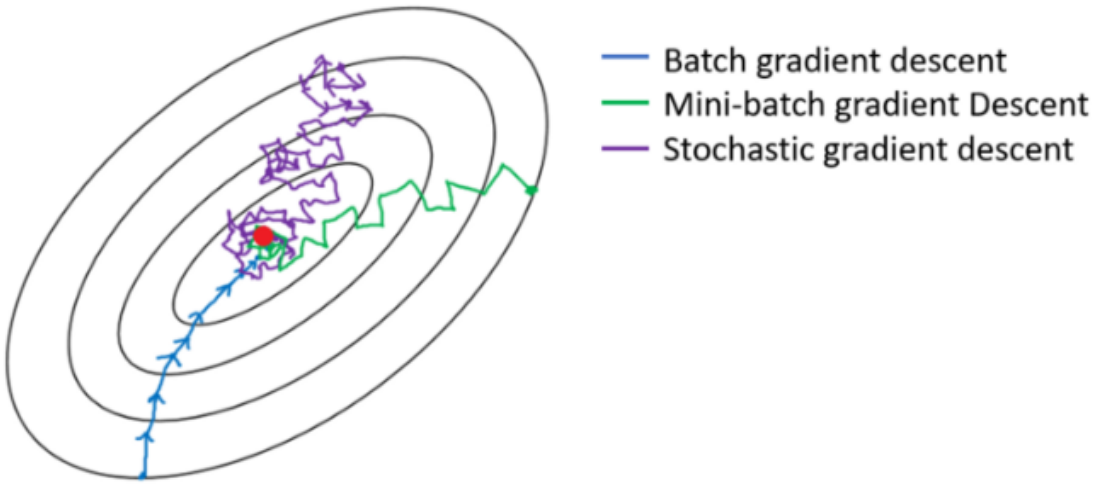


Figure 3.9: Batch, Stochastic and Mini-Batch Gradient Descent.

3.4 Underfitting and Overfitting problem

The model for a particular training set must hold the right level of complexity to perform correctly over unseen data (figure 3.10 can provide intuition of this). Therefore it doesn't exist a model that is good for all training data, but each case should be evaluated before applying a model.

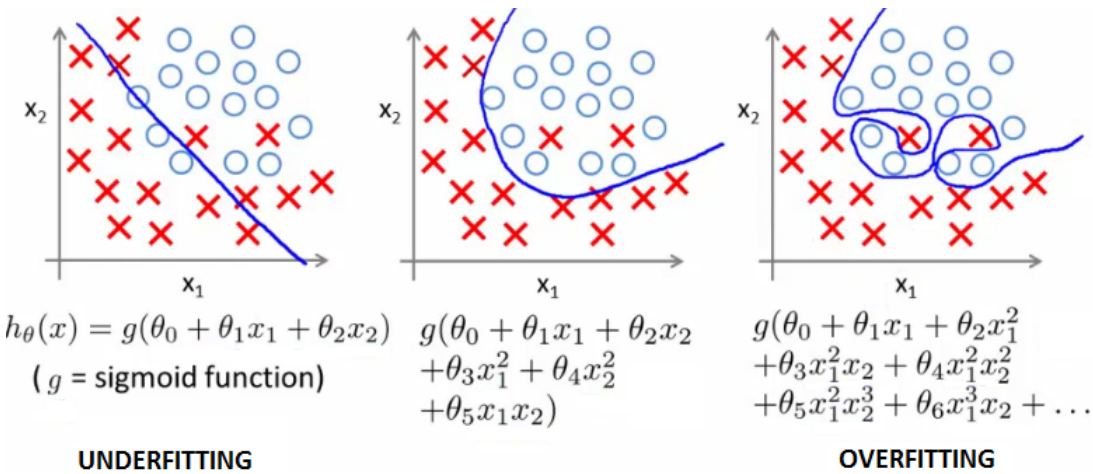


Figure 3.10: Underfitting, Optimal and Overfitting models.

If the model is too simple (low polynomial grade) for the distribution of data given, most of the training data will be far from the prediction made. In this case and it is said that the model **underfits** the data and that the **Bias** is high.
On the contrary, if the model is too complex (high polynomial grade) for the distribution of data given, the model will be extremely accurate on training data, but it will make mistakes with unseen data. In this case and it is said that the model **overfits** the data and that the **Variance** is low

Bias is the tendency of the model to not take into account all the variables, making a simplistic prediction, that makes mistakes on both training set and unseen data.
Variance is the tendency of the model to obtain different prediction performance depending on the input data (typically really low error on training set and mistakes on unseen data).
The concepts of Bias and Variance are useful to synthesize the concept of Underfitting and Overfitting, because they tend to be inversely proportional, and a trade-off between them must be found in each specific case, in order to use the optimal model that minimizes both variables as intuitively shown in figure 3.11.

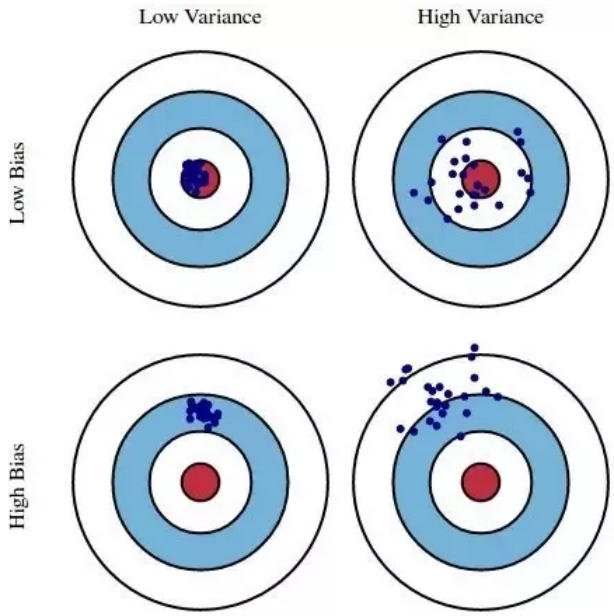


Figure 3.11: Graphical representation of Bias and Variance relationship.

3.5 Regularization

The problem of underfitting is easily avoided by changing the model to a more complex function, while the problem of overfitting can be solved simply modifying the cost function (easier optimization process).
To introduce this concept it is useful to imagine (as an example) that the model function has equation:

$$h_\theta(x) = \theta_0 x + \theta_1 x^2 + \theta_2 x^3 + \theta_3 x^4 + \theta_4 x^5 \tag{3.13}$$

Regularization is a technique that adds a term to the cost function and when the parameters (θ_j) are updated, in order to make them smaller, as shown in equations

3.14 (cost function) and 3.15 (gradient descent).

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2 \tag{3.14}$$

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i) - y_i)x_{i0})\\ \theta_j &:= \theta_j - \alpha[(\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i) - y_i)x_{ij})) + \frac{\lambda}{m}\theta_j\end{aligned} \tag{3.15}$$

Being the parameters of the model smaller, the coefficients of the model equation will be smaller, and the function will be smoother (simpler, and not overfitting). The parameter θ_0 is not regularized, because being the *bias* term, is doesn't contribute to the complexity of the function, but simply to its position in the n-dimensional space.
Therefore by adjusting the value of the **Regularization Parameter** (λ), by trying some possible values and checking the performance of the model obtained, it is possible to make the function perfectly fitting, finding the right trade-off between bias and variance.

3.5.1 Regularized Logistic Regression

In the case of logistic regression the expression of the cost function is different, so the regularized cost function becomes that of equation 3.16, while the gradient descent stays the same as equation 3.15.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \ln h_\theta(x_i) + (1 - y_i)\ln 1 - h_\theta(x_i) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{3.16}$$

However it needs to be considered that, for Logistic Regression, $h_\theta(x) = \frac{1}{1+\exp -\theta^T x}$ (sigmoid function).

3.6 Train, Validation and Test Sets

Usually machine learning algorithms have access to some training data already labeled, and this information should be used for training, optimizing and testing the algorithm, before it can be applied to predict unlabeled data.
Therefore, the training set is usually randomly split into 3 parts that have different purpose:

- **Training Set (60%)** → data used to train the algorithm
- **Cross-Validation Set (20%)** → data unseen during training, used to optimize the hyperparameters $\lambda, \alpha, n, ...$)
- **Test Set (20%)** → data unseen during training, and optimization, used to test the final model obtained

It should be noted that in order to ensure that the algorithm is not optimized over the test set (which would ensure better performance over the test set and worse on new data), the data should be split randomly (in a different way) at each initialization.

3.7 Diagnosing and solving Bias and Variance errors

By splitting the available data into training, cross-validation and test sets,it becomes possible to understand whether the cause of low performance (high error) is underfitting (high bias) or overfitting (high variance), as shown in figure 3.12.

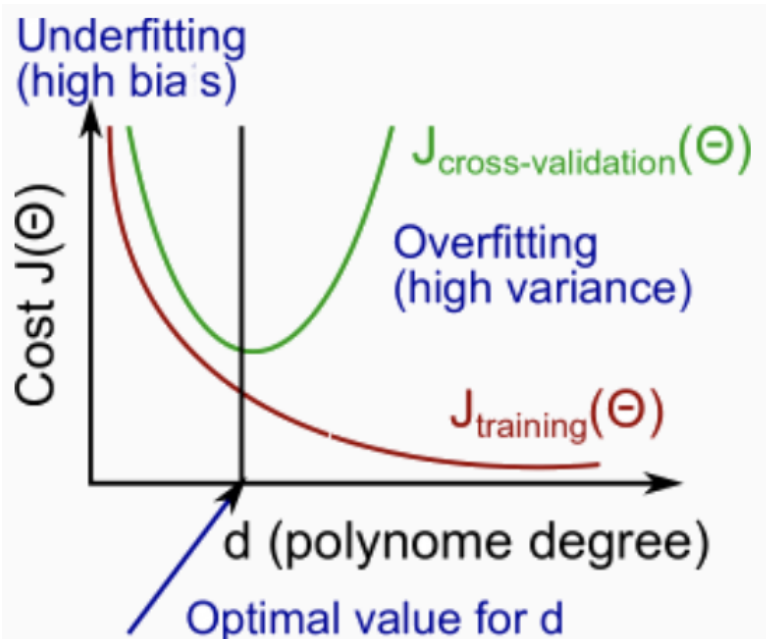


Figure 3.12: Cost Function as function of the complexity of the model.

Understanding whether the error is caused by bias or variance, is particularly useful to understand which actions to take in order to improve the algorithm:

- *Getting more training examples* can fix an error of *high variance*, but it will not lower significantly the bias, as shown in figure 3.13.
- *Reduce the number of features* is equivalent to lower the complexity of the model, so it particularly useful in case of *high variance*. However with this kind of solution, only the redundant features should be neglected in order to avoid the loss of important information.
- *Adding new features (polynomial or with different nature)* can fix a problem of *high bias*
- *Increasing the regularization parameter* λ) fixes an error of *high variance* (as already shown), while decreasing it reduces the bias.

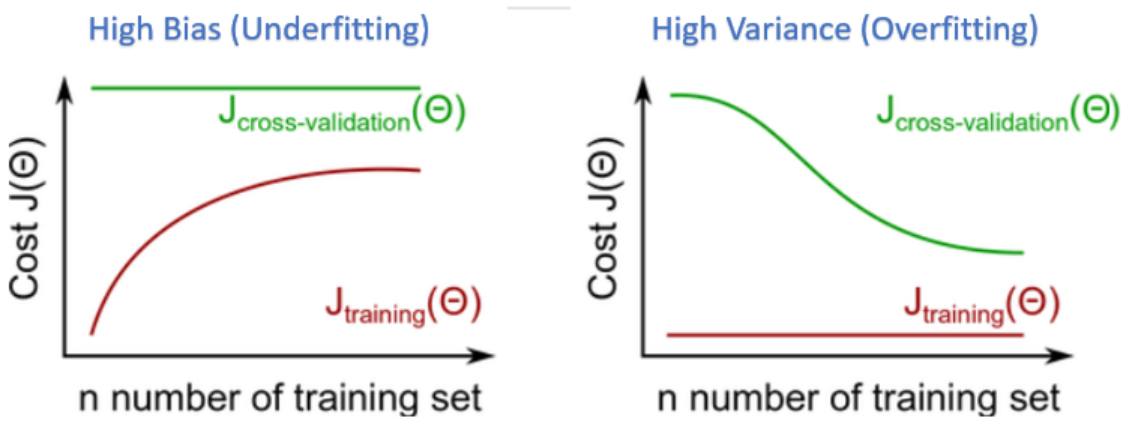


Figure 3.13: Cost Function as function of the number of training examples.

3.8 Skewed Classes: Precision, Recall and F_1 Score

In the case of a classification problem (Logistic Regression algorithm), if one class is highly more frequent than the other one (e.g. 99% vs. 1%), it is said that the classes are **skewed**.
In this case an extremely simple model predicting that every example belongs to the most frequent class would have an extremely high performance without actually giving any useful information.
An example of this situation is an algorithm trying to detect terrorists (who are an extremely small fraction of the population): a model predicting that no one is a terrorist would have a great accuracy, but it would fail to detect any terrorist at all.
To avoid such kind of pointless predicting models, the evaluation metric should be modified in the case of skewed class problems, and estimated looking at the values

of **Precision**, **Recall** and F_1 **Score** (which is the combination of the previous variables), expressed as shown in figure 3.14 and equation 3.17:

- **Accuracy** is the standard way of measuring the performance of a classification algorithm, in the case of non-skewed classes, defined as the amount of correct predictions over the total number of predictions made.
- **Precision** is the ability of a classification model to identify *only* relevant cases, and it is mathematically defined as the amount of true positives predicted, over the total amount of positives existing.
- **Recall** is the ability of a classification model to return *all* relevant cases, and it is mathematically defined as the amount of true positives predicted, over the total amount of positives predicted.
- **$\mathbf{F_1}$Score** combines the information contained into precision and recall (by means of the *harmonic mean*), and it must be close to 1 for a good model (low amount of both false positives and false negatives).

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 3.14: Possible prediction outcomes (TP, TN, FP, FN).

$$\textbf{Accuracy} = \frac{TruePositives + TrueNegatives}{m} \qquad \textbf{Precision} = \frac{TP}{TP + FP}$$
$$\textbf{F}_1\textbf{Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \qquad \textbf{Recall} = \frac{TP}{TP + FN} \tag{3.17}$$

Both Precision and Recall must be close to 1 in a good predicting model, therefore F_1Score is a necessary and sufficient information to assess the performance of the model also in the case of skewed classes.
Reconsidering the example presented before of a terrorists detector that predicts that no one is a terrorist, the performance evaluation parameters would be the following: $Accuracy \sim 1$; $Precision \sim 0$; $Recall \sim 0$; $F_1Score \sim 0$.
An adjustment that is fundamental for the calculation of Precision, Recall and F_1Score, is to always consider the rare class as positive.

3.9 Decision Tree

Decision Tree is a supervised learning algorithm used mainly for classification problems, which is based on splitting the data into homogeneous *sets*, as shown in figure 3.15.

The **Root Node** represents the entire training set (the *population*) and it is

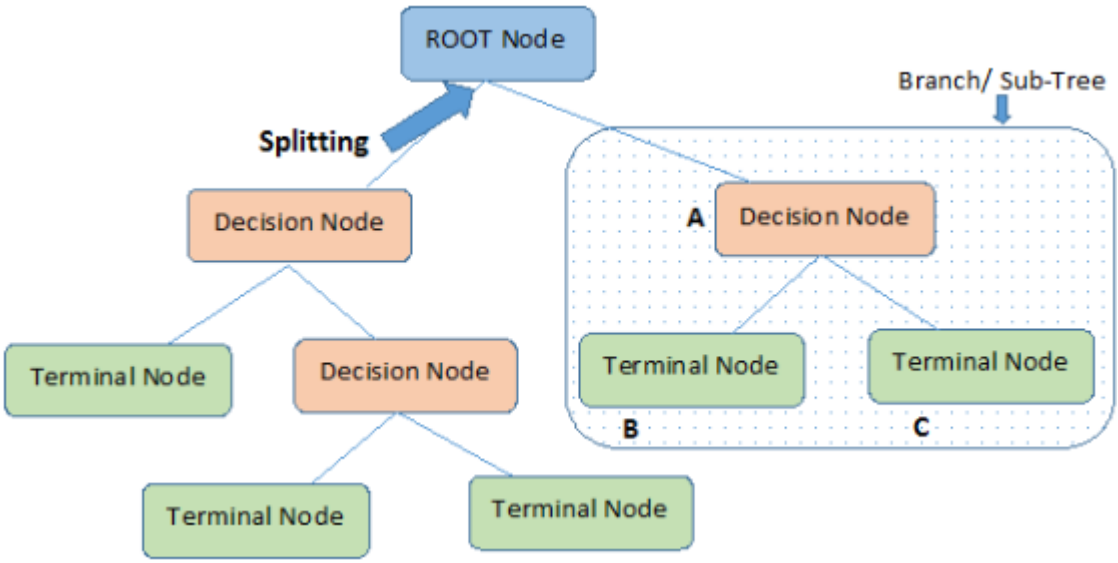


Figure 3.15: Decision Tree logic scheme[2].

split into **Decision Nodes**, which can be split into other decision nodes or **Leaves (Terminal Nodes)**. Inside a branch it is said that an upstream node is the **Parent Node** of some downstream **Child Nodes** (e.g. A is parent node of children nodes B and C in figure 3.15).

What characterize a Decision Tree algorithm is the logic chosen for the splitting, because there are many, with different characteristics.

In order to introduce some of the possible different splitting logics the simple problem of figure 3.16 is analyzed. This example tries splitting a group of 30 students according to three different classes (gender, height and class), in order to try to classify those who play cricket in an homogeneous group.

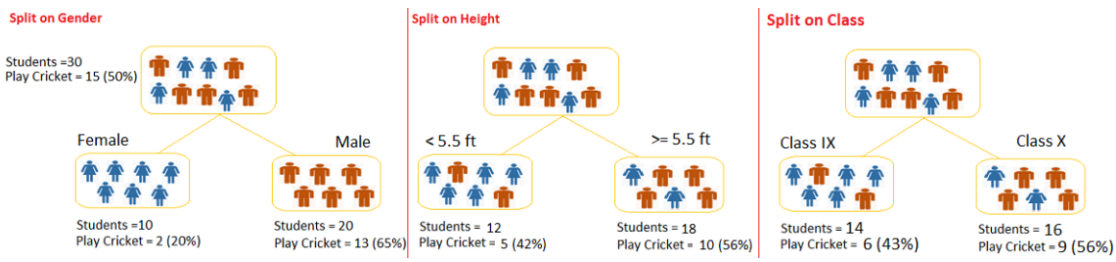


Figure 3.16: Decision Tree problem example.

3.9.1 Gini Index

Labeling as p the percentage of students that play cricket (success) and q the percentage of students who don't (failure), the **Gini Index** can be computed for each hypothetical sub-node as the sum of their squared value for each specific class ($GI = p^2 + q^2$), and then the total score can be computed as the weighted mean of the previous values. The Gini index computed for the example of cricket players presented before is shown in figure 3.17, and the best split proves to be according by gender.
The gini index method is used to perform only *Binary Splits*, because it is the simplest for such operation.

Splitting Logic	First Sub-Node	Second Sub-Node	total
Gender	0.68	0.55	0.59
Height	0.51	0.51	0.51
Class	0.51	0.51	0.51

Figure 3.17: Gini Index calculation example.

3.9.2 Chi-Square

Another possible method is **Chi-Square**, which compares parent node (*Expected Frequency*) with child nodes (*Actual Frequency*), and it can perform also more than two spits at the same time (unlike gini index).
The expression of the chi-square score is: $CS = \frac{(A-E)^2}{E}^{\frac{1}{2}}$, where A is the Actual frequency (child node) and E is the Expected frequency (parent node).
The Chi-Square scores computed for the example of cricket players presented before is shown in figure 3.18, and gender is confirmed to be the best splitting logic among those tested.

Splitting Logic	First Sub-Node	Second Sub-Node	total
Parent Node	/	/	1
Gender	0.72	0.93	0.86
Height	0.98	0.99	0.98
Class	0.98	0.99	0.98

Figure 3.18: Chi-Square calculation example.

3.9.3 Information Gain

Another way to evaluate the quality of the sub-nodes is calculating their **Entropy** as shown in equation 3.18, where p is the percentage of positive cases (students playing cricket) and q is the percentage of negative cases (students not playing cricket).

$$E = -p \log_2 p - q \log_2 q \tag{3.18}$$

Entropy is defined as the measure of the lack of order or predictability, while the aim of Decision Tree is to separate homogeneous groups of elements, therefore a low entropy should be sought by this algorithm.
The total entropy of each branch will be computed as the weighted average of the entropy of all the sub-nodes, and as long as the entropy keep decreasing, the process can go on.
Computing the entropy of the example of cricket players used before according to equation 3.18, yields the results of figure 3.19, and once again the best split would be according to gender.

Splitting Logic	First Sub-Node (positive)	First Sub-Node (negative)	Second Sub-Node (positive)	Second Sub-Node (negative)	total
Gender	1.34	1.34	0.95	0.95	4.58
Height	0.41	0.41	0.33	0.44	1.48
Class	0.38	0.38	0.35	0.35	1.46

Figure 3.19: Information Gain calculation example.

3.9.4 Prevent Overfitting of Decision Tree

If an algorithm of decision tree is not given any constraint, it will ultimately achieve 100% accuracy, returning a model with one training example in each leaf or terminal node.
Therefore it is fundamental to give some constraints to the complexity of the final model, which are usually:

- Setting a *minimum size of split nodes*
- Setting a *minimum size of terminal nodes*
- Limiting the *maximum depth of the tree*, which constrains its vertical length
- Setting a *maximum number of terminal nodes*, which constrains the horizontal length of the tree
- Setting a *maximum number of features* considered when splitting, usually (as a rule of thumb), the square root of the total number of features, randomly chosen between all features available

Setting some constraints to the decision tree is not the most efficient way of limiting the complexity of the model, because it uses a coarse method of constraining the growth of the tree.

Another method, which is also adopted in nature consists in letting the tree grow freely and then cut the leaves that are not necessary, which is a process named **Pruning**.
Therefore in a decision tree algorithm, an extremely complex and overfitting model will be created first, and then it will be simplified by removing the terminal nodes (leaves) that have a low gain.

3.9.5 Advantages and Disadvantages of Decision tree

overall, decision tree is usually better than logistic regression (for classification problems) if the relationship between variables is complex, and to interpret and explain the model found to other people, because it is more intuitive.

3.10 Support Vector Machine

Support Vector Machine (**SVM**) is a supervised machine learning algorithm which is particularly efficient for classification problems.[17]
The working principle is to plot each training sample in a n-dimensional plane (where n is the number of features of each training sample), and then separate the different class with an hyper-plane (or line if $n = 2$), as shown in figure 3.20.

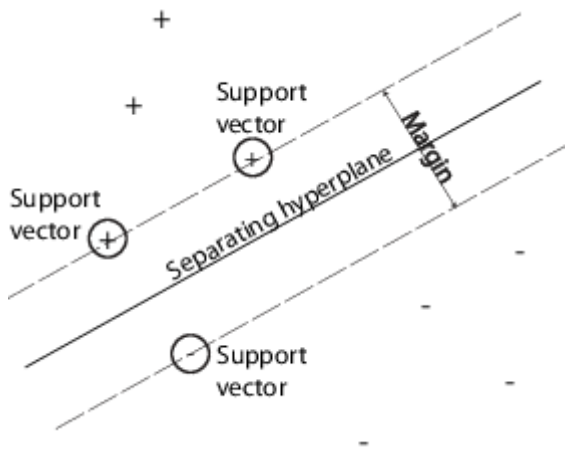


Figure 3.20: Support Vector Machine working principle.

The points (training data) closer to the **Separating Hyperplane** belonging to each possible class, are named **Support Vectors**, and they are used as reference for the choice of the best separating hyper-plane (the one with highest margin).

Anyway if the SVM has the possibility to classify correctly all training samples, it will prefer this solution, penalizing having a high margin, if it proves to be necessary. As an example, looking at figure 3.21, the algorithm will prefer the hyper-plane A more than B, because it has lower margin, but it correctly classifies all training samples.

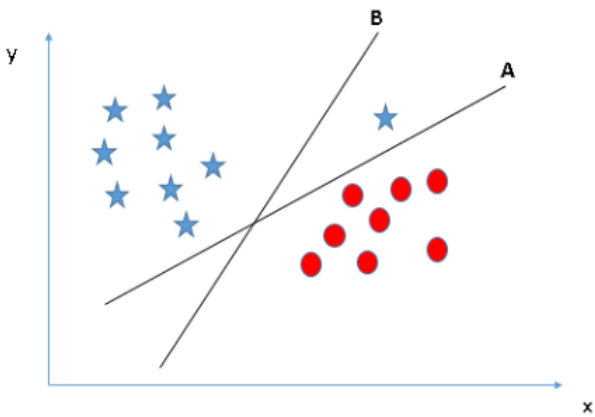


Figure 3.21: SVM: higher margin or better performance?.

In other situations, there are some points that cannot be correctly classified along with all the others, such as the star on the right in figure 3.22. These training samples are **Outliers**, and the algorithm must ignore them.

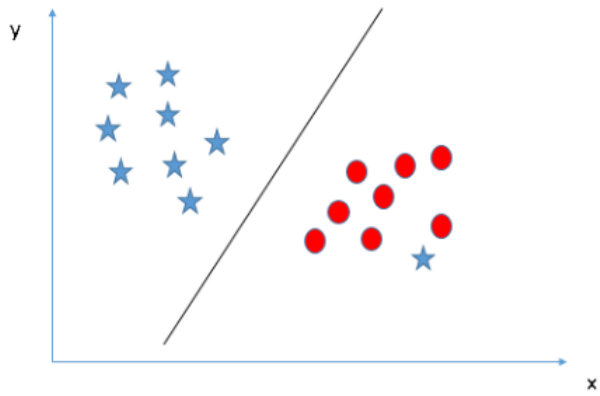


Figure 3.22: SVM: managing outsiders.

Another case may present a non-linear distribution of the data, therefore an hyper-plane would not be useful to separate different classes from each other, unless a particular adjustment of the available data is performed, as shown in figure 3.23.
A **Kernel Function** is chosen to adapt the data, then a standard hyper-plane will be able to separate the data effectively.

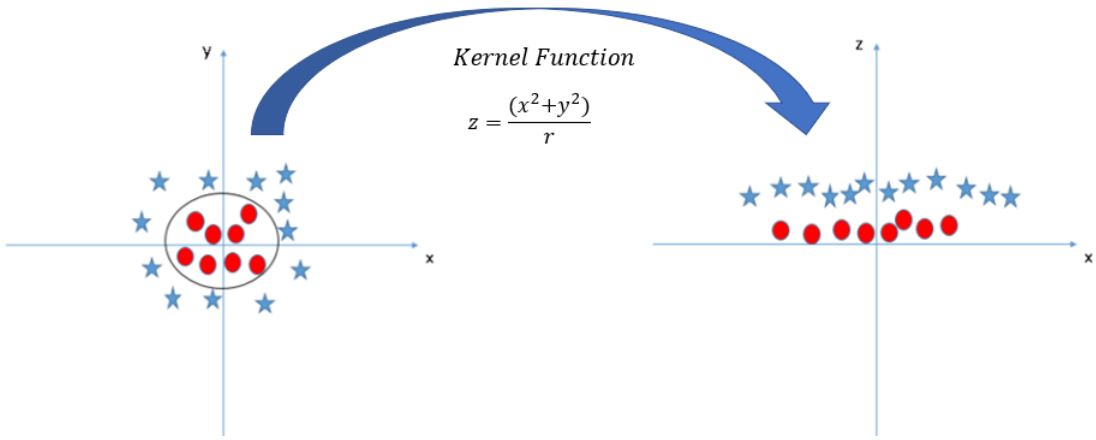
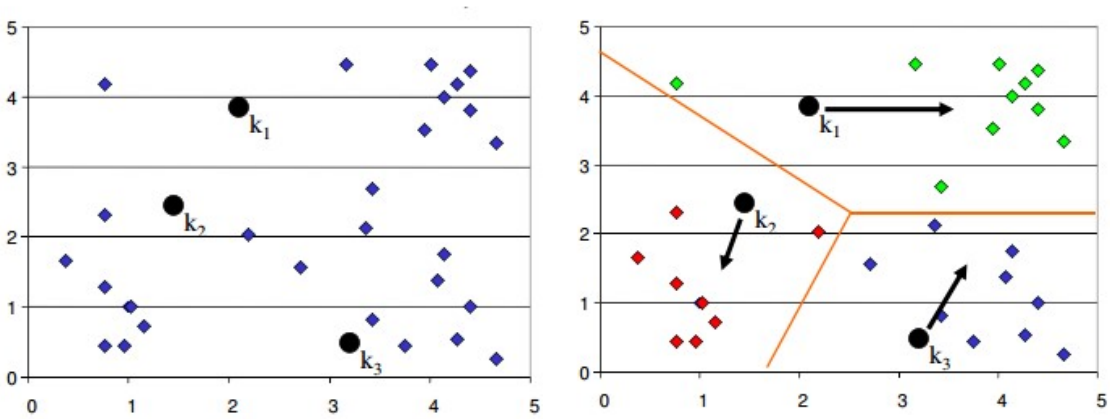


Figure 3.23: SVM: Kernel Function.

3.11 K-Means Algorithm

Changing the topic to Unsupervised Learning problems, **K-Means** is one of the simplest clustering algorithm, which is given as input unlabeled data, such as shown in figure 3.24a, and must classify it into different clusters, as shown in figure 3.24b.[8] In order to obtain this result, some centroids (k_1, k_2, k_3) are randomly initialized, then each input data will be assigned to the closest centroid, creating clusters. To optimize the clusters created in this way, the centroid is then moved into the mean values of the data assigned to it, as shown by the arrows in figure 3.24b, and the input data will be reassigned accordingly to the new position of the centroids.
This process is repeated until the position of the centroids becomes stable (until *convergence*), and the data points will be separated into clusters that can be analyzed in order to understand how the data is organized.



(a) Unsupervised Learning Problem.

(b) Clustering process.

Figure 3.24: Clustering Algorithm working principle.

3.11.1 Choice of the number of Clusters

In clustering problems it is fundamental to *normalize* input data (re-scale all features into the same fixed range), otherwise the clusters will not be meaningful. The only parameter that needs to be chosen in K-Means algorithm is the optimal number of clusters which can separate data in the best way.
In order to do this many values should be tested, and the **Within Sum of Squares** (**WSS**), which represents the mean value of the squared distances between the centroid and each data point which belongs to the cluster corresponding to it for each possible number of clusters tested. The result will look approximately as figure 3.25, and the optimal value is found at the elbow point (from which comes the name

Elbow method to this technique), which represents the number of clusters after which the model doesn't improve significantly.

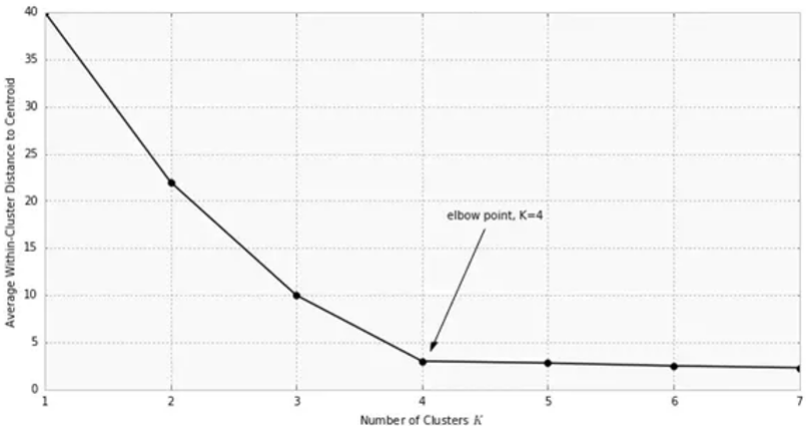
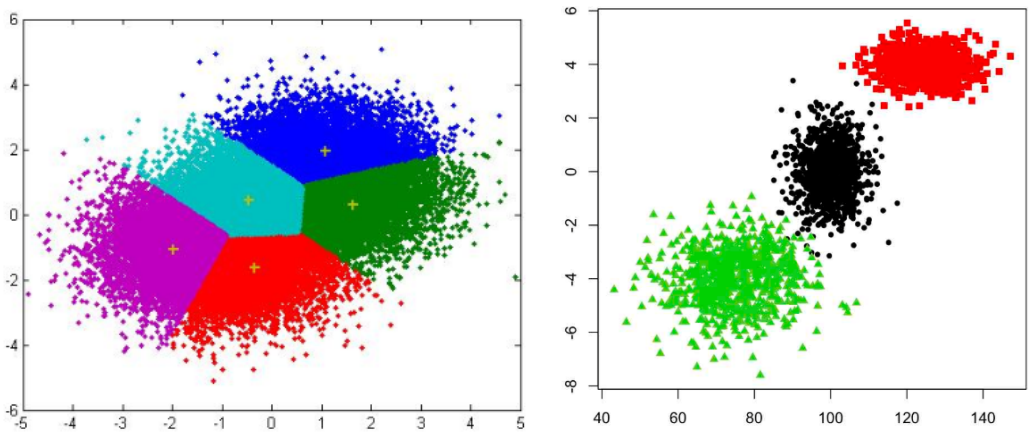


Figure 3.25: Elbow Method example.

Sometimes the Elbow Method will not give an unique and clear number of clusters, and the absence of a noticeable elbow is a symptom of unsuitability of this kind of algorithm for the problem, because it means input data is probably organized in a similar way to figure 3.26a, and it can be graphically noticed that the clusters created are not remarkably meaningful.
On the contrary, analyzing the case shown in figure 3.26b, the elbow will be more evident, and the clusters obtained are well separated.



(a) Ineffective Clustering.

(b) Effective Clustering.

Figure 3.26: Evaluation of the effectiveness of Clustering process.

3.12 Principal Component Analysis

When the problem analyzed has an huge amount of features, some of them may be more significant than others, but generally it is not advisable to simply discard some features, because it would waste some available data which could improve the final model.[15]

Principal Component Analysis (**PCA**) is an algorithm that creates a specified number of new features combining the input ones, making sure that they are oriented such that the loss of information is minimized.

Figure 3.27 graphically shows the transformation process undergone by the original data set: $pc1$ and $pc2$ are named **principal components**, representing the new features generated, and it is visible that in the new coordinates system it is possible to approximate the information by eliminating $pc2$ (done by projecting the points on $pc1$ axis). This process is defined as choosing one principal component ($pc1$).

In such way it becomes possible to carry out a *Dimensionality Reduction* of the

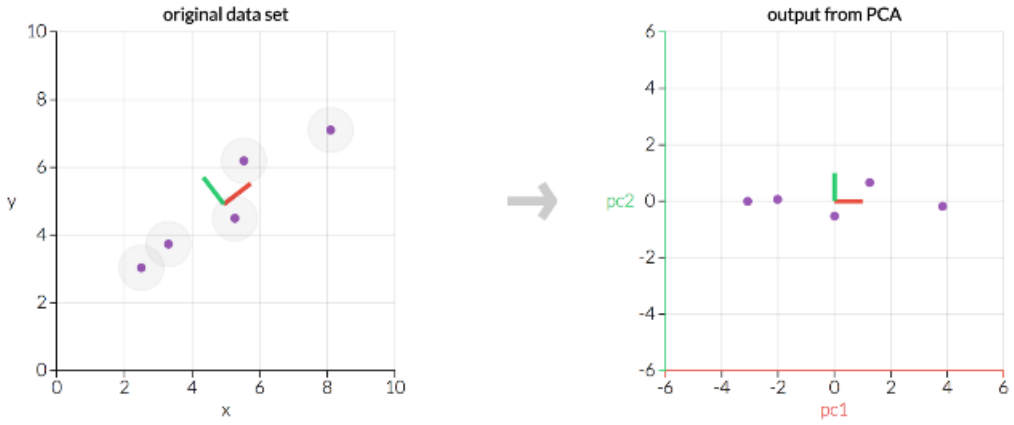


Figure 3.27: Principal Component Analysis 2D intuition.

input data, while keeping information from both input variables x and y.

Operatingly speaking, it is important to note that input data should be normalized before applying PCA, otherwise the principal components found would be heavily influenced by the different ranges of features.

After input data has been normalized, in order to generate the new features, some mathematical steps should be followed:

- Generate the *Covariance Matrix* of the input data.
- Extract the eigenvalues from the covariance matrix and sort them in descending order, choosing an arbitrarily chosen number (k) of eigenvectors corresponding to the highest eigenvalues.

- Build the **Projection Matrix** (W) from the selected eigenvectors, and then transform the original data set using the matrix obtained (multiply the matrix containing original input data with W) in order to obtain a k-dimensional subspace of new features.

In order to choose how many principal components to keep and quantify the amount of information that is being lost in the process, it is useful to compute (and plot, as shown in figure 3.28) the *proportion of variance* relative to each component, which gives the measure of the fraction of information contained inside each principal component.
The possible criteria for choosing the number of principal components to keep can be:

- *Arbitrary choice* $\rightarrow$ done if the objective is to obtain a specific number of features (possibly to be able to plot it).

- *Percentage Threshold* $\rightarrow$ choose a percentage of information that it needs to be explained by the model, and then select enough principal components to cumulatively get to that percentage of variance.

- *Elbow Method* $\rightarrow$ plotting the proportion of variance for each component as shown in figure 3.28 (**scree plot**), it is possible to see which elements are most significant than others, and qualitatively stop where the function presents an *elbow point*. In this example the elbow is at $pc2$, and the 2 out of 13 principal components describe approximately 42% of the total variance.

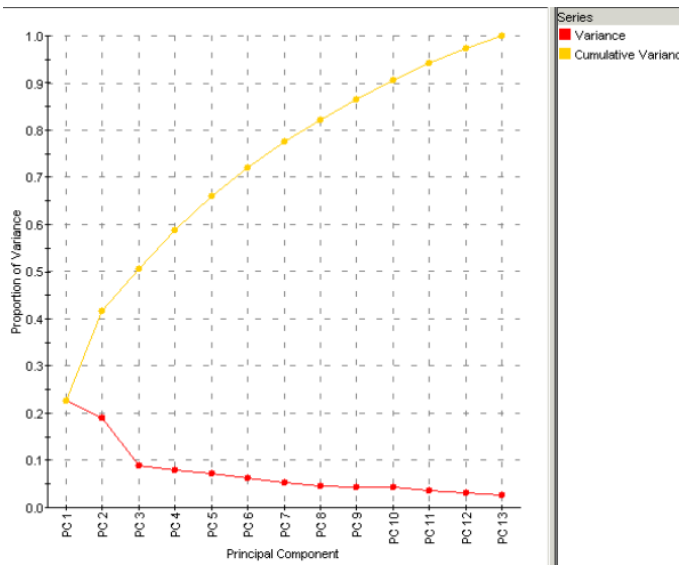


Figure 3.28: Principal Component Analysis example of Variance plot.

Chapter 4

Artificial Neural Networks

4.1 Definition of Artificial Neural Networks

Machine learning is consistently trying to improve artificial intelligence to make it possible for automated systems to substitute human efforts for the simplest tasks of everyday life and work, leaving to people the possibility to concentrate on more complex problems and activities.
Leaving out the ethical and sociological problems of this inevitable process that has been going on since the beginning of industrial revolutions, it has been natural that the process of imitating human work brought researchers to try to artificially emulate the structure of human brain in order to achieve better performance than already existing machine learning algorithms, giving birth to **Artificial Neural Networks** (**ANN**).

This kind of approach is a branch of Machine Learning, named **Deep Learning**, because its working process is characterized by some intermediate steps in the learning process, which process the information to generate an output.
The most important difference between Deep Learning and other machine learning techniques building analysis in a linear way, is that Neural Networks are organized hyerarchically, processing data in a non-linear way.

4.2 Structure of Artificial Neural Networks

Neural networks are made of different **layers** of **Neurons**, as shown in figure 4.1.

This example shows a **fully connected** neural network, because all neurons are connected to both those in the next layer and those in the previous one.
The *size* of each hidden layer (number of neurons per layer) and the *depth* of the network depend on the complexity of the problem and on the design choice.

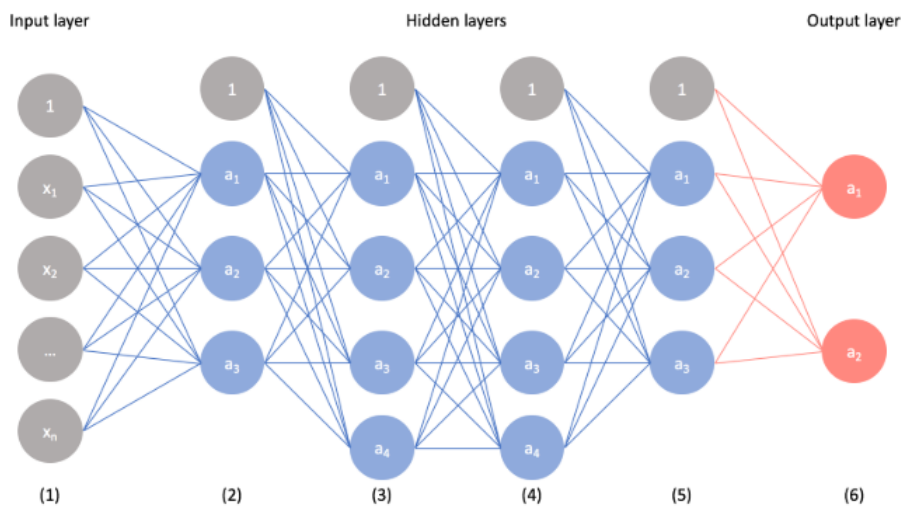


Figure 4.1: Artificial Neural Networks base structure

The size of the input layer depends on the number of the input features (n), while the size of the output layer depends on the number of possible classes for the prediction (usually neural networks are used for classification problems), and the class predicted will be the one corresponding to the output neuron with the highest score.
Each layer has a bias input that has value b, while for each *neural connection* there is a weight ($W_i^{(j)}$) that is multiplied to the previous neuron to obtain the value of the following one.
Another element that must be chosen is the **Activation Function** to *propagate* the values of the neurons up to the output layer, and among all the possible ones (shown before in figure 3.6), the *sigmoid* function (represented as σ) will be chosen in the following examples.

4.3 Feedforward and loss Function

Given the elements described, it is possible to describe mathematically the propagation of the input information in a simple two-layer neural network (the input layer doesn't count) as shown in equation 4.1 (where W_1 and W_2 are the matrices containing all the weights between one layer and the previous, and Y is the vector of all the output classes values).

$$Y = \sigma(W_2 \cdot \sigma(W_1 x + b_1) + b_2) \tag{4.1}$$

This process is named **Feed-Forward Propagation** and it is shown graphically in figure 4.2.

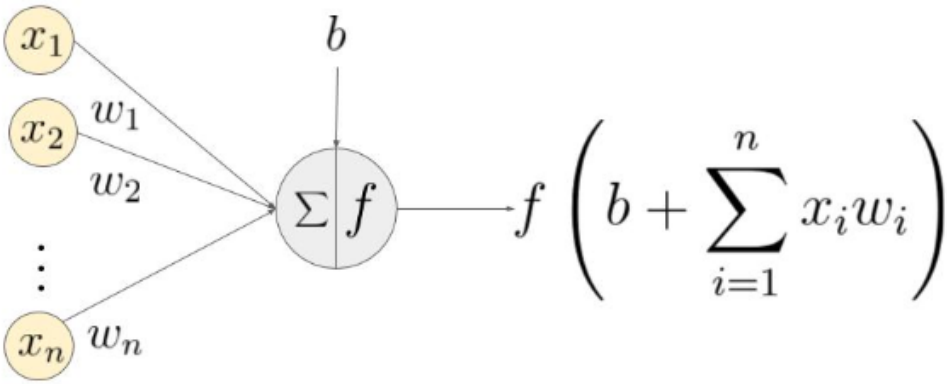


Figure 4.2: Feedforward propagation of inputs.

In order to evaluate the precision of the prediction made a **Loss Function** will be employed which can be simply the sum of the squared error for each class:

$$Loss(y, Y) = \sum_{i=1}^{n} (y_i - Y_i)^2 \tag{4.2}$$

4.4 Backpropagation

The goal of the algorithm is to find the best set of weights (W) and biases (b) that minimize the loss function.
In order to update the weights and biases in every step of the optimization, the gradient of the loss function with respect to the weights and biases must be computed. Recalling that the derivative of the sigmoid function is retrieved in equation 4.3

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)) \tag{4.3}$$

Since the loss function is not directly expressed as function of the biases and weights the **chain rule** of derivation must be employed as shown in equation 4.4 in order to obtain the gradient ($z = W_j x + b$).

$$\begin{aligned} \frac{\delta Loss(y, Y)}{\delta W_j} &= \frac{\delta Loss(y, Y)}{\delta W_j} \cdot \frac{\delta Y}{\delta z} \cdot \frac{\delta z}{\delta W_j} \\ &= 2(y - Y) \cdot \frac{\delta \sigma(z)}{\delta(z)} \cdot x \\ &= 2(y - Y) \cdot z(1 - z) \cdot x \end{aligned} \tag{4.4}$$

The process shown in figure 4.3 must be repeated backwards until the input layer, updating the weights accordingly.

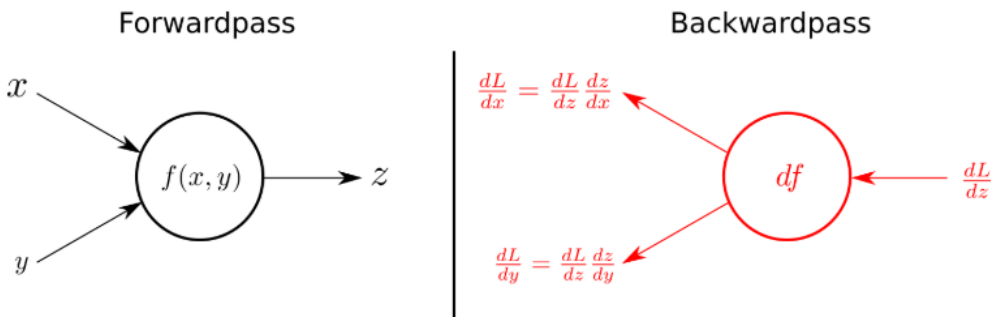


Figure 4.3: Feedforward and Backpropagation in a single Neuron.

As in logistic regression, the neural network should be adapted to each specific case by choosing the optimal *activation function* and *learning rate*, and this can be done dedicating a fraction of the training set to *cross-validation* and to *testing*.

4.5 Softmax Function

The **Softmax** Function, is a special kind of activation function employed often in neural networks, and it is represented by equation 4.5, where W are the weights given to features.

$$F(x_i) = \frac{e^{x_i W_j}}{\sum_{i=1}^{m} e^{x_i W_k}} \tag{4.5}$$

With respect to the sigmoid function, the peculiarity of Softmax, is that the sum of the values of the output layer (each one associated to a class) is 1, meaning that the value of the output neurons can be considered the predicted probabilities that the element analyzed belongs to each class, therefore this function is usually applied before the output layer of an Artifical Neural Netowrk.
Due to this difference, sigmoid is used mainly for Binary Classification tasks, while softmax proves particularly useful for Multi-Classification problems.

4.6 Dropout

In neural networks Regularization is used to prevent overfitting with a great amount of input data.
However sometimes also **Dropout** is used, which is graphically shown in figure 4.4, and during training it randomly ignores some neurons chosen by probability (p), reconsidering them during testing (scaling them according to p).

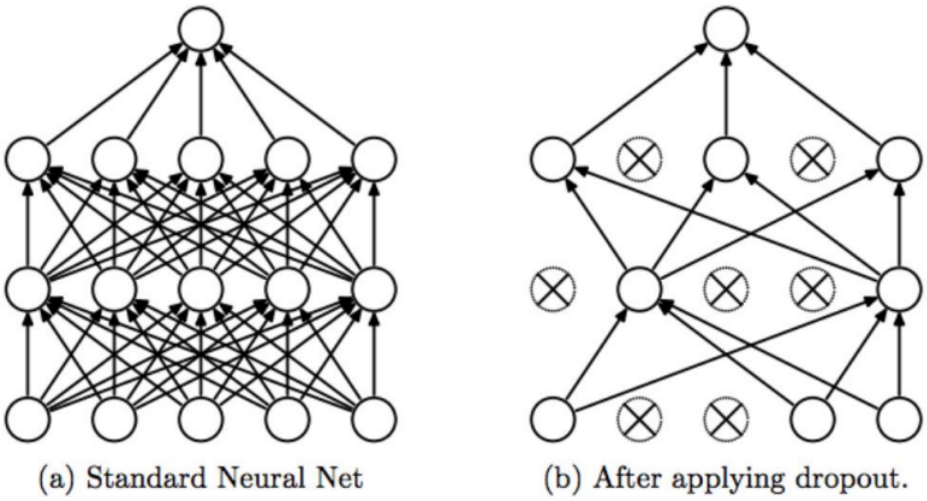


Figure 4.4: Dropout graphical representation.

4.7 Convolutional Neural Networks

4.7.1 Architecture

Images are seen by the computer as matrices of pixel values whose size depends on the resolution. As an example, a color image in JPG with resolution 480x480 pixels is seen as a matrix 480x480x3 with values of intensity of color between 0 and 250, being composed by three layers relative to Red, Green and Blue (**RGB**).

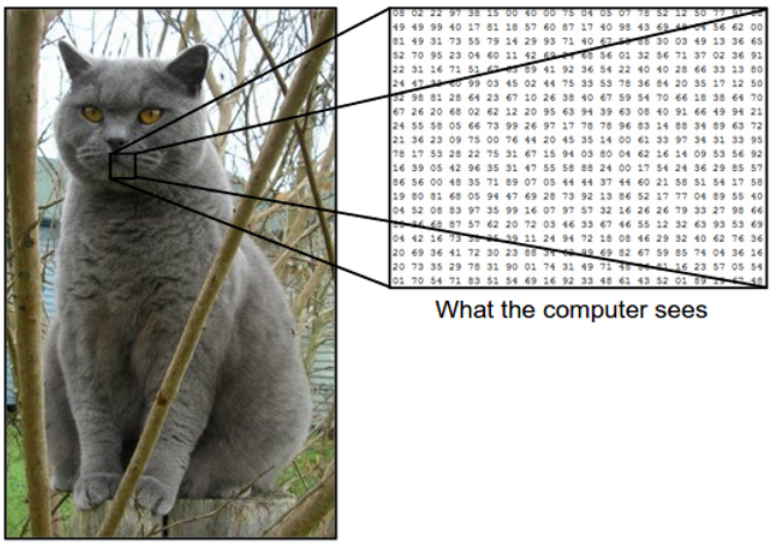


Figure 4.5: Digital representation of images.

Regular Fully-Connected Neural Networks (FC-ANN) are not suitable to be used for image analysis, because the number of features is enormous (e.g. for 480x480x3 images the features are 690K), and this generates an even higher number of parameters (one for each neural connection). The model produced by connecting every input feature to some neurons would be extremely complex, therefore it would overfit easily, and it would be slow to train.

Convolutional Neural Networks (**CNN**) make this process simpler by arranging the neurons of each layer in a 3D structure, as shown in figure 4.6.[1]

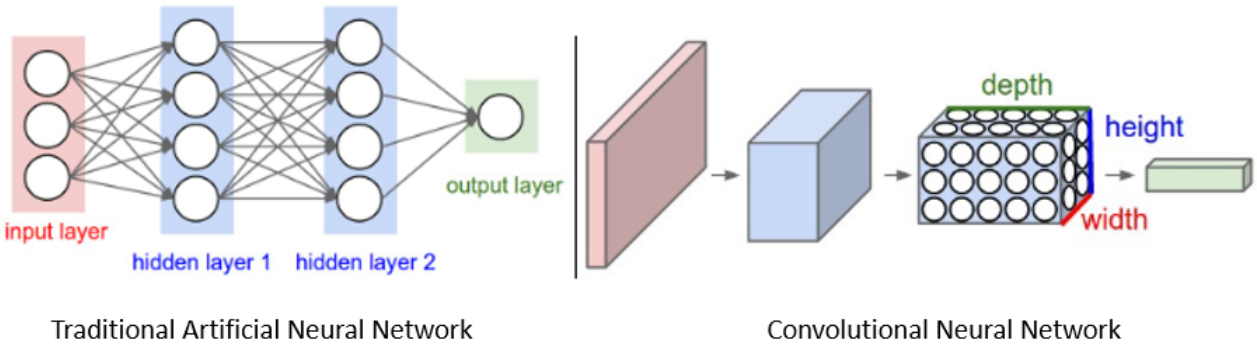


Figure 4.6: Structure comparison between FC-ANN and CNN.

4.7.2 Layer Types

Convolution Layers

The operation performed by CNN can be seen as *sliding* some filters (as many as the depth of each layer, and with dimensions equivalent to the height and width of the layer) throughout the surface of the image, multiplying the *weights* of the filter by pixel values and producing a single output value for each possible position of the filter above the image, as shown in figure 4.7.

In this way a **map of features** is produced, which can be processed by subsequent layers.

In figure 4.7, a simple example of a **Çonvolution Layer** is presented: the resolution of the image is 7x7 and the size of the filter is 3x3. A layer is composed by multiple filters (n) with the same dimension), meaning that the size of the output (map of features) will be 5x5xn. The width and height of the output is reduced by 2, because the filter should never be centered on the edges of the image, otherwise part of it would be outside of the image, producing a disturbance. Usually the size of the images is even, so the height and width of the feature maps produced is not reduced by convolutional layers.

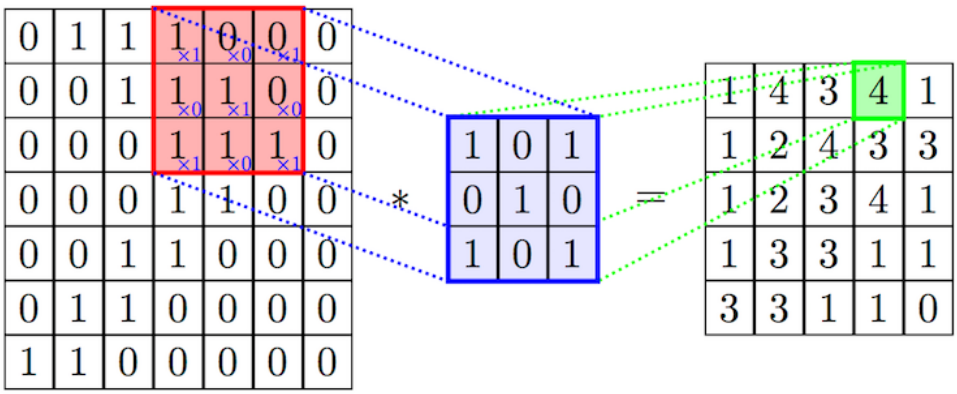


Figure 4.7: Convolution Layer working principle.

The size of the filters is generally small (as in this example), but the resolution of the images is much higher, and many different filters (generally with the same size) are used in a single layer of the network.
In this way it is possible to avoid a complex fully connected structure with an enormous number of weights, because the same set of parameters (filter values) will be used to process the whole image.

Activation Layer

The **Activation layer** is usually placed after a Convolution layer, and it simply applies an activation function to all values of the map of features produced by the previous layer, without alterating its size.
Out of all the functions presented in section 3.2.1, ReLU function is the most commonly used (ReLU Layer), due to its simplicity.

Pooling Layer

Pooling layer performs a *downsampling* operation, reducing the height and width of the map of features generated by the previous layer.
This operation happens usually by using a filter which moves on the map of features with a *stride* of 2 or more, meaning that the resultant size of the next map of features will be reduced (the depth stays the same).
A particular and widely used example of a pooling technique is **max-pooling**, which samples the highest value of the pixels covered, as shown in figure 4.8.

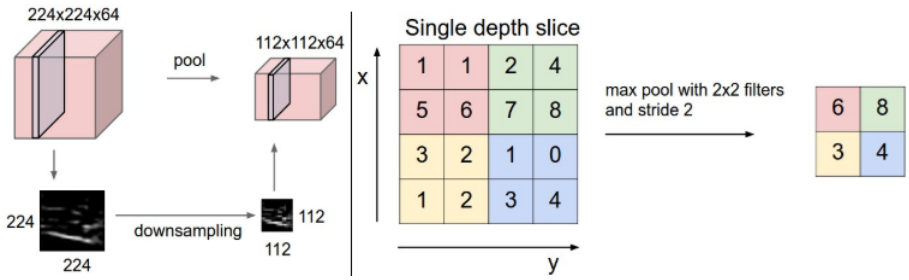


Figure 4.8: Max-Pooling technique example.

Fully Connected Layer

Another type of layer present in the last part of the first convolutional neural networks designed is a **fully connected layer**, which maps extracted features to desired outputs in the output layer, giving the values for each class available.
Usually this layer feeds output data into a **softmax Layer**, which makes sure that the sum of the output values is 1, meaning that the value in each output neuron is the probability of the item belonging to the class corresponding to that neuron.

Figure 4.9 shows an example of a complete convolutional neural network, which presents many steps of convolution and max-pooling layers, where the values are treated with ReLU activation function.

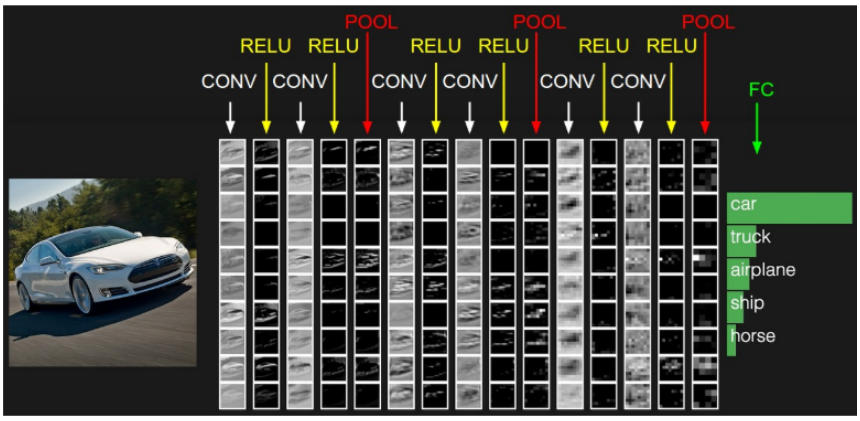


Figure 4.9: Complete Convolutional Neural Network example.

4.7.3 Backpropagation

Backpropagation in Convolutional neural networks is done exactly as in other ANN, but for what concerns the max-pooling (which is the most used type of pooling) layers, the index of the max value chosen is stored into memory, making it possible to propagate back to each pixel of the image without approximations.

Chapter 5

Object Detection

One of the most important fields of Artificial intelligence is **Computer Vision**, which is the ability of computers and software to recognize and understand visual information (images).
The analysis of visual information in machine learning focuses on two main problems: **Image Classification**, whose aim is to understand what is the main element present in images, classifying it as belonging to a specific category, and **Object Detection**, which analyzes the image, finding all significant subjects (they can be more than one) in it and giving their position, delimited by a **bounding box**, as shown in figure 5.1.
Another more precise type of object detection is **Instance Segmentation**, which assigns each pixel of the image to a class, recognizing not only the position of each object, but also its shape.

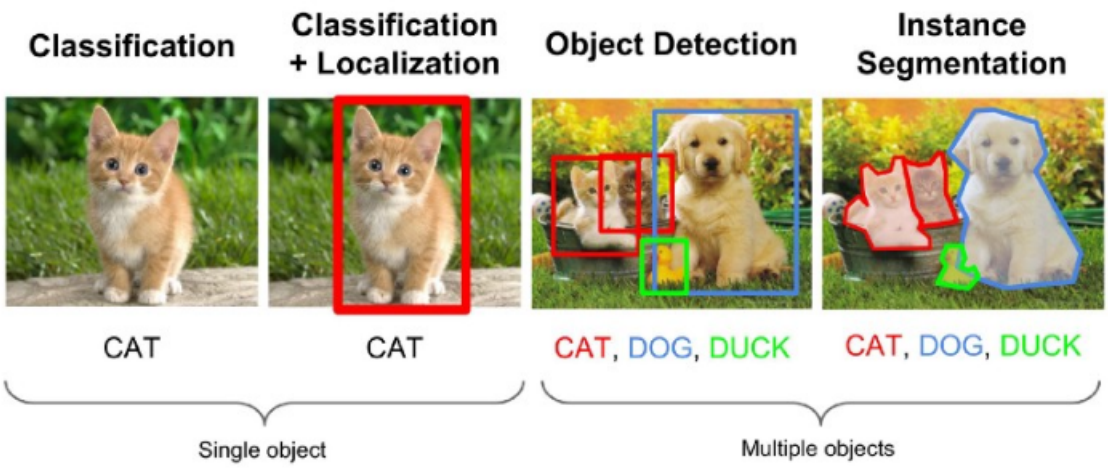


Figure 5.1: Difference between Image Classification and Object Detection[3].

Generally speaking, a model for object detection is capable of recognizing and localizing the presence of multiple objects in an image, thus it can extract from an

image more information than a model used for image classification.
As a matter of fact, image classification can sometimes be ambiguous, if in the image there are more than one subject, or if the subject that needs to be recognized occupies a small area.

5.1 Evaluation Metrics for Object Detection

In order to train and evaluate Object Detection algorithms, training images are labeled with some bounding boxes manually drawn, which constitute the **ground truth**, as shown in the example of figure 5.2. These bounding boxes are usually identified by five numbers: the coordinates of the centre of the bounding box, its width, its height, and the **confidence** of the prediction.
The value of the confidence is usually expressed in percentage, and there can be many different way of calculating it.

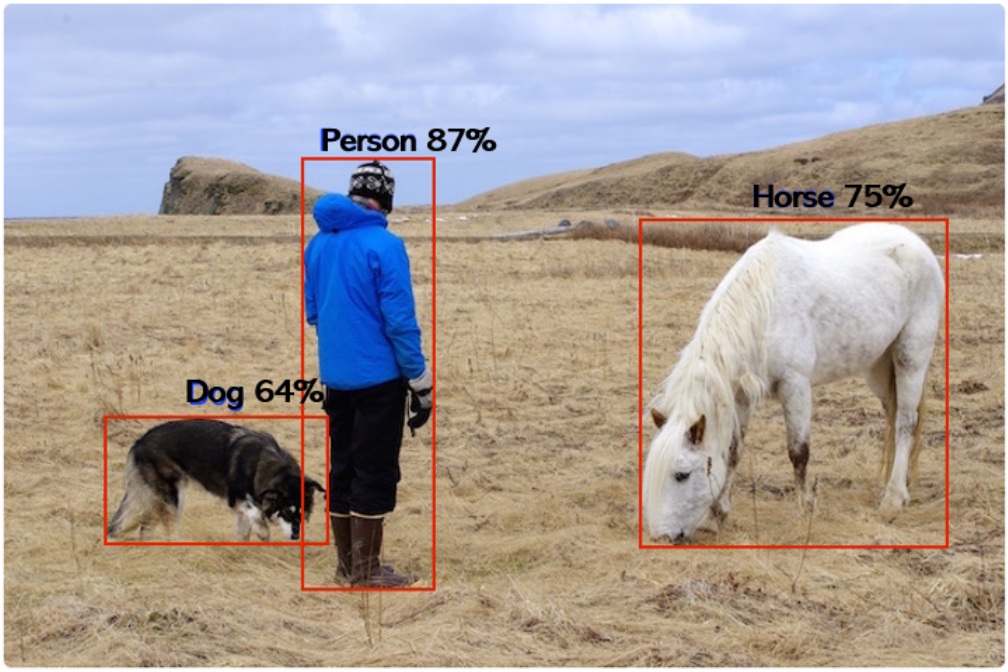


Figure 5.2: Ground Truth Bounding Boxes example.

Class	X coordinate	Y coordinate	Box Width	Box Height
Dog	100	600	150	100
Horse	700	300	200	250
Person	400	400	100	500

Figure 5.3: Bounding box coordinates example (1024x768 pixels image).

The simplest way would be to just report as confidence the value of the output node (after softmax has been applied) associated to the predicted class.
The problem of this approach is that it would make no difference between the case when many classes have similar high scores (confidence should be low) and when only one class have significantly higher score than all the others (confidence should be high).
Furthermore, other variables can be taken into account, such as the size of the bounding box predicted (smaller objects are more difficult to detect), hence every network uses a peculiar approach to determine the confidence of its predictions.

The challenge of object detection encloses the two different problems of classification and localization of objects. Hence two different metrics for evaluating the accuracy of a model for object detection have been designed: **mean Average Precision** (**mAP**) and **Intersection over Union** (**IoU**).

5.1.1 Intersection over Union

The evaluation metric used to discriminate whether an object has been correctly recognized and localized is based on looking at the **Intersection** and **Union areas** of the detected and ground truth bounding boxes, as shown in figure 5.4.

The minimum value of IoU accepted as a correct localization is a design choice,

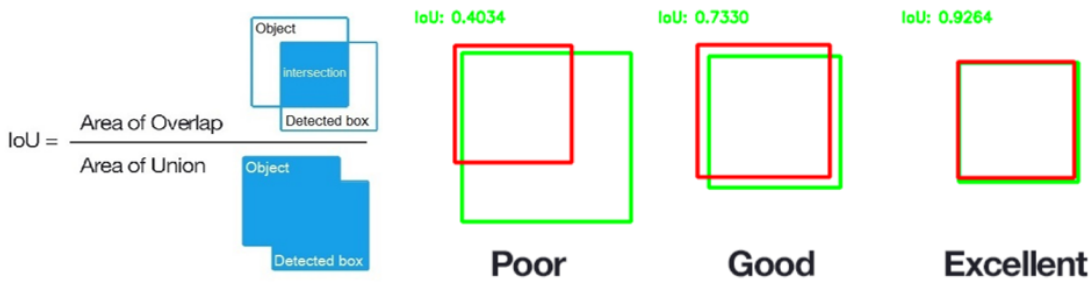


Figure 5.4: Intersection over Union (IoU) definition and examples.

which depends on whether it is more important to detect all objects, or to find their exact location.
As a matter of fact, if object detection is more important than exact localization, a lower minimum *IoU threshold* can be chosen (e.g. 0.5), otherwise, if correct localization is also important, an higher minimum value of IoU will be chosen (e.g. 0.7).

5.1.2 Mean Average Precision

In order to introduce the concept of **Average Precision**, the concepts of *Precision* and *Recall* introduced in section 3.8 will be employed.
One important note is that the relation between precision and recall depends on the quality and IoU threshold of the model, but they are generally inversely proportional, as shown in figure 5.5.

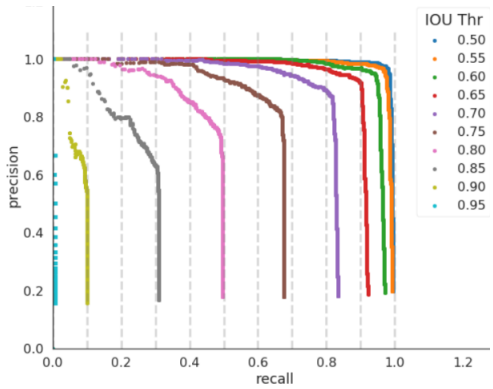


Figure 5.5: Precision-Recall relationship.

These precision-Recall points are computed rearranging all predictions by decreasing confidence, and then associating them to the corresponding ground truth (if the prediction in correct), in order to determine whether they are True Positives (TP) or False Positives (FP). If there are more than one predictions referring to the same ground truth BB, only the one with the highest confidence will be considered true positive.[13]
The **Average Precision** relative to each class predicted, can be represented as the area under the precision-recall curve drawn (readapted to be monotonically decreasing), with predictions sorted by decreasing confidence, as shown in figure 5.6.

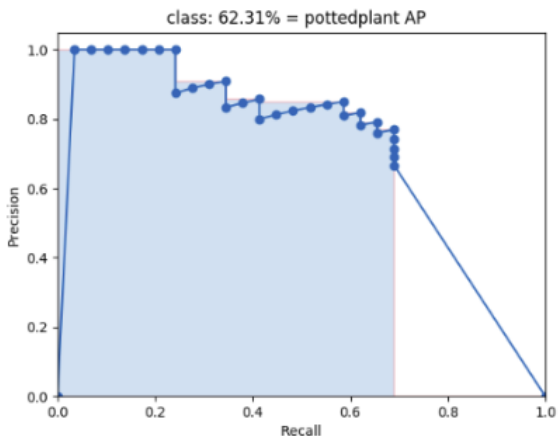


Figure 5.6: Average Precision (AP) Representation example.

The **Mean Average Precision** (**mAP**) is the mean value of all Average Precisions calculated for the possible classes predicted by the model.

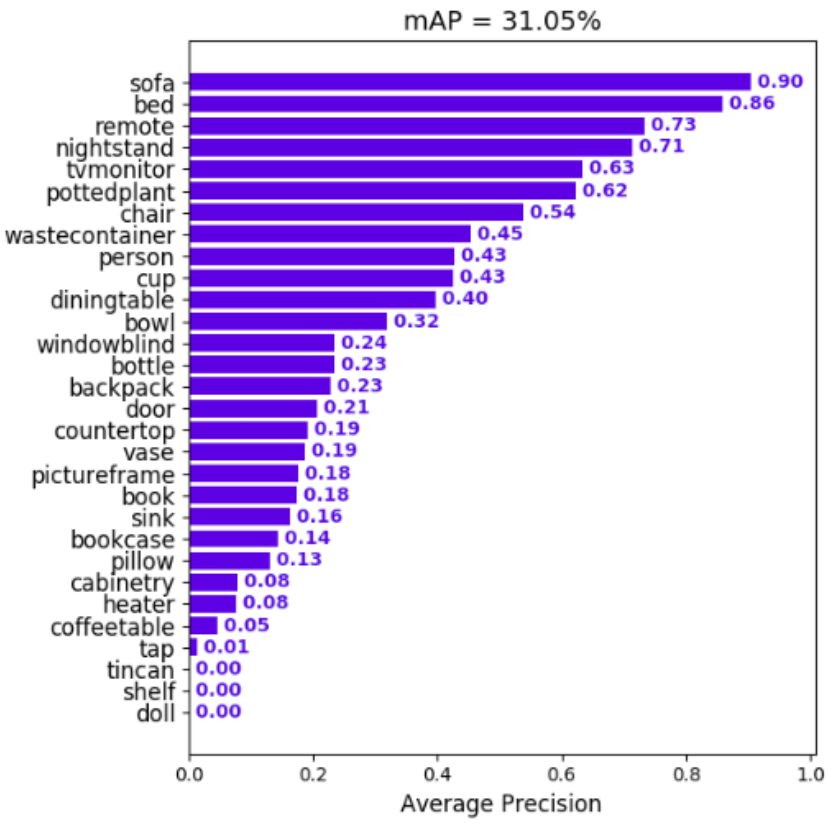


Figure 5.7: Mean Average Precision (mAP) example.

The mean Average Precision can be calculated only for a specific value of IoU threshold (e.g. IoU of 0.5 for the Pascal VOC challenge), or as the mean value of mAPs calculated for different Iou values (e.g. evaluation of COCO dataset is done with IoU spanning from 0.5 to 0.95 with a step of 0.05, and the result of some networks appears as figure 5.8).

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figure 5.8: Example of mAP results some CNNs on COCO dataset.

5.2 Object Detection Framework

Typically object detection is realized through three steps:

1. An algorithm (many different ones exist) is used to generate **Regions of Interest** (**RoI**), which are a large set of bounding boxes spanning throughout the image, chosen with a specific logic.

2. An image classification algorithm is applied to each region proposed, to understand which objects are present in each specific area.

3. A final synthesis of the results obtained is performed, where overlapping bounding boxes which detect the same object are combined into a single one.

5.2.1 Region Proposals: from R-CNN to SSD/YOLO

The first approach for the generation of some hypothetical RoI, was employed by the **R-CNN** architecture (figure 5.9), which uses an algorithm named **Selective Search** that recognizes shapes and colours in a picture generating a great number (1-2k) of regions.
Selective Search works by initializing many initial regions and expanding them by absorbing other adjacent regions which are most similar.
Each of the regions proposed by the Selective Search algorithm, will be **warped** (resized them into a fixed size), processed by a Convolutional Neural Network, and then classified using SVM, as shown in figure 5.9.[1]
This approach is extremely slow in both training and testing, because the classifier

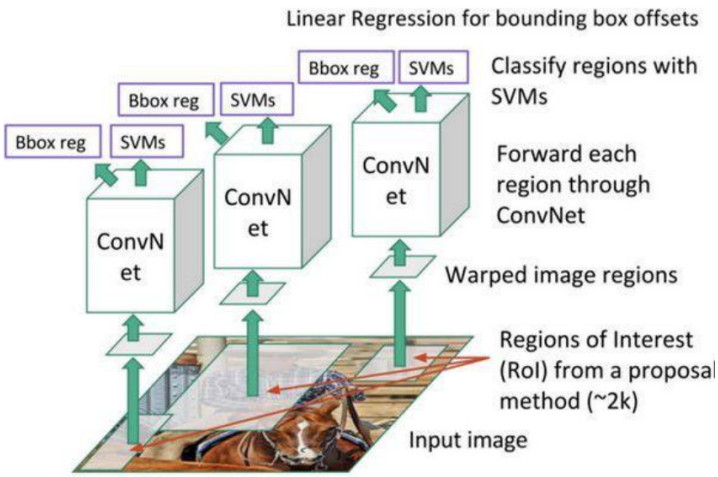
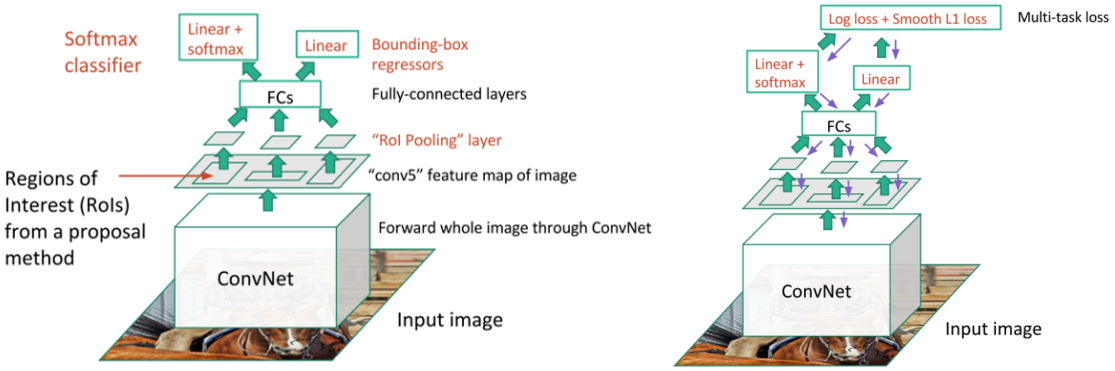


Figure 5.9: R-CNN architecture[5].

must be run 1000-2000 times for each training or testing image.

In order to speed up the process, **Fast R-CNN** was developed, which uses a CNN to generate the map of features, and then apply the region proposal algorithm on it. The structure of this algorithm is shown in figure 5.10a, and makes it possible to train all parameters at the same time, performing backpropagation throughout all layers (figure 5.10b).

Faster R-CNN further improves the speed by implementing also the region pro-



(a) Architecture. (b) Training.

Figure 5.10: Fast R-CNN.

posal algorithm with a Convolutional Neural Network, which can be trained, and strongly improves testing speed.

Figure 5.12 shows the progressive speed-up from R-CNN to Faster R-CNN (SPP-

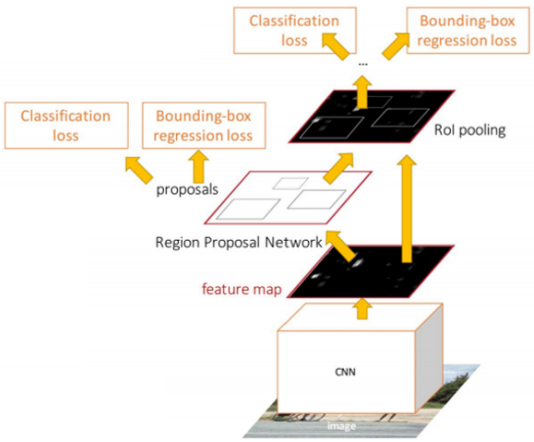


Figure 5.11: Faster R-CNN architecture.

Net is a step before Fast R-CNN where backpropagation could be performed only throughout the final layers).

The most modern object detectors eliminate the problem of region proposals slowing down the prediction process from the root, by avoiding to use such algorithms.

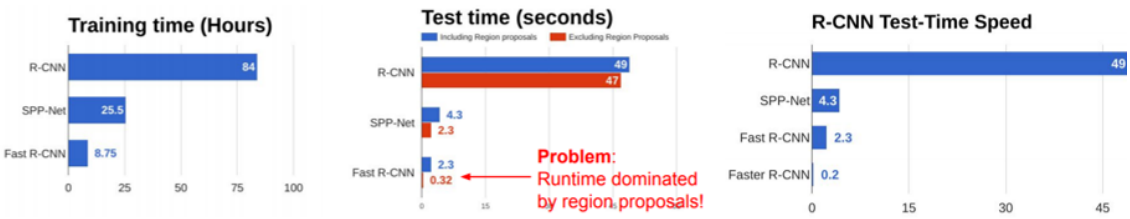
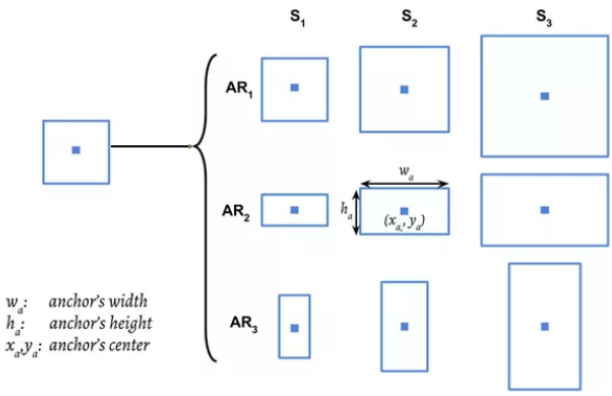


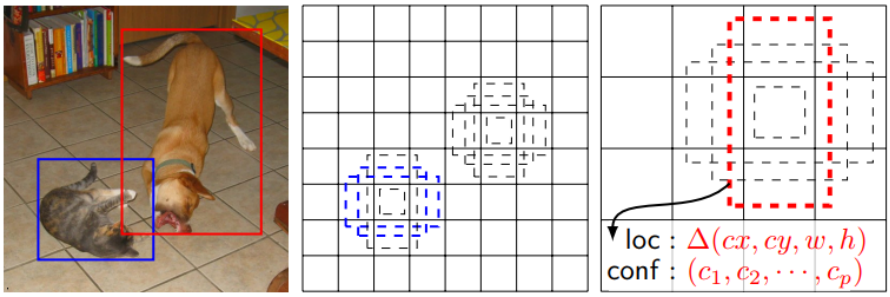
Figure 5.12: Comparison between different algorithms'speeds.

They start by generating automatically a default number and type (hyperparameters) of hypothetical bounding boxes, then they run classification on them, and choose only the significant ones.

This kind of approach guarantees a great speed-up of both training and testing,



(a) BB generated around one *c*entroid.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

(b) SSD approach example.[11]

Figure 5.13: Fast R-CNN.

which could enable real time image processing, but it intrinsically lowers the accuracy of the predictions.

Networks like **YOLO** (**You Only Look Once**) and **SSD** (**Single Shot Detector**)[11], use this technique, and they solve the problem of accuracy by adjusting the position of the final bounding box found, reaching a level of precision which is similar to that of networks that employ a region proposal network.

5.3 Main Object Detection Datasets available

The following is a list of the most common datasets for object detection that are available online as open source projects[10]:

- **Pascal VOC Challenge (2005-2012)**: reference dataset used (and updated every year) until 2012, with an increasing number of classes (from 4 to 20) and training examples (from 1.578 to 11.530). [4] Since 2009 it started featuring also segmentation, however most modern networks consider this dataset too small and prefer to evaluate their performance using bigger datasets.

- **MS-COCO** (MicroSoft Common Objects in Context)[9]: dataset of 330K images with 5 captions per image in 91 categories, used for object detection and object segmentation.

- **ImageNet**[7]: follows the *WorldNet*[18] hierarchy of phrases and provides around 1K images for each phrase (1.5M total images). It is extremely large and used for object detection.

- **KITTI** (Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago): dataset with 15K images taken from a camera on top of a car while driving. It features 9 classes (Car, Van, Truck, Pedestrian, Person sitting, Cyclist, Tram, Misc, DontCare), and it can be useful for training of autonomous driving systems. [6]

- **Playing For Benchmarks**: dataset of 254K synthetic images taken from *Grand Theft Auto V* with different weather conditions, labeling objects from 32 classes. [16]

Chapter 6

Experimental Results

6.1 Objective

The purpose of the training performed, was to quantify the degradation of performance of object detection when training is done on synthetic images, with respect to when training is performed on real images.
This was done because generating a training set of digital images is simple if the source code of the images is available.
As a matter of fact, the source code generates objects in the image, therefore it automatically knows their position, and the ground truth bounding boxes don't have to be manually drawn.

The network chosen to realize this test was SSD (Single Shot Detector) [11], because it is one of the most common state-of-the-art networks, and the datasets chosen were **Playing for Benchmarks** [16] (synthetic) and **KITTI** [6] (real).

6.2 Training Data Organization

Due to the difference between available classes in the two datasets, they were both filtered, resulting in a model with 4 classes: *Car, Van, Truck and Person.* In particular KITTI had two separated classes for *Pedestrians* and *Person Sitting*, and for the scope of this training they were merged into one class (*Person*).

The dataset KITTI contains 7480 training images (the testing images were not used in the experiment because the labels are not public), thus the images available were randomly separated into 6000, 500 and 980 for the training set, cross-validation set and testing set.
During the training performed with synthetic images, all labeled examples in the Playing For Benchmarks dataset were used for the training process, and the test

was performed on 980 images randomly selected from KITTI.

6.3 Experimental Procedure

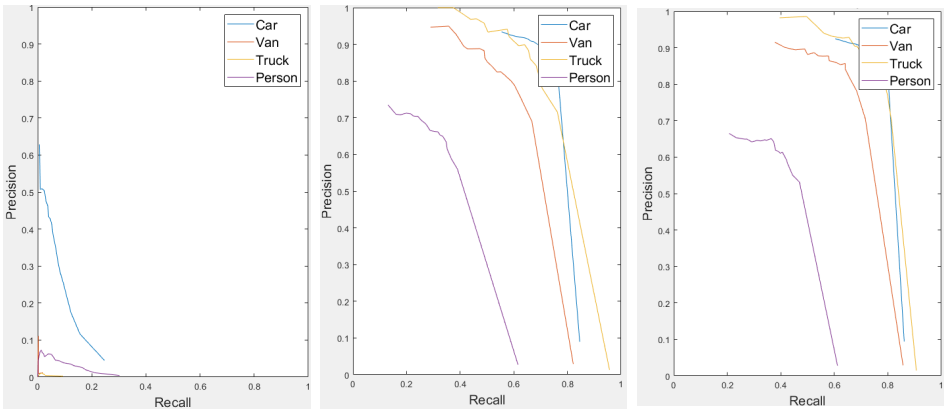
- Training and testing SSD over real images (150.000 training steps).
- Training SSD with synthetic images (150.000 training steps) and testing over real images.
- Training SSD with real images (150.000 training steps), but after initializing the weights as those obtained by the training performed using synthetic images, and then testing on real images. This process is defined *finetuning*, because it works on synthetic data until a checkpoint and then improves the results obtained by continuing the training over real images.

The training was stopped at 150.000 steps and not pursued until convergence in order to save some training time, because the results obtained with this method can be generalized to a complete training.

6.4 Results

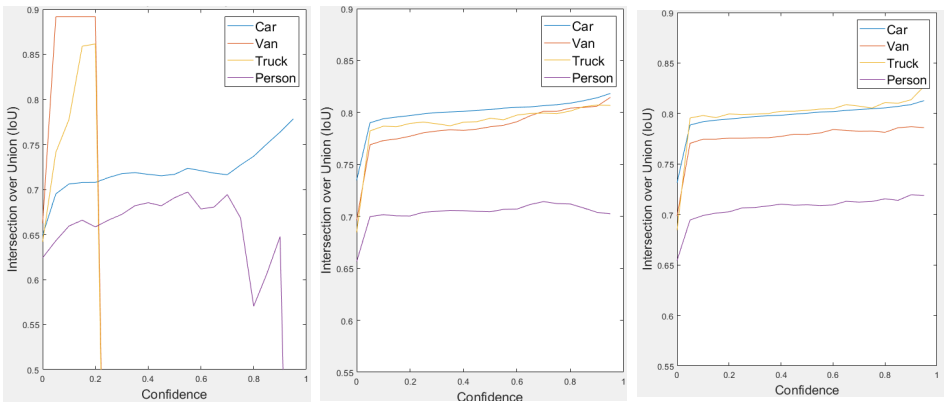
As it can be visualized in figures 6.2 (AP) and 6.1 (IoU), the testing results obtained for training with synthetic images were not satisfactory, because while the IoU showed comparable results between the two options, the Average Precision of the first training (synthetic images) was extremely low, denouncing the fact that the model produced fails to detect the great majority of objects.
However the results of the finetuned model had better results than training on real images for both IoU and AP of all classes, proving that the information learned from synthetic images has been useful to improve the performances of the model.

The numerical data are summarized in figure 6.3, and they show more evidently the comparison between training with real images and finetuning, which was difficult to detect in the plots.
The average improvement due to finetuning recorded, has been of more than 3% (3.2% extra IoU and 3.5% extra mAP), which is a satisfying result.



(a) Synthetic Images. (b) Real Images. (c) Finetuning.

Figure 6.1: AP Experimental Results.



(a) Synthetic Images. (b) Real Images. (c) Finetuning.

Figure 6.2: IoU Experimental Results.

IoU Results

Method	IoU	Car	Van	Truck	Person
Synthetic Images	**43.85**	71.99	*21.14*	*19.41*	62.85
Real Images	**75.68**	79.99	78.48	78.91	70.33
Finetuning	**76.93**	79.69	77.55	79.86	70.63

mAP Results

Method	mAP	Car	Van	Truck	Person
Synthetic Images	**1.53**	5,2	*0.05*	*0.04*	0.83
Real Images	**57.79**	71.24	60.08	72.28	27.56
Finetuning	**61.34**	74.04	63.87	76.97	30.49

Figure 6.3: mAP and IoU numerical results.

A visualization of the predictions obtained by the different models is presented in figures 6.4 and 6.5, and it shows that most of the time the model trained on synthetic images struggles to make accurate predictions, and its confidence is usually low.

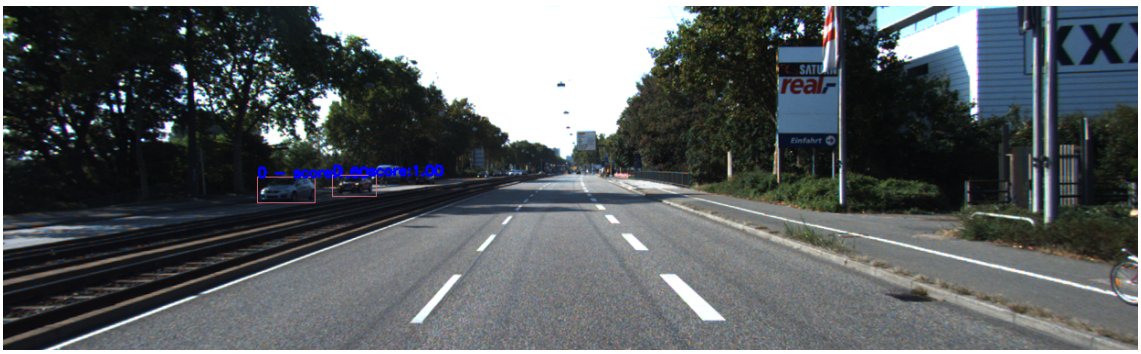
(a) Visualization of the Ground Truth.

(b) Visualization of the model trained with real images.

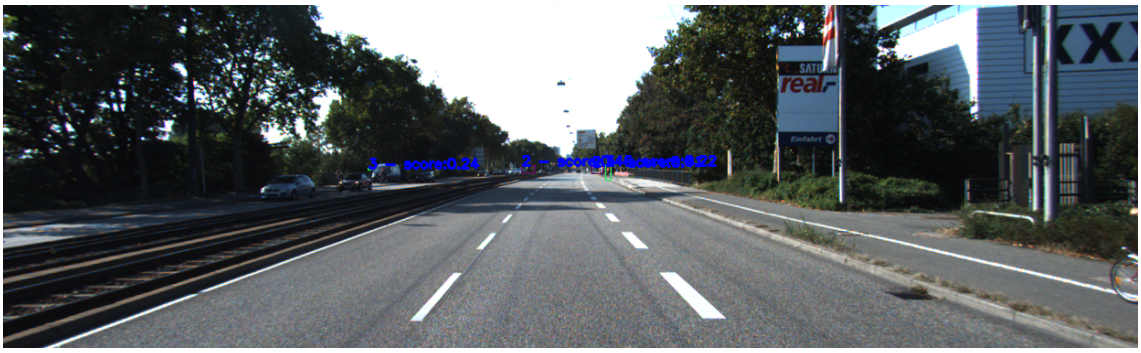
(c) Visualization of the model trained with synthetic images.

Figure 6.4: Visualization result example (1).

(a) Visualization of the Ground Truth.



(b) Visualization of the model trained with real images.



(c) Visualization of the model trained with synthetic images.

Figure 6.5: Visualization result example (2).

Chapter 7

Conclusion

Regarding training on synthetic images, the outcome of the experiment has been worse than expected, because the *background noise* of synthetic images and of real images are really different, even if they look similar at a human eye.
This *semantic gap* is known as the problem of the **domain shift**, and in order to improve this result, it would be useful to perform **Domain Adaptation** on synthetic images before training, in order to make them more similar to real ones, but this technique is still an active field of research.

Anyhow, the result obtained about finetuning is valid, and shows that pre-training the network with synthetic images can make it possible to learn some useful features, which prove to be better than random initialization or initialization with weights taken from training over a completely different dataset (e.g. ImageNet).
This kind of pre-training could be particularly useful when training sets of real images are not available or of small dimension (e.g. training sets that need to be labeled specifically for an application), because if the training images are not enough, the network would probably overfit the training set.

Bibliography

[1] *CS231n: Convolutional Neural Networks for Visual Recognition (Stanford University)*. http://cs231n.stanford.edu/. 2017 lectures.

[2] *Decision Tree Algorithm*. https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/.

[3] *Deep Learning Algorithms for Object Detection*. https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852.

[4] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338.

[5] *Fast and Faster R-CNN networks*. https://jhui.github.io/2017/03/15/Fast-R-CNN-and-Faster-R-CNN/.

[6] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[7] *ImageNet Website*. http://image-net.org/.

[8] *K-means Clustering algorithm*. https://www.datascience.com/blog/k-means-clustering.

[9] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[10] *List of open source deep learning datasets available*. https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets/.

[11] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[12] *Machine Learning Course on Coursera from Stanford University (Andrew Ng)*. https://www.coursera.org/learn/machine-learning/home/welcome.

[13] *mean Average Precision synthesis*. https://github.com/Cartucho/mAP.

[14] Thomas Mitchell. *Machine Learning*. McGraw Hill Professional, 1997.

[15] *Principal Component Analysis*. https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c.

[16] Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun. "Playing for Benchmarks". In: *CoRR* abs/1709.07322 (2017). arXiv: 1709.07322. URL: http://arxiv.org/abs/1709.07322.

[17] *Support Vector Machine*. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[18] *WorldNet website*. https://wordnet.princeton.edu/.