République Algérienne Démocratique et Populaire
Université Abou BakrBelkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Modèle Intelligent et Décision (M.I.D)

Object Localization In Computer Vision

Réalisé par :

- MESMOUDI Qwider.
- ZIGH Mohamed.

Présenté 15 Juillet 2021 devant le jury composé de.

● *Mr BElABED Amine.*	*(Président)*
● *Mr HADJILA Fethallah.*	*(Encadreur)*
● *Mr SMAHI Mohamed Ismail.*	*(Examinateur)*

JUILLET,2021

Acknowledgement

First of all, we would like to thank **ALLAH** the most merciful, for His help and generosity.

Second, we would like to thank our supervisor **Mr.HADJILA Fethallah**, for his supervision, continuous caring and guidance discussion.

We also thank the jury members for their time and dedication.

Finally, our deep and sincere gratitude to our family for their continuous and unparalleled love, help and support. We are grateful to our parents for their support and encouragement in all times.

Contents

Acknowledgement **1**

Content **3**

List of Figures **5**

1 Introduction **6**

2 Introduction to object localization methods **8**

2.1 Introduction to computer vision . . . 8
2.2 Challenges . . . 8
2.3 Low-level image processing . . . 9
2.3.1 Image Enhancement . . . 10
2.3.2 Image Restoration . . . 10
2.4 Medium-Level Image Processing . . . 11
2.4.1 Image segmentation . . . 11
2.5 High-Level Image Processing . . . 13
2.6 Some techniques for object localization . . . 14
2.6.1 Haar cascade . . . 14
2.6.2 Histogram of Oriented Gradients . . . 16
2.7 conclusion . . . 18

3 Deep learning **19**

3.1 Introduction . . . 19
3.2 An artificial neuron . . . 19
3.2.1 Structure . . . 19
3.2.2 behaviour . . . 19
3.3 Activation function . . . 20
3.3.1 The Sigmoid function . . . 20
3.3.2 Tanh (Tangent Hyperbolic function) function . . . 20
3.3.3 Rectified Linear Unit ReLU . . . 21
3.4 Artificial neural network . . . 21
3.5 Loss function . . . 22
3.6 Gradient Descent . . . 22
3.7 Back propagation algorithm . . . 24
3.8 Regularization . . . 25
3.8.1 Early stopping . . . 25
3.8.2 L2&L1 . . . 26
3.8.3 Dropout . . . 26
3.9 Convolutional Neural Network CNN . . . 27

3.9.1 convolutional layer 28
3.9.2 Pooling layer 29
3.9.3 CNN's architectures 30
3.9.4 LeNet-5 30
3.10 Conclusion 31

4 Contribution 32
4.1 Introduction 32
4.2 Data Set 32
4.3 Development tools 33
4.3.1 Software tools 33
4.3.2 Hardware tools 34
4.4 Building the model 34
4.4.1 Model selection 34
4.4.2 Evaluation metric&Loss function 40
4.5 Development process 41
4.6 Results and Discussion 43
4.7 Conclusion 46

General conclusion 47

List of Figures

2.1 Limitation of edge extraction in computer vision 9
2.2 Variation between instances of the same class 9
2.3 Digital image processing : image enhancement.[18] 10
2.4 Deblurring images using a Wiener filter. 11
2.5 Tresholding. 12
2.6 Edge detection. 12
2.7 Region-based segmentation 13
2.8 Object recognition. 14
2.9 Haar features.[20] 14
2.10 Face detection using Haar features.[20] 15
2.11 Kernels.[22] 16
2.12 Gradient magnitude and direction.[22] 17
2.13 Histogram of Gradients.[22] 18

3.1 Structure of an artificial neuron.[13] 20
3.2 Sigmoid function 20
3.3 TanH function 21
3.4 ReLu function 21
3.5 Schematic representation of a neural network. 22
3.6 a convex function. 23
3.7 Large learning rate.[21] 23
3.8 A small learning rate.[21] 23
3.9 Forward pass.[12] 24
3.10 Backward pass.[12] 25
3.11 Early stopping.[6] 26
3.12 Dropout.[6] 27
3.13 Features extraction.[3] 27
3.14 Convolutional layer.[11] 28
3.15 Convolutional layers output.[6] 29
3.16 Max and Mean pooling.[12] 29
3.17 Max pooling layer.[6] 30
3.18 VGG layers.[11] 30
3.19 LeNet-5architecture.[10] 30
3.20 LeNet-5 structure. 31

4.1 Sample from the Oxford-IIIT Pet Datase. 33
4.2 Transfer learning.[6] 35
4.3 Final architecture 40
4.4 Intersection Over Union.[6] 41
4.5 The training and validation losses,IoUs and accuracies 43
4.6 Accuracy and IoU metrics. 43

4.7 The training and validation losses,IoUs and accuracies 43
4.8 Accuracy and IoU metrics. 44
4.9 The training and validation losses,IoUs and accuracies 44
4.10 Accuracy and IoU metrics. 44
4.11 The training and validation losses,IoUs and accuracies. 44
4.12 Accuracy and IoU metrics. 45
4.13 Prediction vs target. 45
4.14 IoU Summary 46

Chapter 1

Introduction

Context

The human visual system is an accurate and fast architecture allowing us to perform complex tasks like driving cars, walking, or any daily activities. In this context, The computer vision can be viewed as a discipline that allows the machines to better understand the real scenes and perform the aforementioned tasks. we can notice several problems that constitute the backbone of computer vision such as object recognition, object localization, semantic segmentation, and image captioning. The present work falls into the category of machine learning models that enhance the performance of existing prototypes in object localization issues.

Problematic

Our main goal is to handle the object localization task in the computer vision field. The key difficulty is to locate the bounding box that contain the main object of the image. We also notice that these objects can suffer from different issues, such as contrast variation, background variability, and occlusion.

Contribution

Solving the localization problem using deep learning approaches depends on the neural network configuration. One of the main drawbacks of neural networks is its flexibility, because there are several hyperparameters to tweak, and their tuning is still an active area of research. Our work introduces a deep neural network architecture that handles the localization task of the Oxford-IIIT Pet Dataset. The main goal is to achieve an accuracy higher than 95% and Intersection over Union (IoU) higher than 60%. to this end, we leverage the transfer learning and search for a pretrained CNN model that gets a higher performance (e.g., VGG16, VGG19 and Xception). In addition, we evaluated many combinations of hyperparameters (like adding layers, and loss function weighting) and and derive the best model.

Manuscript plan

Chapter 1 :
The first chapter contain an introduction to the object localization methods. We will see

some challenges, and some technique used for image processing .

Chapter 2 :
The second chapter will cover general information about Neural Networks and Deep Learning.

Chapter 3 :
The third chapter shows the architecture used for the localization problem, and the techniques used for getting better results.

General conclusion :
In this part, we summarize our main contribution and give insights about future improvements.

Chapter 2

Introduction to object localization methods

2.1 Introduction to computer vision

Computer vision is a field of study focused on the problem of helping computers to interpret and understand the visual world. At an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world [15].

The purpose of computer vision is to understand a scene or features in an image. Typically, this involves developing methods that seek to reproduce the capability of human vision.

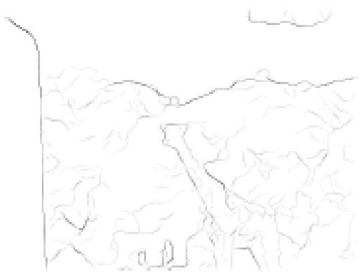
Understanding the content of digital images may involve extracting a description from the image, which may be an object, a text description, a three-dimensional model, etc. Computer vision is the automated extraction of information from images. Information can mean anything from 3D models, camera position, object detection and recognition to grouping and searching image content[16].

2.2 Challenges

While human can achieve visual tasks like object detection and recognition in complex scenes easily, this is not the case for artificial systems. It has been proven that contour-based object detection and recognition in complex scenes is one of the most challenging problems in computer vision. So, implementing a fast, efficient and reliable algorithm to do so is a difficult task.

There are many problems that complicate the ability to detect and recognise an object within a complex scene. One of these problems is getting a clean edge map of the scene by low-level image processing. Even with a state-of-the-art edge-detection algorithm, like the Berkeley edge detector [16], we will miss parts of the contours in the presence of clutter [see Figure 2.1].

(a) An image of a giraffe in a complex scene.


(b) Edge map of the giraffe using the Berkeley edge detector.

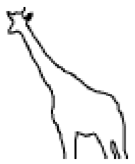
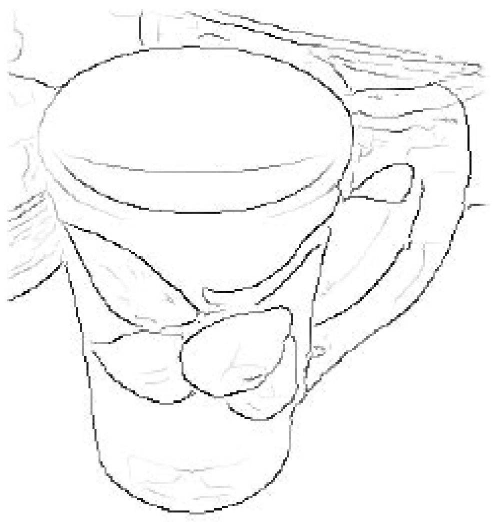

(c) The real contours of the giraffe marked manually.

Figure 2.1: Limitation of edge extraction in computer vision

In addition, objects from the same class may differ significantly, which is referred to as intra-class variation [see Figure 2.2].


(a) Slim mug

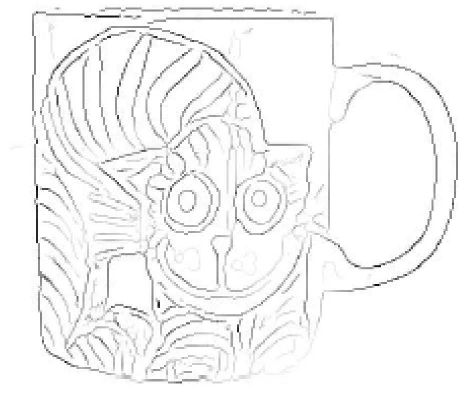

(b) Wide mug

Figure 2.2: Variation between instances of the same class

Another reason that makes recognizing an object such a challenging problem is the complexity of the scene. In real-world images all types of complications can be expected :

- Occlusion : is the basic element that limits the information in an image.
- clutter : means there's a lot of objects in the image and it's hard for an observer to focus on a particular object.
- changes in scale and viewpoint of the objects and scenes.

2.3 Low-level image processing

Low level image processing is mainly concerned with extracting descriptions from images (that are usually represented as images themselves) [17], and both input and output are images. This operation involves tasks such as image preprocessing, image sharpening, contrast enhancement, etc.

2.3.1 Image Enhancement

Image enhancement refers to the process of highlighting certain information of an image, as well as weakening or removing any unnecessary information according to specific needs. For instance, eliminating noise, revealing blurred details, and adjusting levels to highlight features of an image [see Figure 2.3].

Image enhancement techniques are often divided into 2 broad categories:

- **Spatial domain :** refers to the image plane itself and are based on direct manipulation of pixels in an image.
- **Frequency domain :** enhancement obtained by applying the Fourier Transform to the spatial domain.

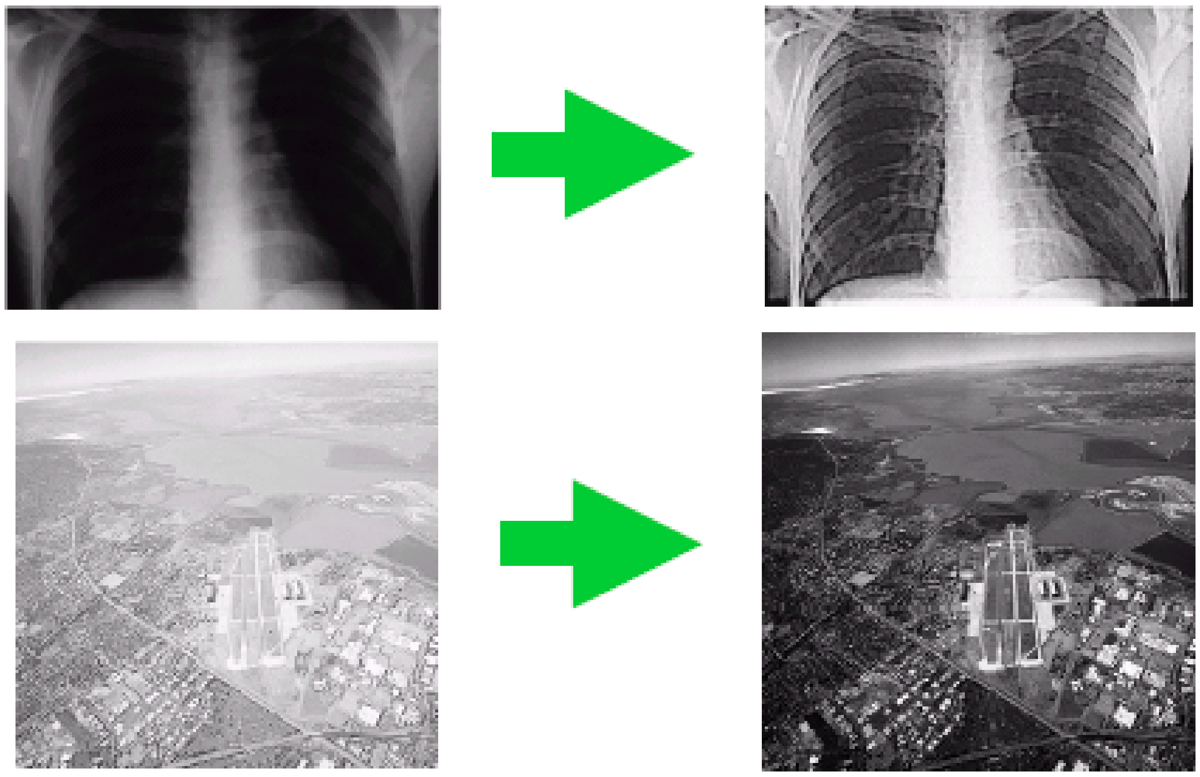


Figure 2.3: Digital image processing : image enhancement.[18]

2.3.2 Image Restoration

Image restoration is based on the attempt to improve the quality of an image through knowledge of the physical processes which led to its formation [18]. The main goal of image restoration is to improve the quality of an image. That involves recovering an image that has been degraded [see Figure 2.4]. Degradation comes in different forms such as motion blur, noise, and camera misfocus. It is possible to come up with a good estimate of the actual blurring function and ”undo” the blur to restore the original image.

In cases where the image is corrupted by noise, the best we can do is to compensate for the degradation caused. Here are some techniques of image restoration : Inverse filter, the median filter, the wiener filter, and adapting filter.

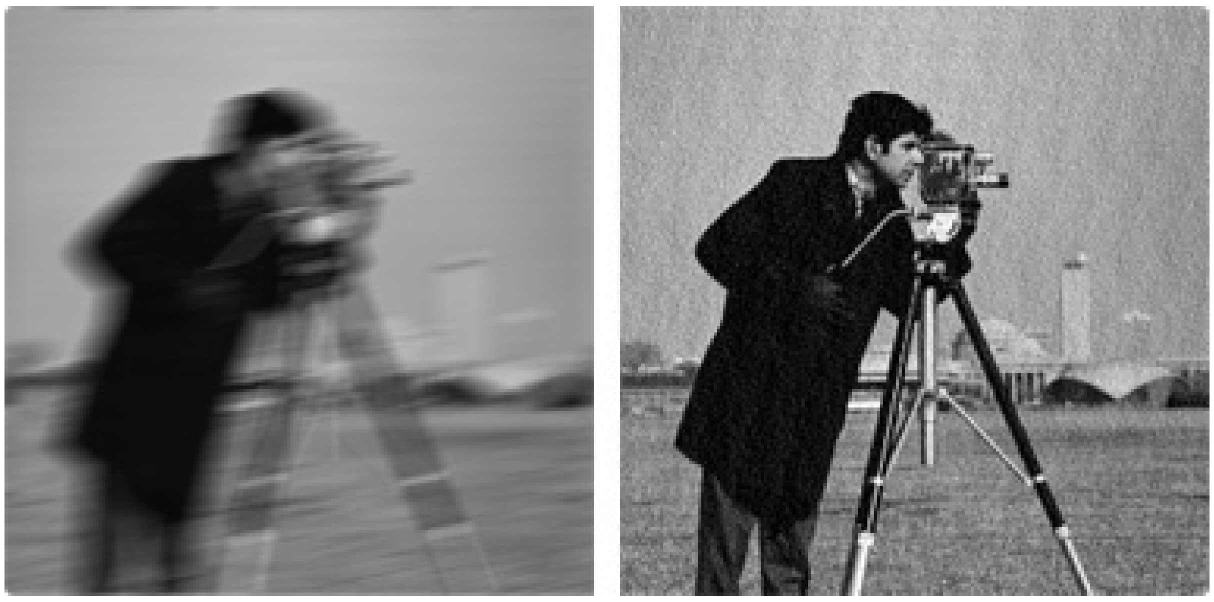


Figure 2.4: Deblurring images using a Wiener filter.

Difference between image enhancement and image restoration

Image enhancement is a subjective processing technique, and as the name suggests, the original image is processed so that the new image is better than the original. In the other hand the image restoration image restoration is an objective processing technique that tries to fix the image to get back to the real, true image, That means improving the image to match the original one.

2.4 Medium-Level Image Processing

The middle level of image processing is concerned with extracting descriptions of the scene from the image descriptions extracted at the low level.

The mid-level processing involves tasks such as description of images, image segmentation, object recognition, etc. the inputs are generally images but its outputs are generally image attributes. The analysis usually does not know anything about the content of the scene, but does use a lot of knowledge of scene shape and how shape appears in an image.

2.4.1 Image segmentation

Image segmentation is an image processing operation which aims to bring pixels together according to predefined criteria. Many techniques are used for the image segmentation, these techniques are chosen based on the type of processing that is needed. Here are some techniques :

Threshold Method

Gray-level thresholding is the simplest segmentation process. Many objects or image regions are characterized by constant reflectivity or light absorption of their surfaces; a brightness constant or threshold can be determined to segment objects and background [16].[see Figure 2.5]

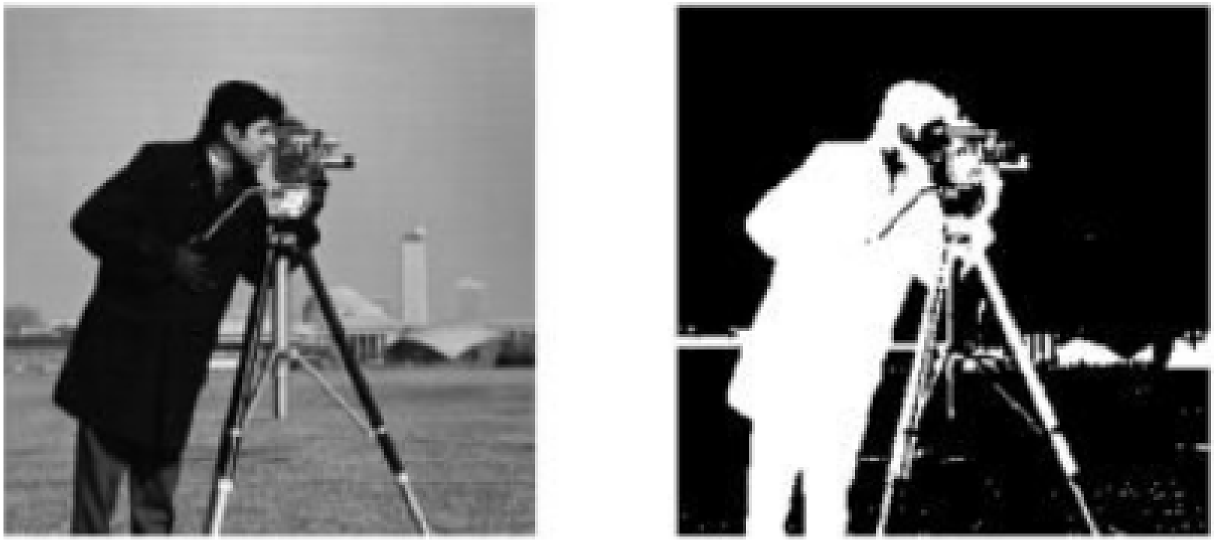


Figure 2.5: Tresholding.

Edge Based Segmentation

Edge detection is the process of locating edges in an image. it is one of the earliest segmentation approaches and still remains very important [16]. [see Figure 2.6]

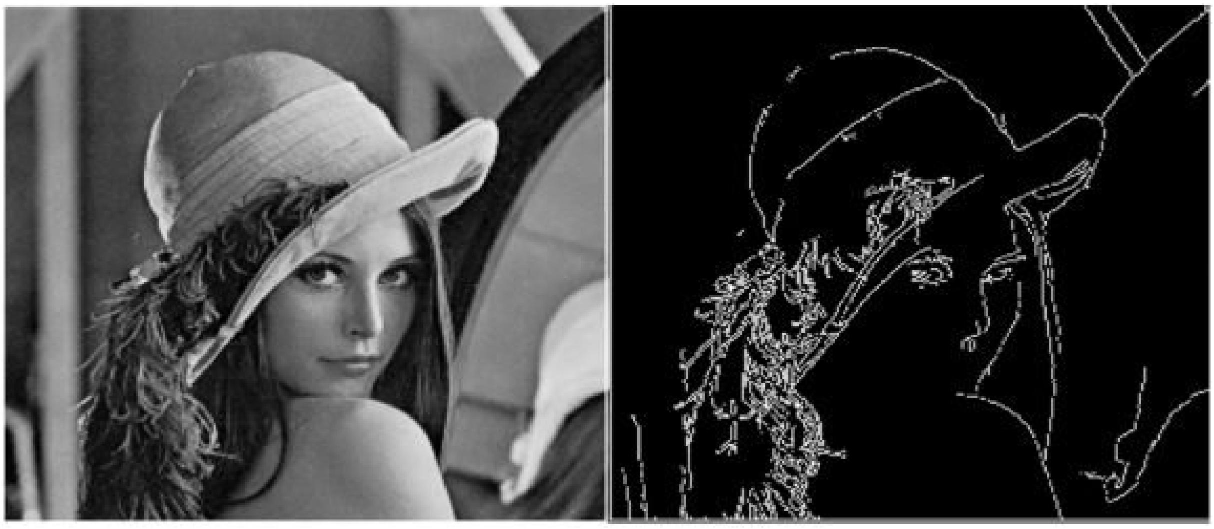


Figure 2.6: Edge detection.

Region Based Segmentation

The region-based segmentation method looks for similarities between adjacent pixels, and the pixels that possess similar attributes are grouped into the same regions.[see Figure 2.7]

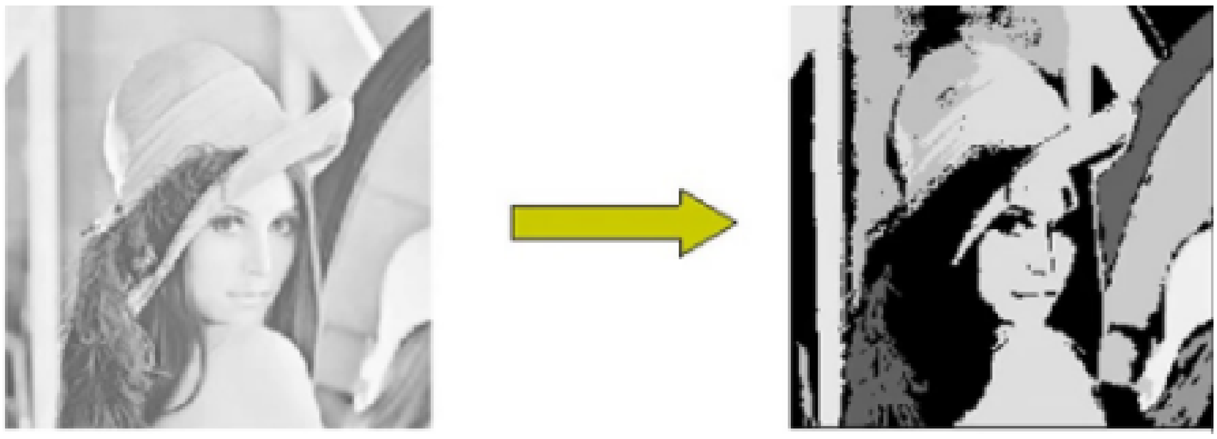


Figure 2.7: Region-based segmentation

2.5 High-Level Image Processing

High-Level processing means making sense from a group of recognized objects in the image. In the high-level processing we will talk about object recognition. Which is a general term to describe a set of related computer vision tasks that cover identifying objects in images. We will be using the term object recognition broadly to encompass both image classification (a task requiring an algorithm to determine what object classes are present in the image) as well as object detection (a task requiring an algorithm to localize all objects present in the image [8]. Object recognition can perform different tasks, depending on the type of information needed from the image.

- **Image classification** (object recognition) : consists in recognizing to which category an object belongs.[see Figure 2.8]
- **Object localization** : this task consists in finding where is the object in the image and drawing a bounding box around it.[see Figure 2.8]
- **Object detection** : classify and locate every single object within the image. Assign a class to every object and draw a bounding box around it.[see Figure 2.8]
- **Instance Segmentation** : classify every pixel in the image to a class so that each pixel is assigned to a different instance of an object. [see Figure 2.8]
- **Image captioning** : is the process of generating textual description of an image. It uses Natural Language Processing and Computer Vision to generate the description.
- **Object tracking** : means estimating the state of the target object present in the scene from previous information.

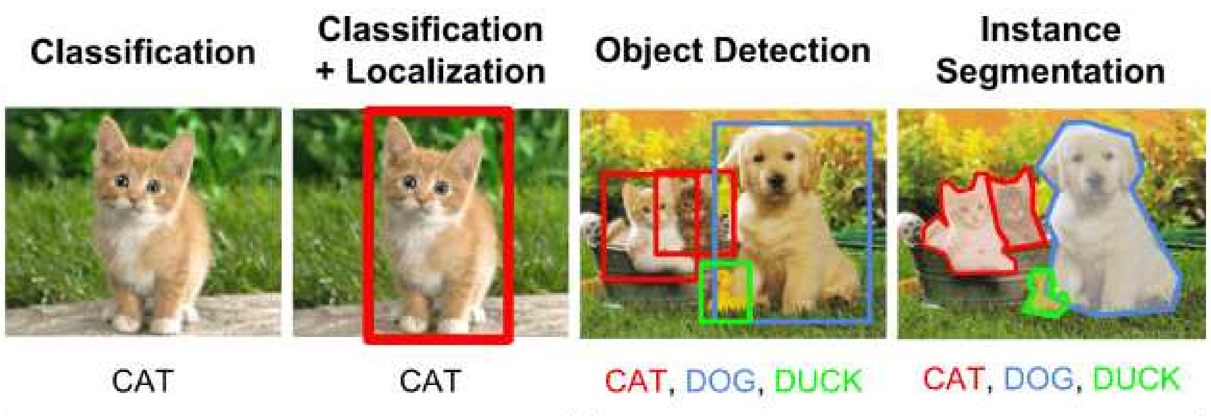


Figure 2.8: Object recognition.

2.6 Some techniques for object localization

2.6.1 Haar cascade

Object detection using Haar feature-based cascade classifiers is One of the oldest technique, used for detection before the deep learning era had even started, it was proposed by Paul Viola and Michael Jones in their paper [20]. It is an effective method and works well for face detection. The concept is very simple, given a feature set and a training of positive and negative images (for face detection positive means images with faces ,but negative means images without faces) ,any machine learning approaches can be used to learn a classification function , And then it used to detect objects in other images. In order to extract features from images, like edges or lines Haar feature shown below are used [see Figure 2.9]. Each feature is a single value, which is calculated by subtracting the sum of pixels under the white rectangle, from the sum of pixels under the black rectangle, if the obtained result is greater than a given threshold that mean the feature represented by the Haar feature is detected.

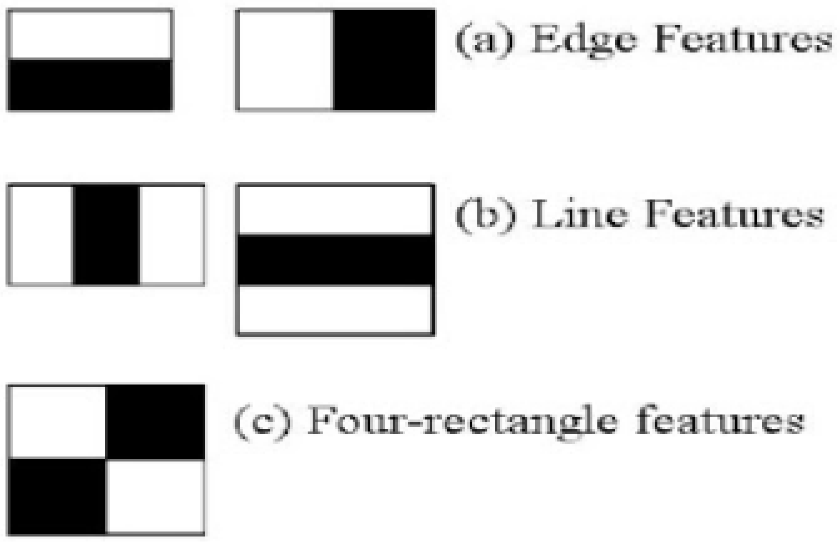


Figure 2.9: Haar features.[20]

Since all sizes and location of each kernel are used to generate an enormous number of features, (The authors mentioned that they used about 180 000 rectangle features (Haar features)). they introduce a concept known as the integral image, to perform the same operation mentioned before, but this time only four pixels are used for calculation, and with this the time complexity is reduced.

Not all the features are useful, some of them are irrelevant for example in the image below [see Figure 2.10] For example, consider the image below. In The top row The first feature of The top row seems to encode the property of the region of the eyes is often darker than the region of the nose and cheeks. The second feature encode the property of eyes are darker than the bridge of the nose. But they both applied to cheeks or other place and that is irrelevant, thus a feature selection is needed to keep only the important features, and this is achieved by AdaBoost .

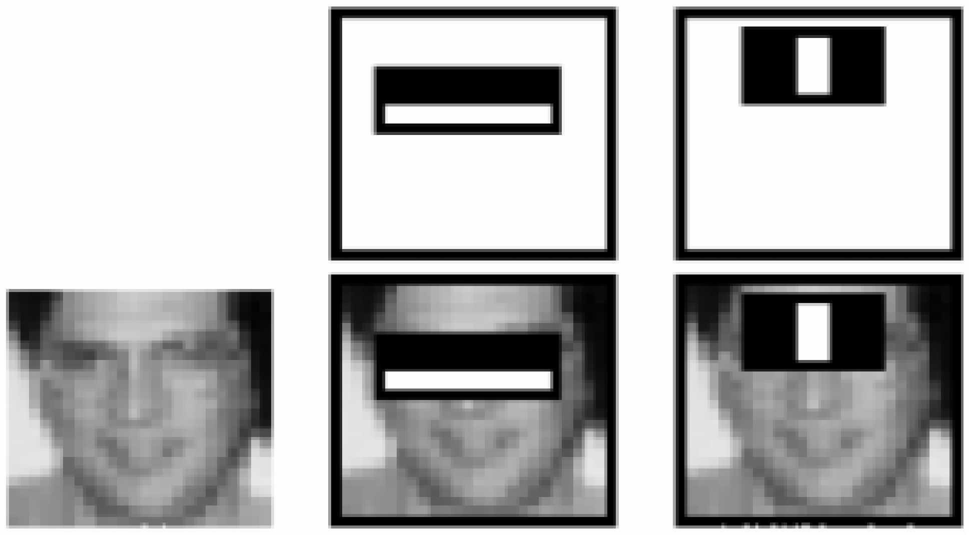


Figure 2.10: Face detection using Haar features.[20]

In order to select the features, they all applied on the training images. For each feature the weak learning algorithm determines the best threshold which will best separate the faces, the features are selected only with minimum error rate (the features that most accurately classify the face and non-face images). The final classifier is a weighted sum of these weak classifiers. The authors said that only 6000 are selected in their experiment.

For the test phase ,the 6000 features are applied to each 24x24 window of an image, then check if it is face or not, it seems it is a little inefficient and time consuming.

For this, they introduced their third contribution, which is the concept of Cascade of Classifiers. Which consist of not all the features need to run on each and every window, instead of that the features are grouped into different stages of classifiers and applied one-by-one. If a window fails the first stage, get rid of it. No need to consider the remaining features on it. If it passes, the second stage of features is applied and continue the process. The window which passes all stages is a face region.

The author's detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages.

2.6.2 Histogram of Oriented Gradients

In traditional computer vision approaches a step of feature extraction is important ,and designing these features affect the performance of the machine learning algorithm.there are many techniques used for feature extraction and one of these is The Histogram of Oriented Gradients(HOG). It is a method introducedby Dalal and Triggs in their paper[22],they proved that combining the Histogram of Oriented Gradients (HOG) image descriptor with Linear Support Vector Machine (SVM) lead to train highly accurate object classifiers especially in their study human detector.

feature descriptors

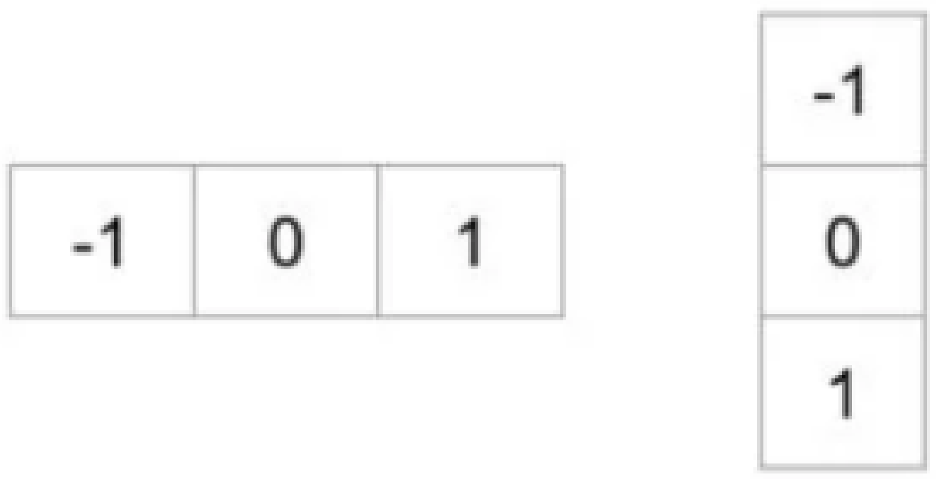
A feature descriptor is the representation of an image that extracts only the useful information and disregards the unneeded information from the image.Typically, a feature descriptor converts an image of size width x height x 3 (channels) to a feature vector / array of length n. In the case of the HOG feature descriptor, the input image is of size 64 x 128 x 3 and the output feature vector is of length 3780.

The following steps should show how to calculate the HOG feature descriptor :

- Preprocessing : The image is reprocessed so it has a fixed aspect ratio of (width:height) is of 1:2, so the images can be 200x400, 300x600, etc. The image size should preferably be 64 x 128. That will make the the calculation simple after dividing the image by 8x8 and 16x16 patches in order to extract the features.

- Calculating the Gradients :

  To make the HOG feature descriptor, as discussed before, we need to first calculate the respective horizontal and vertical gradients to provide the histogram that can be used later in the algorithm. This can be done by simply filtering the image with the following kernels (see Figure 2.11) :



-1	0	1

-1
0
1

Figure 2.11: Kernels.[22]

  Next, the magnitude and the direction of the gradients can be found by using the following formulas :

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

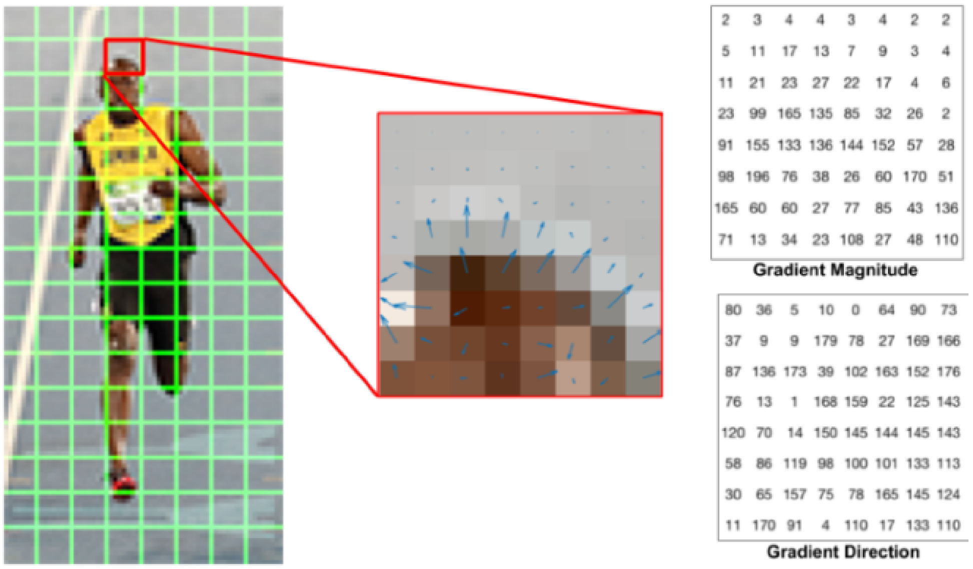
The magnitude of the gradient increases wherever there is a sharp change in intensity.So, the unnecessary information is removed like the background and only the essential parts remain like edges.

- Making a histogram from these gradients :

  Now, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells

  Feature descriptors will allow for a concise representation of particular patches of the images; an 8x8 cell can simply be explained using 128 numbers (8x8x2 where the number 2 represents magnitude and directional values). calculating a histogram over a patch makes this representation more robust to noise.

  The histogram is essentially a vector (or an array) of 9 bins (numbers) corresponding to angles 0, 20, 40, 60 . . . 160. The following example shows how an image with the respective gradient magnitudes and directions can look like (notice the arrows get larger depending on the magnitude).



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

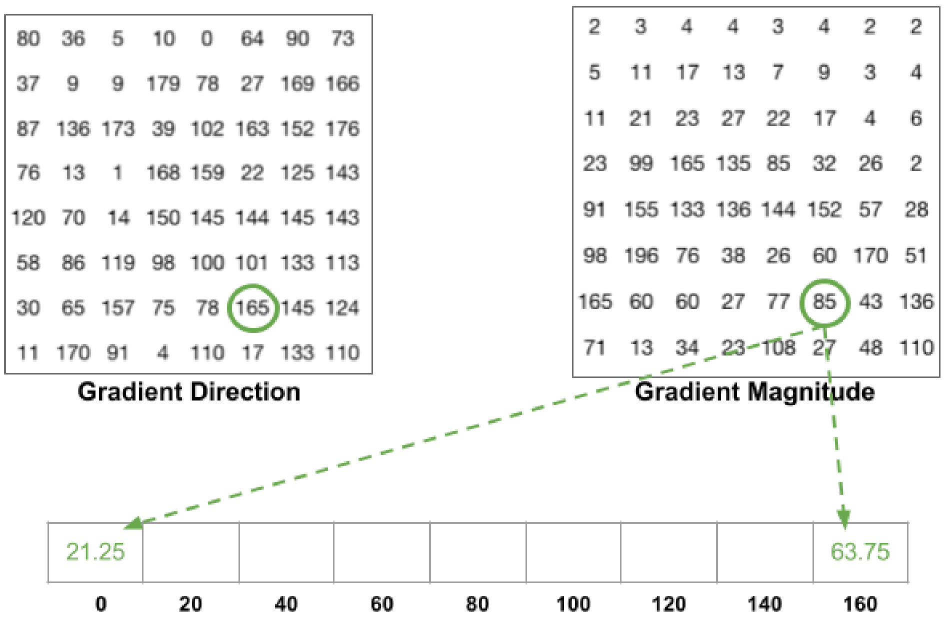
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Figure 2.12: Gradient magnitude and direction.[22]

- Then, we decide where each pixel goes inside of the histogram. A bin is selected depending on the direction chosen, and the value that is placed inside of the bin is dependent on the magnitude. If a pixel is halfway between two bins, then it splits up the magnitudes accordingly depending on their distance away from each respective bin. After that, a histogram can be formed, and the bins that have the most weight can easily be seen.



Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

0	20	40	60	80	100	120	140	160
21.25								63.75

Figure 2.13: Histogram of Gradients.[22]

2.7 conclusion

To allow the computer to identify and perceive objects in a manner similar to the human being, several approaches have been proposed by researchers, some are know as traditional methods, where a filter must be specified for each specific object. The other methods are based on neural networks, these methods are more efficient. The information obtained is used to solve computer vision problems such as object detection and localization.

Chapter 3

Deep learning

3.1 Introduction

As you know, during recent years all eyes are on deep learning. A lot of researchers think that it's the most interesting field in computer science, because it solved many complex problem such as natural language processing,image recognition,etc. In this chapter we will define what is a neural network, and introduce different concepts allowing us to train it, after that we will explore CNN which is mostly used to solve computer vision tasks, and take a look at some of its architecture.

3.2 An artificial neuron

An artificial neuron is represented by a mathematical function based on a model of a biological neuron, they are presented as follow :

3.2.1 Structure

Each artificial neuron is an elementary processor. It receives a variable number of inputs from neighbouring neurons, each of these inputs is associated with a weight w which represents the strength of the connection. Each elementary processor has a single output, which then connects to multiple number of neurons[19].

3.2.2 behaviour

First, the neuron calculates the weighted sum of inputs (Z) according to the following expression : $z = \sum (w_i \cdot x_1)$. Then, this value is passed through an activation function to the network[19].

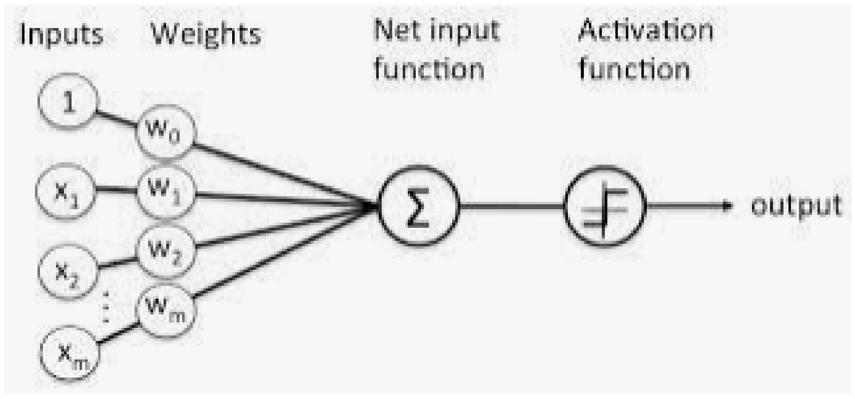


Figure 3.1: Structure of an artificial neuron.[13]

3.3 Activation function

Activation functions take the value of the sum. It determines whether or not a node ought to be activated .The activation function used might improve or scale back the performance of the artificial neuron. Their main task is to remodel the signal within the node into an output signal.Here are some of the most popular types of activation functions :

3.3.1 The Sigmoid function

Historically, the sigmoid function is the oldest and most popular activating function. It is defined as follows : $\sigma(x) = \frac{1}{1+e^{-x}}$ We first set the variable (z) to the weighted sum of the inputs, then pass it to the sigmoid function : $z = b + \sum_i w_i x_i$, $\sigma(z) = \frac{1}{1+e^{-z}}$.

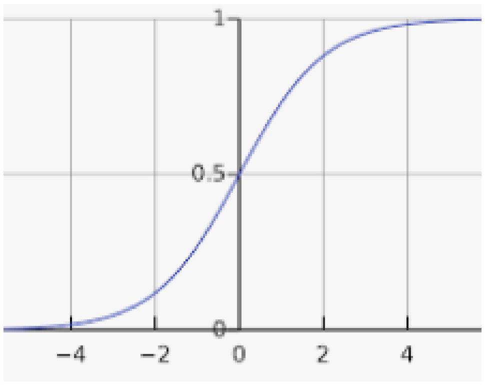


Figure 3.2: Sigmoid function

3.3.2 Tanh (Tangent Hyperbolic function) function

The tanh function is similar to the sigmoid. The difference is that it is symmetric around the origin. The range of values is from -1 to 1. therefore the inputs to the next layers won't always be of the same sign. The tanh function is defined as : $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

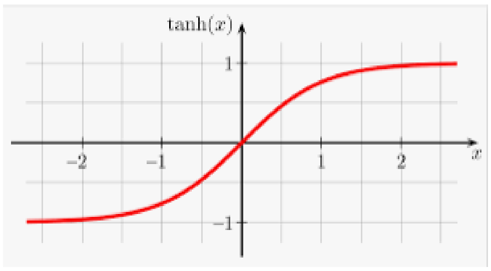


Figure 3.3: TanH function

3.3.3 Rectified Linear Unit ReLU

The Rectified Linear Unit or just ReLU has become popular in the last few years. The activation is simply thresholded at zero: $R(x) = \begin{cases} x & if x \geq 0 \\ 0 & otherwise \end{cases}$

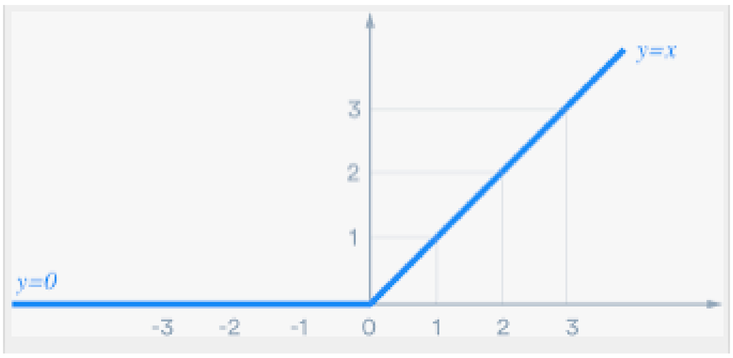


Figure 3.4: ReLu function

3.4 Artificial neural network

Artificial neural network is an interconnected group of neurons. A row of neurons is named a layer and one network can have multiple layers.The input layer reads incoming signals, one neuron per input x_j, an output layer provides the system's response. One or more hidden layers that contribute to the transfer.Each neuron of the hidden layer is connected to neurons of the previous next layer[7].

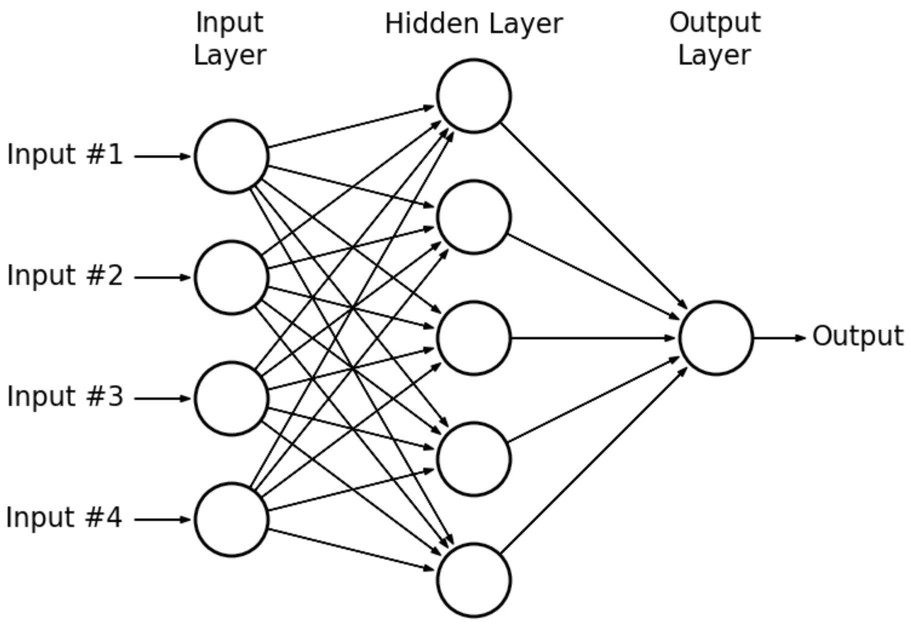


Figure 3.5: Schematic representation of a neural network.

3.5 Loss function

A loss function is a function that evaluates the difference between the predictions made by the neural network and the actual values of the observations used during training. So,if the predictions are totally off ,the loss function will output a higher number. The minimization (reducing to a minimum the difference between the predicted value and the real value) is done by adjusting the different weights of the neural network. 'Loss' helps us to understand how much the predicted value differ from actual value.

- Cross Entropy :

  Cross-entropy loss, measures the performance of a classification model whose output is a probability value between zero and one. Cross-entropy loss will if the predicted probability diverges from the actual label.

  $I(\Theta) = -\frac{1}{m}\Sigma_{i=1}^{m} \Sigma_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$.[6]

  $y_k^{(i)}$: is the target probability that the $y_k^{(i)}$ belongs to class k.

- L2 loss : L2 Loss Function is used to minimize the error which is the sum of all squared differences between the true value and the predicted value.

  $\text{L2Loss} = \Sigma_{i=1}^{n} \left(y_{true} - y_{predicted}\right)^2$

3.6 Gradient Descent

Gradient descent is an optimization algorithm used to find the minimum of the cost function. The general formula is as follows: $x_{t+1} = x_t - \eta \times \nabla f\left(x_t\right)$, where η the learning rate and $\nabla f\,(x_t)$ the direction of descent. This class of algorithm aims to have $f(x_{t+1}) \leq f(x_t)$ at each iteration, with f a convex function [see Figure 3.6] that we want to minimize.

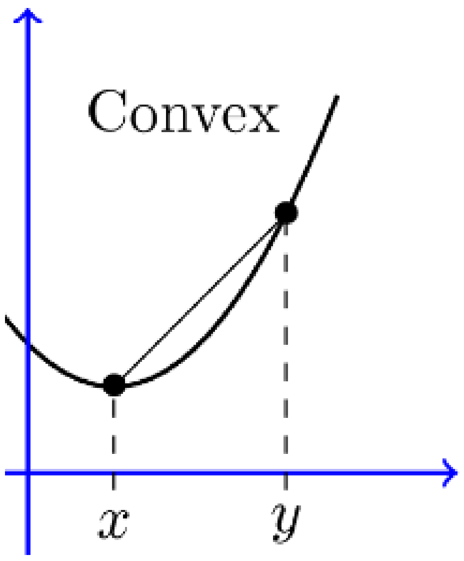


Figure 3.6: a convex function.

The learning rate η determines how big the steps into the direction of the local minimum are taken by the descent gradient. So, to reach the minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. If the steps are too big, it may not reach the local minimum because it bounces between the convex function of gradient descent [see Figure 3.7]. If we set the learning rate to a small value, gradient descent will reach the local minimum but that will take a while [see Figure 3.8].

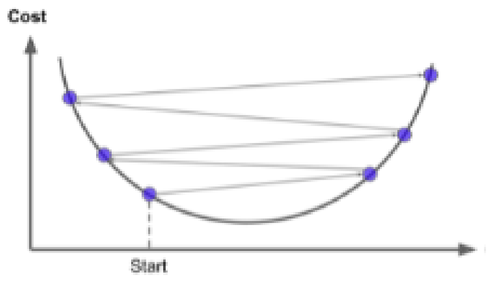


Figure 3.7: Large learning rate.[21]

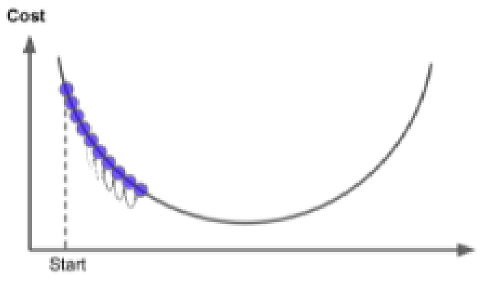


Figure 3.8: A small learning rate.[21]

3.7 Back propagation algorithm

Back propagation is the algorithm that made training MLP's (Multilayer perceptron) successful after a long research[6], and it is still used nowadays. It is very efficient approach to compute gradients of a complex cost function regardless of the number of parameters of the network, and That's how it works :

- The algorithm takes one mini batch at a time from the training set and it passes it through the network.When all the training instances are passed that means an epoch is done (the training process is a set of epochs).
- The mini batch goes to the input layers first and since this layer does not have weights, it has no effect on the input features at all, it just sends the features X_n to the next layer.
- Next, the algorithm computes for each neuron of the current hidden layer the dot product $Z^{(L)}$ of the output of the previous layer $A^{(L-1)}$ (X_i if the previous layer is the input layer) and the weight of the current layer $W^{(L)}$, and then an activation function $\phi(z)$ is applied to the result. Finally the final result $A^{(L)}$ (the output of the current layer) will pass to the next layer, and so on until it reaches the output layer, which its outputs $A^{(out)}(\hat{y})$ is called the prediction of the network, this step is called the **forward pass**.[12]

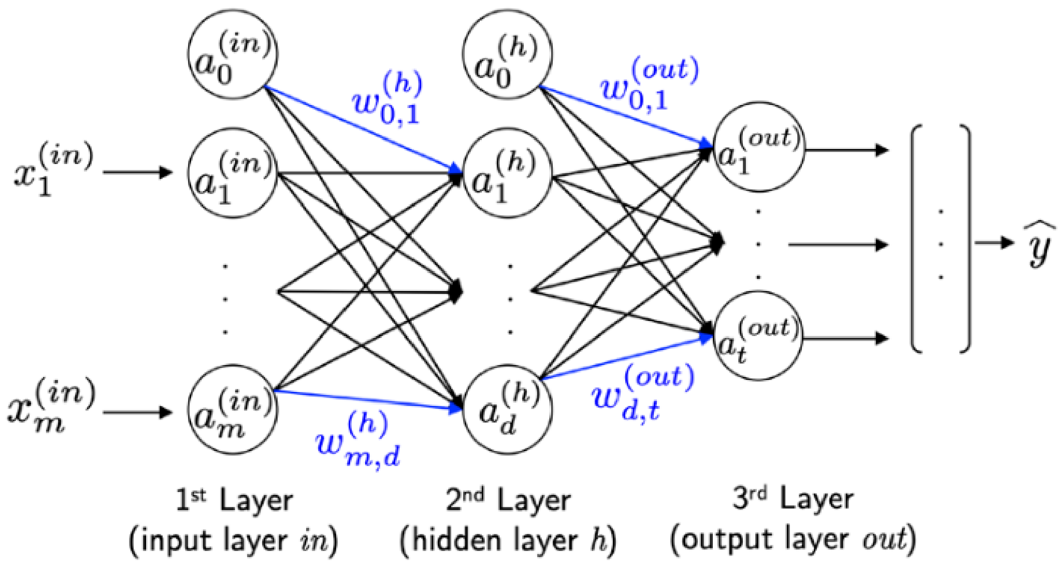


Figure 3.9: Forward pass.[12]

- After the algorithm makes the forward pass to get the prediction, it uses a lot of function that compares the predictions ($\hat{y}$) and the true prediction (y) and it gets the error of the output layer $\delta^{(out)}$.
- Then, the error $\delta^{(out)}$ of the output layer is used to measure how much the output of the previous layer and the weights of the output layer contributed to this error, and so on it keeps propagate the error in the opposite direction.

- Finally, a gradient descent step is applied to update the weights, this step is called backward pass.

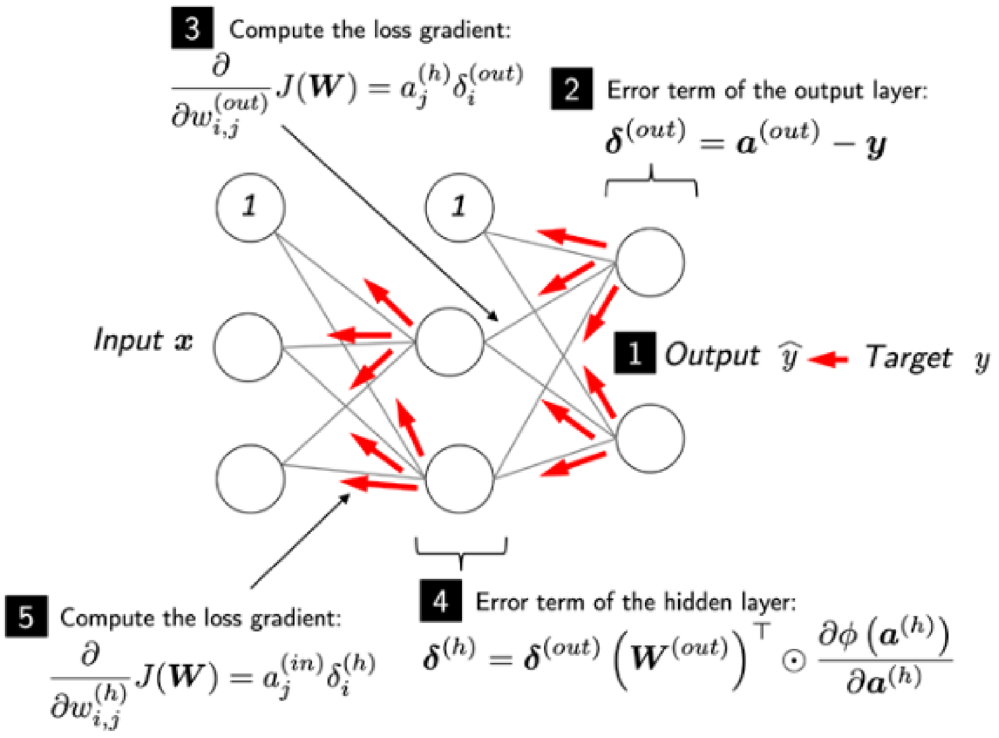

3 Compute the loss gradient:
$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\mathbf{W}) = a_j^{(h)} \delta_i^{(out)}$
2 Error term of the output layer:
$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boldsymbol{y}$
1
1
Input x
1 Output $\hat{y}$ Target y
5 Compute the loss gradient:
$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\mathbf{W}) = a_j^{(in)} \delta_i^{(h)}$
4 Error term of the hidden layer:
$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)} \left(\mathbf{W}^{(out)}\right)^{\top} \odot \frac{\partial \phi\left(\boldsymbol{a}^{(h)}\right)}{\partial \boldsymbol{a}^{(h)}}$

Figure 3.10: Backward pass.[12]

3.8 Regularization

"**Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Regularization is one of the central concerns of the field of machine learning, rivaled in its importance only by optimization".[7]
In this section we will see three techniques to deal with overfitting : Early stopping, L2&L1 and dropout.

3.8.1 Early stopping

During the training, the algorithm measures the performance of the model on both the training data and the validation set after each epoch, to check how the model is doing. When the prediction error of the model on the training set and and the validation data is going down that indicates that the model is doing well, but the training error keeps decreasing and the validation error start to go up, this means that the model begin to overfit the training data (overfitting problem). So, one of the simplest techniques used to avoid this issue is to stop the training when the model's prediction error of the validation set starts to go up, and only keeps the model having the smallest prediction error on the validation, this technique is called **early stopping**.[6]

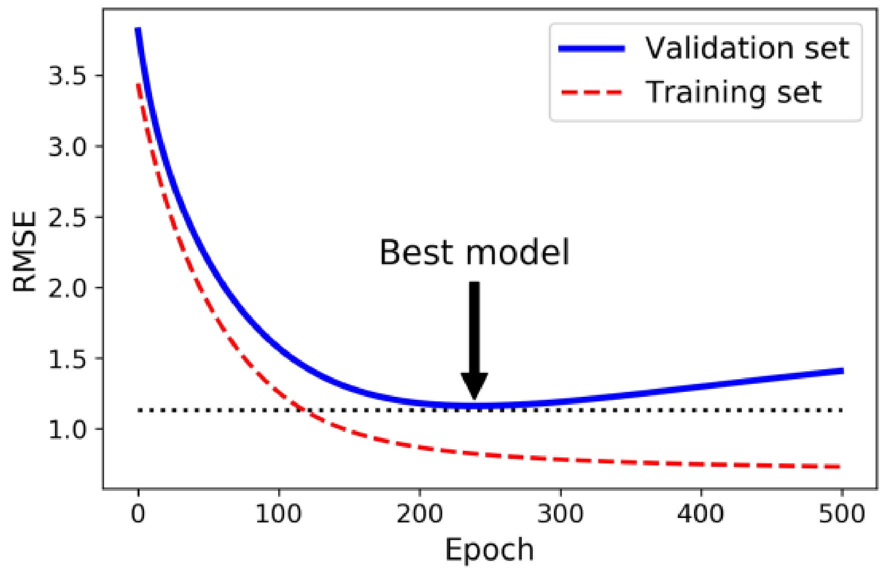


Figure 3.11: Early stopping.[6]

3.8.2 L2&L1

Another approach to deal with the overfitting problem is to use the **l2** regularization, which consist in adding a regular term to the cost function , for instance in simple learning algorithm the term $\alpha \sum_{i=1}^{n} w_i^2$ is added to the cost function and that reduces the complexity of the model by keeping its weights as small as possible. **l1** regularization is used to reduce the complexity of the model too, which consists in adding the term $\alpha \sum_{i=1}^{n} |\omega_i|$ to the cost function, but this technique tends to remove the less impotent features by setting their weights to zero. Similarly, we can add l1 and/ or l2 regularization to constrain the neural network weights[6].

3.8.3 Dropout

Dropout is a very powerful technique for regularizing neural networks, and it is highly successful. It has been proposed in paper [5] . The way that it works is simple : during the training a subset of neurons is randomly dropped at every training step with probability Pdrop and this means that these neurons will output zero, if a neuron is inactive in one step it may be active in the next step. The neuron is dropped just in training phase, when in the test phase all the neurons have to be active. This probability is called drop rate and it is determined by the user.[6]

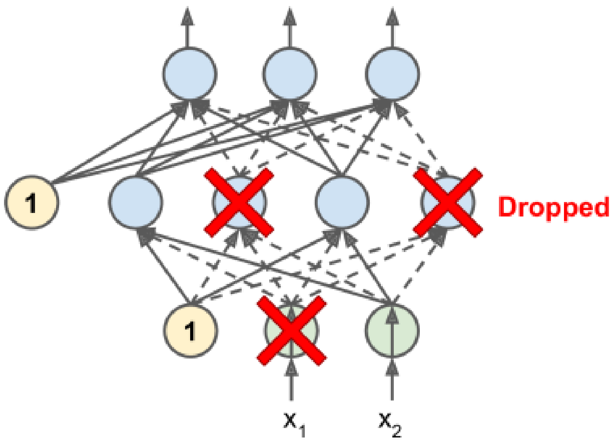


Figure 3.12: Dropout.[6]

3.9 Convolutional Neural Network CNN

CNN was inspired by how the brain human's visual cortex works, it gained a lot attention and made a success in computing vision tasks. It can handle multi dimensional layer, each layer is represented in 2D array or more, so no need to flatten the input images in 1D array before feeding them to a network.CNN can learn spatial hierarchies of patterns.The first layer of CNN learn low level of features such as lines and edges.[3] The second layer learn large features made of the features of the first layer and so on. The upper layers learn complex features [see Figure 3.13].

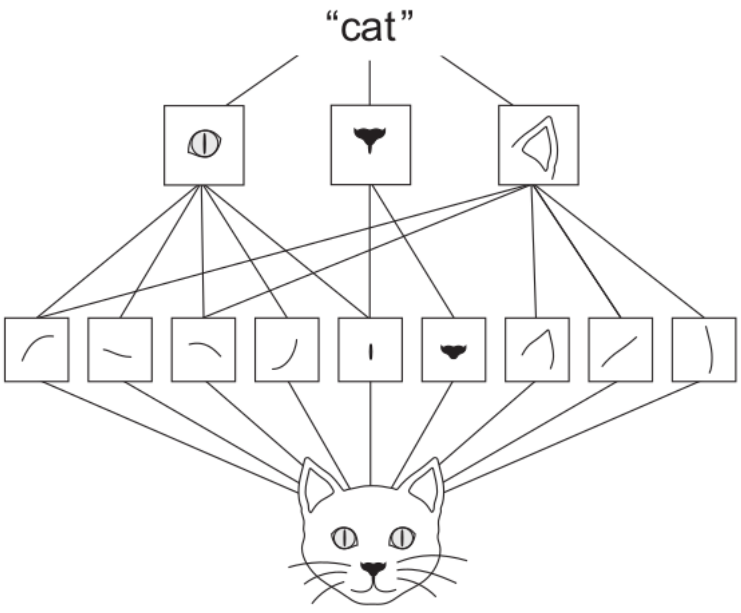


Figure 3.13: Features extraction.[3]

So, CNN is a feature extractor, and often the last layer is fully connected layer that uses these features to predict a class label. Let's see the other category of layers that exist in CNN.

3.9.1 convolutional layer

This is the most important category of layer in CNN, and because of it CNN gets its name. instead of matrix multiplication. This layer apply a convolution operation to its inputs. For simplicity a convolution operation in CNN can be understood a sliding matrix w which called Kernel (filter) over an input over an input matrix and at each position the element-wise-product of the rotated filter w^r and the patch of x. The hyper-parameter that controls how many cells the filter w is shifted from one position to another is called stride [see Figure 3.14].

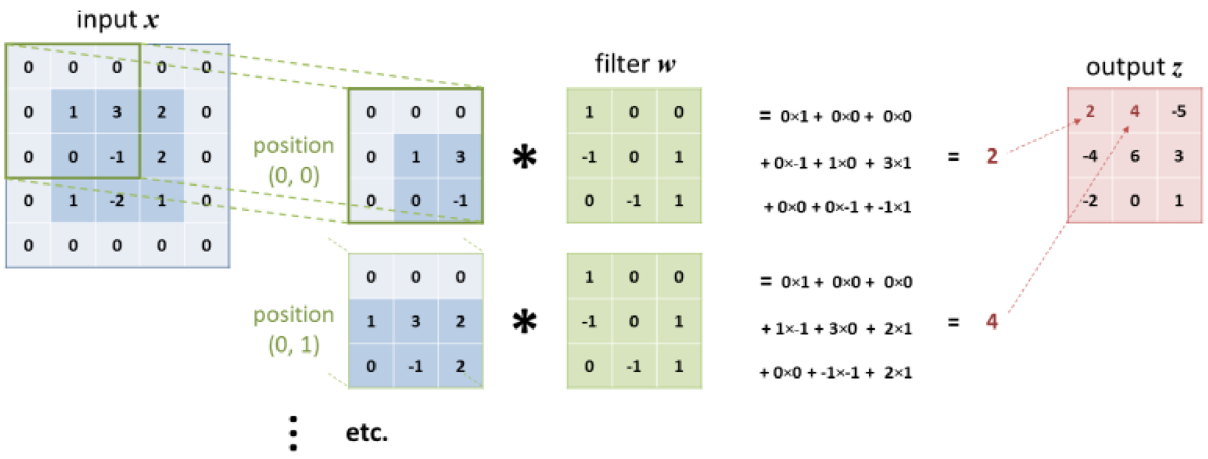


Figure 3.14: Convolutional layer.[11]

In the figure above [see Figure 3.15] we saw that the input matrix x was padded with zero, it is another hyper-parameter called **padding**, its goal is just to control the size of the output of the convolutional layer. In contrast of fully connected layer each is connected with a few neurons located in a small window (receptive field) in the previous layer. since the neurons of a single layer shares the same weights and the bias term the neuron's weights called the filter of the layer (Kernel) w. So in order to compute the outputs of a convolutional layer, a convolutional operation of the filter w and the input is performed first, and then an activation function is applied to the results. The final result is called a feature map, the following formula represent the output of a neuron :
$z_{i,j} = \sigma \left(b + \sum_{l=0}^{k_H-1} \sum_{m=0}^{k_W-1} \sum_{n=0}^{D-1} w_{l,m,n} \cdot x_{i+l,j+m,n} \right)$.[11]
Since the convolutional layer can have multiple filter, so it outputs one feature map per filter.

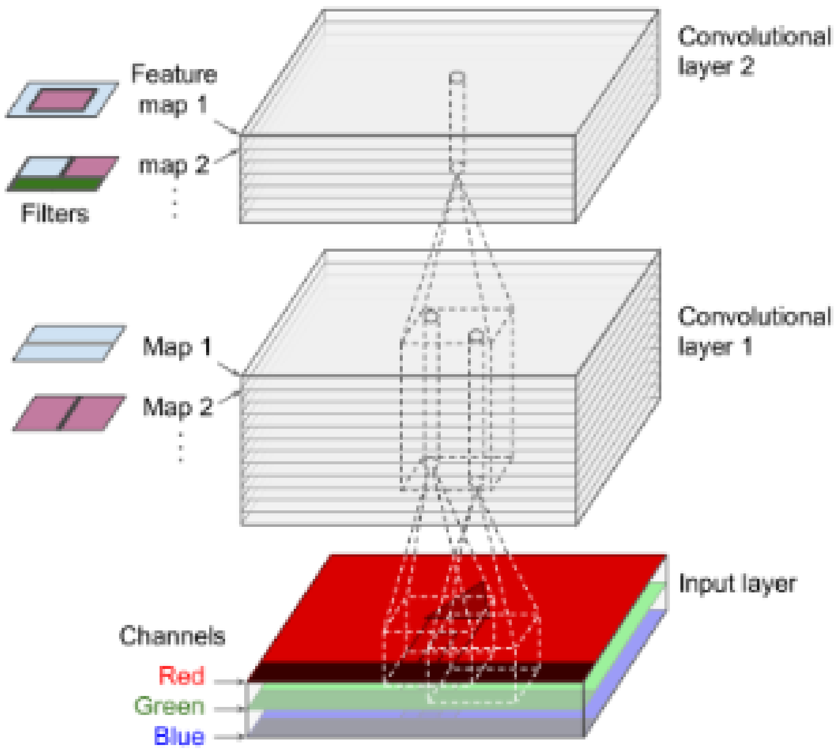


Figure 3.15: Convolutional layers output.[6]

3.9.2 Pooling layer

Another category of layers can be found in the CNN is the pooling layer. Just like the convolutional layer each neuron of the pooling layer is connected only to a few neurons of the previous layer, located within a small window(reception field). The pooling layer has no trainable parameters.Indeed, each neuron apply aggregation function to its inputs (window) such as max or mean, which return a single output, it is common to set the stride equal to the size of the window (receptive field). The layer that uses a max function is called Max pooling layer (the same with Mean-pooling layer also known Average pooling layer) [see Figure 3.16].

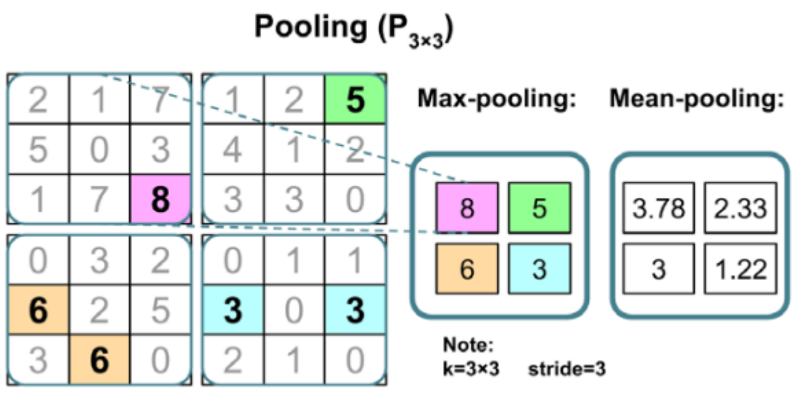


Figure 3.16: Max and Mean pooling.[12]

The pooling layer is used to shrink the input images and that for decreasing the number of parameters, reducing the number usage and accelerating the training a little bit [see

Figure 3.17].

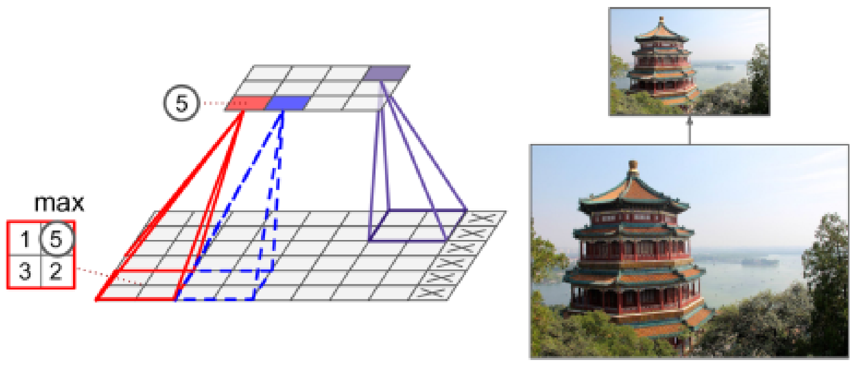


Figure 3.17: Max pooling layer.[6]

3.9.3 CNN's architectures

This is one of the best architectures today, it achieved second place in the **ILSVRC** (ImageNet Large Scale Visual Recognition Challenge) class affection task, it was developed by visual geometry group from Oxford university. Its architecture is very simple, it is composed of five blocks, each block has two or three convolutional layers followed by a pooling layer, and at the end of the architecture there are three dense layers(with dropout).[18]. In other words there are 16 or 19 trainable layers depending VGG variant (VGG16 or VGG19) [see Figure 3.18].

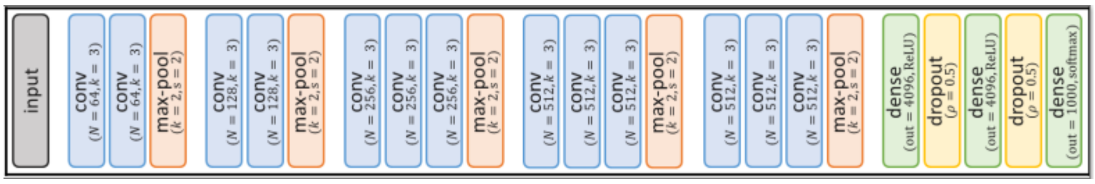


Figure 3.18: VGG layers.[11]

3.9.4 LeNet-5

This is the most known architecture, it was developed by Yan Lecun in 1998, and it is widely used for hand writing recognition.[see Figure 3.19].

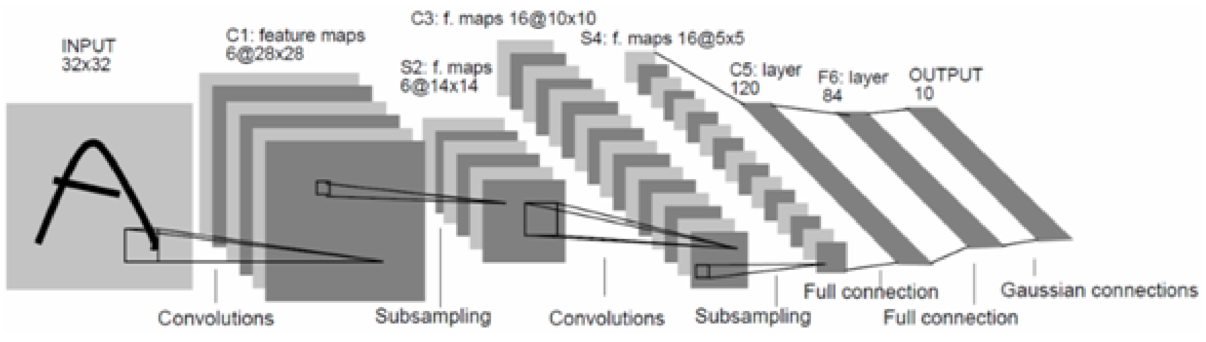
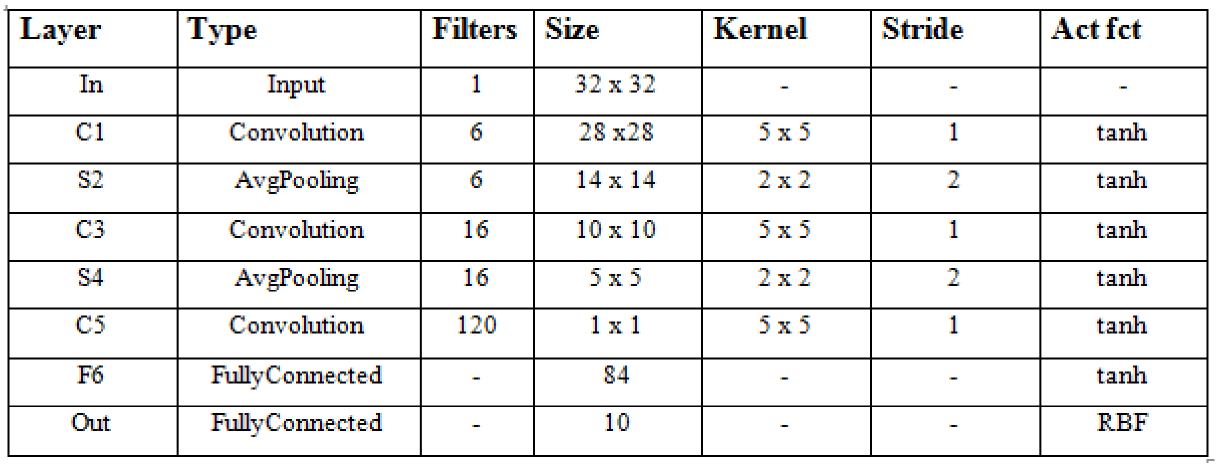


Figure 3.19: LeNet-5architecture.[10]



Layer	Type	Filters	Size	Kernel	Stride	Act fct
In	Input	1	32 x 32	-	-	-
C1	Convolution	6	28 x28	5 x 5	1	tanh
S2	AvgPooling	6	14 x 14	2 x 2	2	tanh
C3	Convolution	16	10 x 10	5 x 5	1	tanh
S4	AvgPooling	16	5 x 5	2 x 2	2	tanh
C5	Convolution	120	1 x 1	5 x 5	1	tanh
F6	FullyConnected	-	84	-	-	tanh
Out	FullyConnected	-	10	-	-	RBF

Figure 3.20: LeNet-5 structure.

3.10 Conclusion

Deep learning has progressed a lot over the past few years and solved many complex problems, especially CNN that made big success in computer vision so we can benefit from using deep learning to solve our localization problem, and this is precisely what we will attempt to accomplish in the next chapter.

Chapter 4

Contribution

4.1 Introduction

In this chapter we will talk about the practical requirements to tackle the localization problem, we will start by presenting the dataset, the development tools needed to accomplish our project. Next ,we will talk about how to build and designed a neural network, and apply the evaluation metrics and loss functions. Finally, we will see the development process that we followed to handle the localization task, and discuss the results.

4.2 Data Set

The hard part of any machine learning project is getting the label of the dataset (without need to mention how difficult is to collect the data), fortunately there are a lot of open datasets to choose from in order to experiment different machine learning tasks such as regression, classification, localisation, and detection.
In our project **"object localisation"** we experiment with the **Oxford-IIIT Pet Dataset** created by Visual Geometry Group in Oxford university. it is a dataset that contains 3685 images with their annotations, each image represent a category of either cat or dog and its annotation is an XML file that contain information of the image such as the size, the class label, the name of the category and 4 real numbers which represent the box around the face of the object.

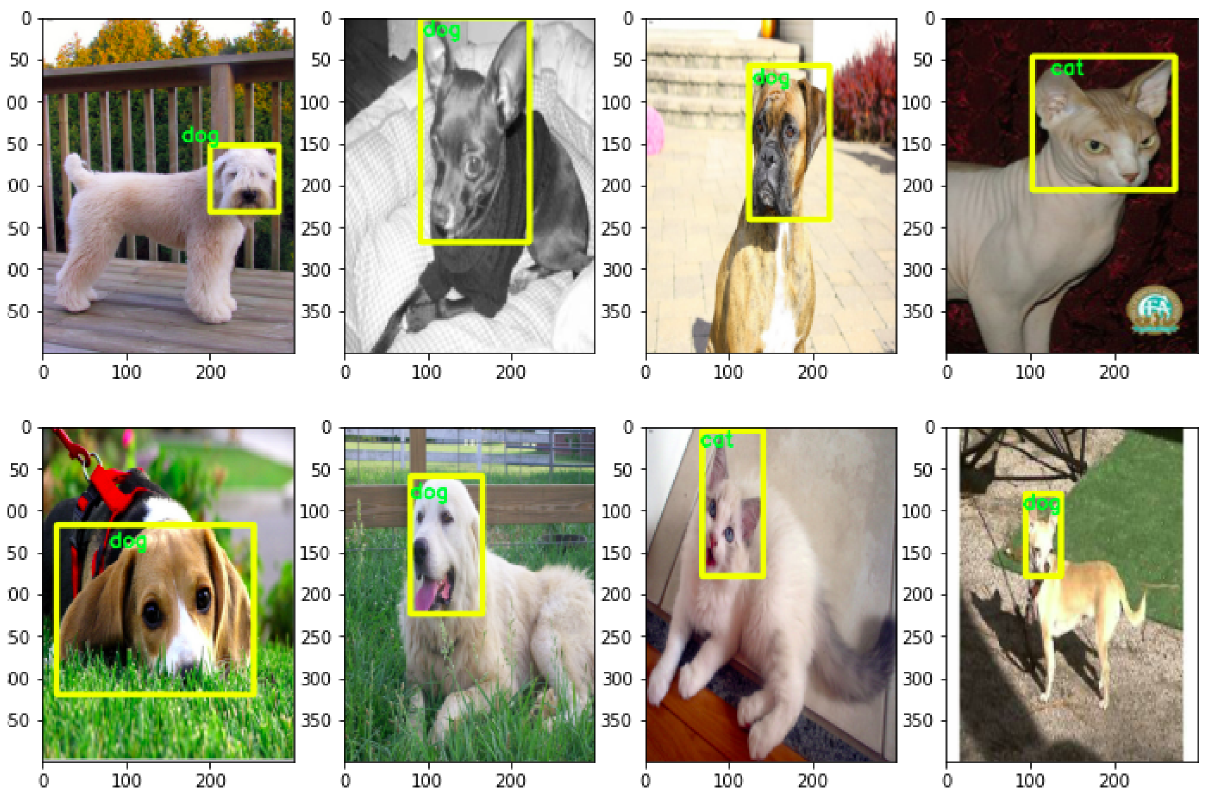


Figure 4.1: Sample from the Oxford-IIIT Pet Datase.

4.3 Development tools

The tools used to achieve this project are mentioned bellow :

4.3.1 Software tools

- **Colaboratory** : often shortened to "Colab", is a product of Google Research. Colab allows writing and running the Python code through the browser. It is an environment particularly suited to machine learning and data analysis.
- **TensorFlow** : is an end-to-end open source machine learning platform. It offers a complete and flexible ecosystem of tools, libraries and community resources that allow researchers to advance in the field of machine learning, and developers to easily build and deploy applications that exploit this technology.
- **Keras** : is an open source library written in Python, it enables interaction with deep neural network and machine learning algorithms.It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit,etc.
- **Numpy** : is a python library, intended to handle multidimensional matrices or arrays and also mathematical functions.
- **Matplotlib** : is python library, it is used for plotting and visualizing data as a graph. It can be combined with the NumPy and SciPy scientific computing python libraries.

4.3.2 Hardware tools

- **CPU** : Intel(R) Xeon(R) CPU @ 2.30GHz.
- **GPU** : NVIDIA Tesla P100.
- **RAM** : 13 GB.
- **Disk space** : 69 GB.
- **OS**

4.4 Building the model

As we mentioned before, localizing an object in an image is trying to draw a bounding box around it. the bounding box is define by 4 real numbers that represents the coordinates of the upper-left point and bottom-right point of this bounding box. So first we have a regression task which consists on predicting the coordinate of the bounding box, the second is a classification task which consist on figuring out the category of the object (in our case cat or dog).

4.4.1 Model selection

To solve this problem we can use two architecture of neural network one for the classification task, and the other for the regression task or we can train only one neural network with two output layer for both tasks at the same time which works usually works better,[16] for example in our dataset (**Oxford-IIIT Pet Dataset**) we used the second architecture with the two output layers, one has 4 neurons (the coordinate of the bounding box) and no activation function and the other contains 2 neurons (cat or dog) and the softmax activation function, we already talked about CNN and how successful it is in computer vision tasks, and works well as a feature extractor, so we can design an architecture for our problem by putting convolutional layers and pooling layers together and connect them to the two output layer (we can add fully connected layers before each output layer), but it is not a good idea to train a deep neural network from scratch because it is hard to set the hyper-parameters and it takes a lot of time. Instead we can try to find an existent neural network that was already trained on a similar or global task so we can re-use its lower layers and adapting them to our task, this technique is called **transfer learning** [see Figure 4.2].[6]

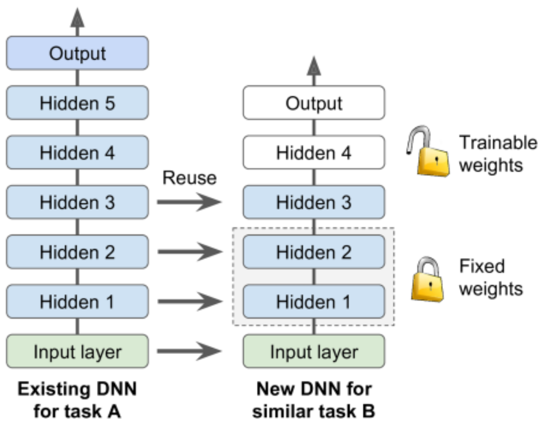
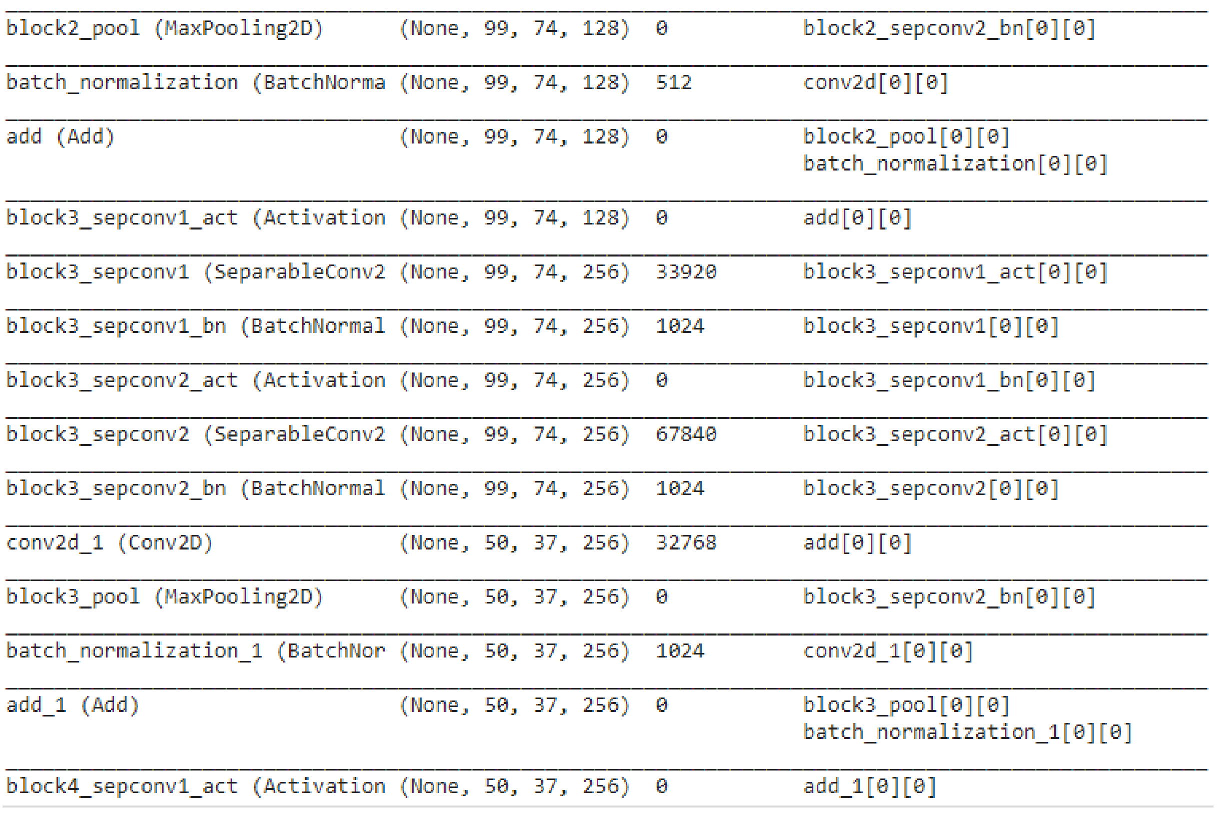


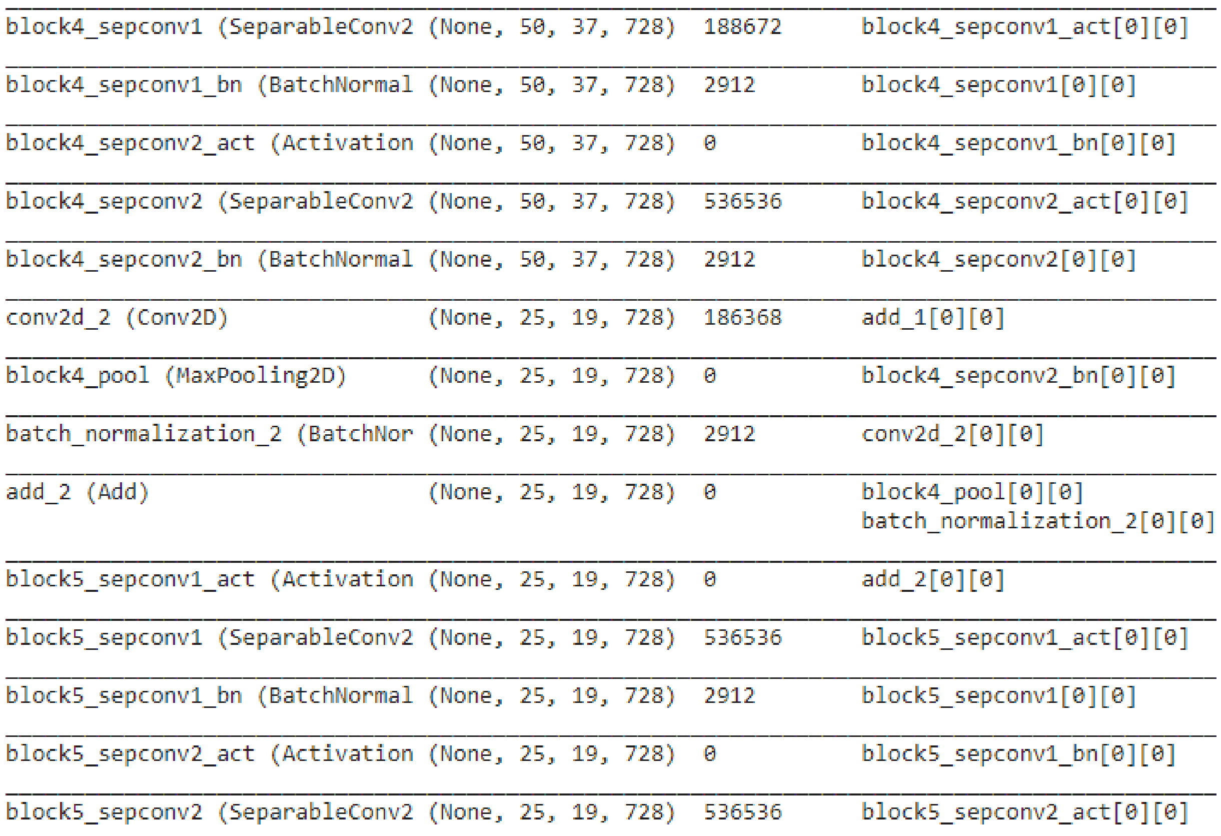
Figure 4.2: Transfer learning.[6]

For instance in our architecture problem we re-used **Xception** (one of the state of the art architectures that has been trained on **ImageNet**[2],The pretrained network can classify images into 1000 object categories, such as pencil and many animals,etc. As a result,so the network has learned rich feature representations for a wide range of images.) and at the top of it we added a few dense layers in addition of the output layers. Here is the architecture of our final model :

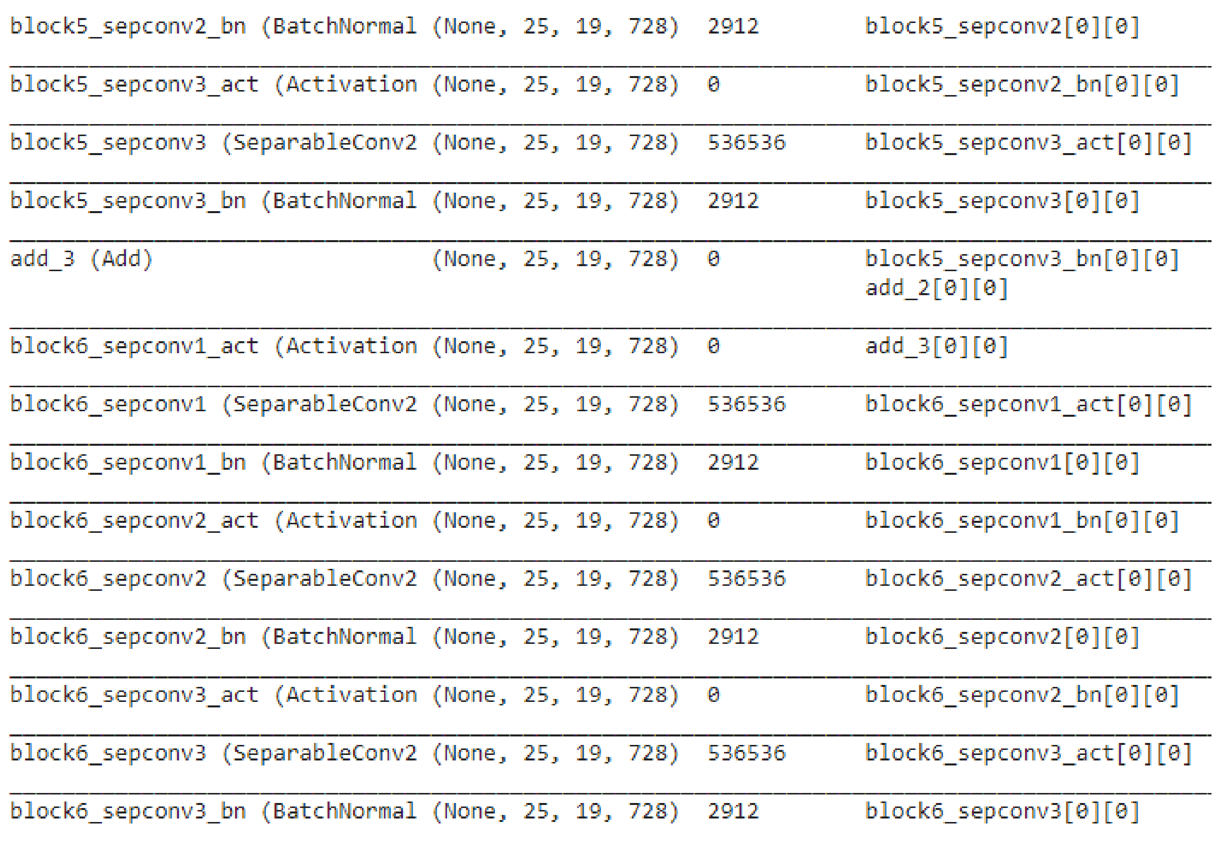
```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 400, 300, 3) 0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 199, 149, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
block1_conv1_bn (BatchNormaliza (None, 199, 149, 32) 128         block1_conv1[0][0]
__________________________________________________________________________________________________
block1_conv1_act (Activation)   (None, 199, 149, 32) 0           block1_conv1_bn[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 197, 147, 64) 18432       block1_conv1_act[0][0]
__________________________________________________________________________________________________
block1_conv2_bn (BatchNormaliza (None, 197, 147, 64) 256         block1_conv2[0][0]
__________________________________________________________________________________________________
block1_conv2_act (Activation)   (None, 197, 147, 64) 0           block1_conv2_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv1 (SeparableConv2 (None, 197, 147, 128 8768        block1_conv2_act[0][0]
__________________________________________________________________________________________________
block2_sepconv1_bn (BatchNormal (None, 197, 147, 128 512         block2_sepconv1[0][0]
__________________________________________________________________________________________________
block2_sepconv2_act (Activation (None, 197, 147, 128 0           block2_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv2 (SeparableConv2 (None, 197, 147, 128 17536       block2_sepconv2_act[0][0]
__________________________________________________________________________________________________
block2_sepconv2_bn (BatchNormal (None, 197, 147, 128 512         block2_sepconv2[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 99, 74, 128)  8192        block1_conv2_act[0][0]
```



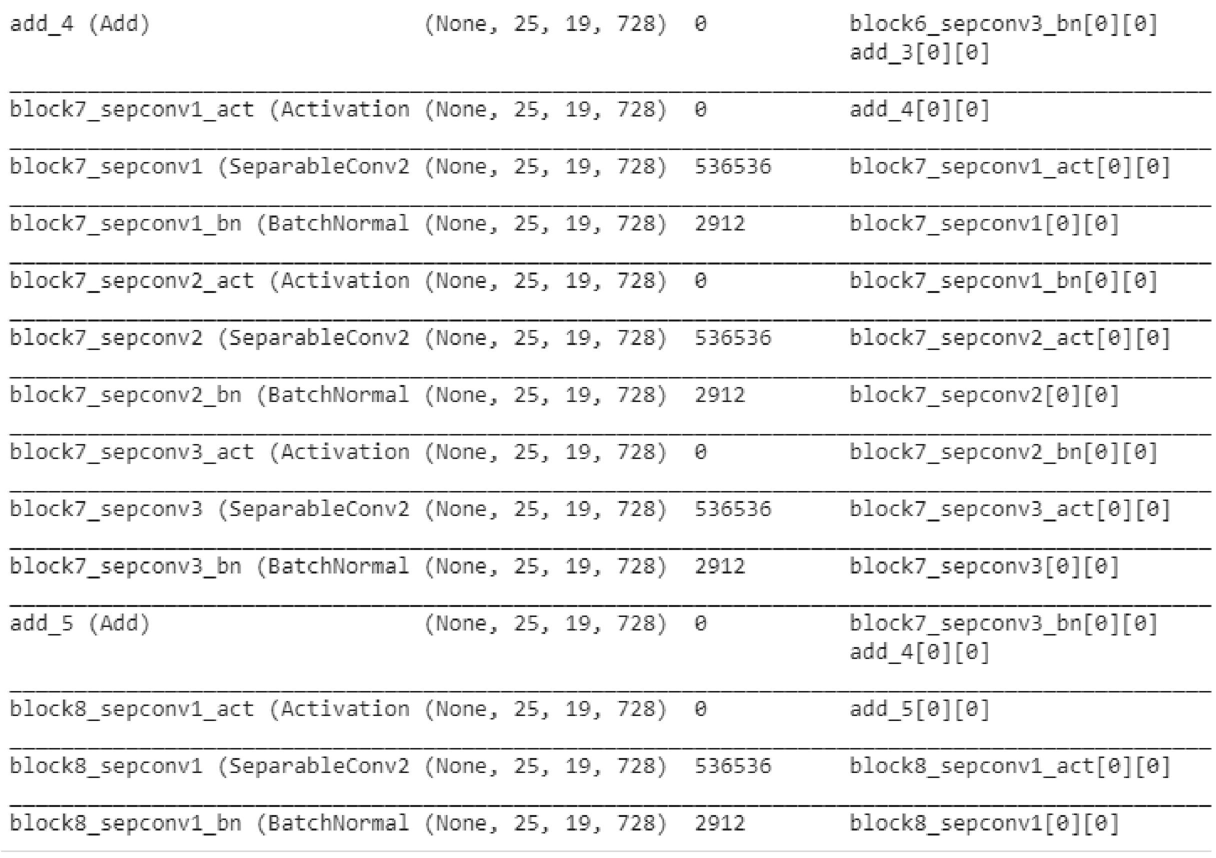
```
______________________________________________________________________________________________
block2_pool (MaxPooling2D)       (None, 99, 74, 128)  0           block2_sepconv2_bn[0][0]
______________________________________________________________________________________________
batch_normalization (BatchNorma (None, 99, 74, 128)  512         conv2d[0][0]
______________________________________________________________________________________________
add (Add)                        (None, 99, 74, 128)  0           block2_pool[0][0]
                                                                  batch_normalization[0][0]
______________________________________________________________________________________________
block3_sepconv1_act (Activation (None, 99, 74, 128)  0           add[0][0]
______________________________________________________________________________________________
block3_sepconv1 (SeparableConv2 (None, 99, 74, 256)  33920       block3_sepconv1_act[0][0]
______________________________________________________________________________________________
block3_sepconv1_bn (BatchNormal (None, 99, 74, 256)  1024        block3_sepconv1[0][0]
______________________________________________________________________________________________
block3_sepconv2_act (Activation (None, 99, 74, 256)  0           block3_sepconv1_bn[0][0]
______________________________________________________________________________________________
block3_sepconv2 (SeparableConv2 (None, 99, 74, 256)  67840       block3_sepconv2_act[0][0]
______________________________________________________________________________________________
block3_sepconv2_bn (BatchNormal (None, 99, 74, 256)  1024        block3_sepconv2[0][0]
______________________________________________________________________________________________
conv2d_1 (Conv2D)                (None, 50, 37, 256)  32768       add[0][0]
______________________________________________________________________________________________
block3_pool (MaxPooling2D)       (None, 50, 37, 256)  0           block3_sepconv2_bn[0][0]
______________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 50, 37, 256)  1024        conv2d_1[0][0]
______________________________________________________________________________________________
add_1 (Add)                      (None, 50, 37, 256)  0           block3_pool[0][0]
                                                                  batch_normalization_1[0][0]
______________________________________________________________________________________________
block4_sepconv1_act (Activation (None, 50, 37, 256)  0           add_1[0][0]
```



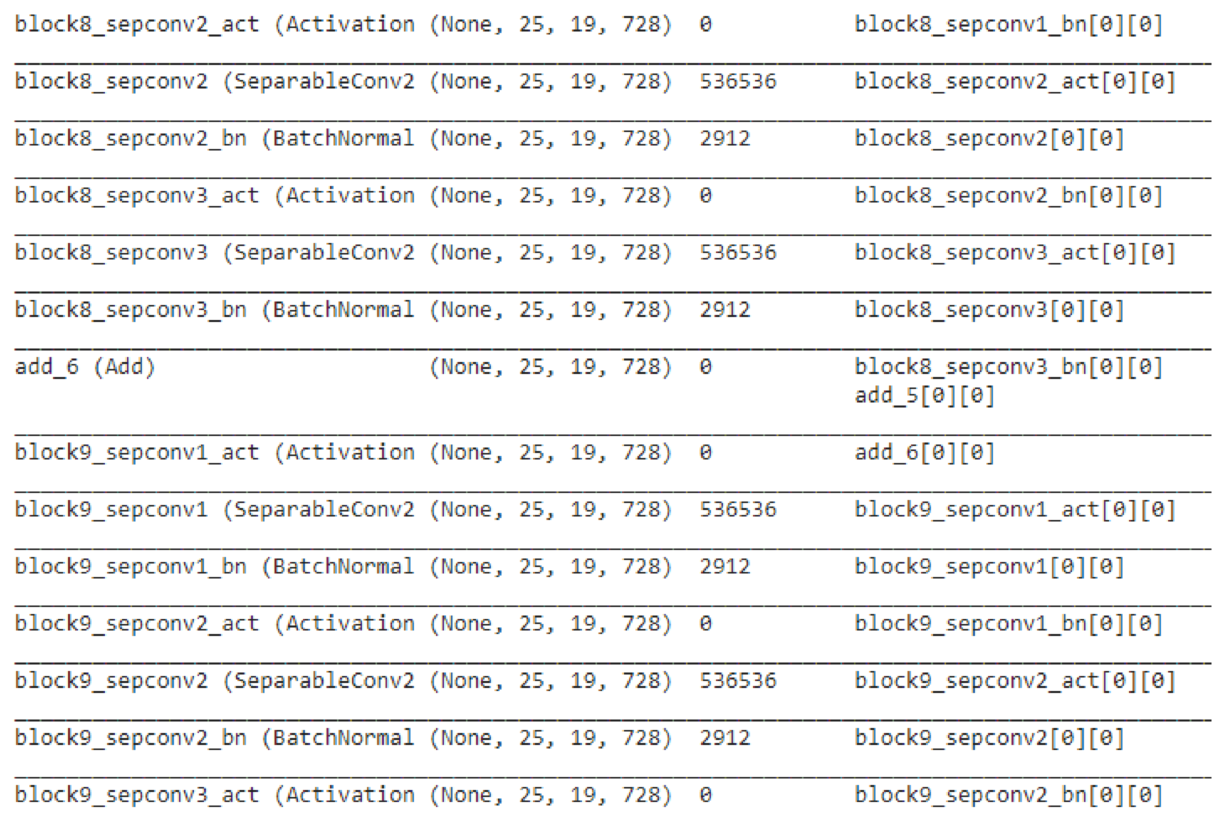
```
______________________________________________________________________________________________
block4_sepconv1 (SeparableConv2 (None, 50, 37, 728)  188672      block4_sepconv1_act[0][0]
______________________________________________________________________________________________
block4_sepconv1_bn (BatchNormal (None, 50, 37, 728)  2912        block4_sepconv1[0][0]
______________________________________________________________________________________________
block4_sepconv2_act (Activation (None, 50, 37, 728)  0           block4_sepconv1_bn[0][0]
______________________________________________________________________________________________
block4_sepconv2 (SeparableConv2 (None, 50, 37, 728)  536536      block4_sepconv2_act[0][0]
______________________________________________________________________________________________
block4_sepconv2_bn (BatchNormal (None, 50, 37, 728)  2912        block4_sepconv2[0][0]
______________________________________________________________________________________________
conv2d_2 (Conv2D)                (None, 25, 19, 728)  186368      add_1[0][0]
______________________________________________________________________________________________
block4_pool (MaxPooling2D)       (None, 25, 19, 728)  0           block4_sepconv2_bn[0][0]
______________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 25, 19, 728)  2912        conv2d_2[0][0]
______________________________________________________________________________________________
add_2 (Add)                      (None, 25, 19, 728)  0           block4_pool[0][0]
                                                                  batch_normalization_2[0][0]
______________________________________________________________________________________________
block5_sepconv1_act (Activation (None, 25, 19, 728)  0           add_2[0][0]
______________________________________________________________________________________________
block5_sepconv1 (SeparableConv2 (None, 25, 19, 728)  536536      block5_sepconv1_act[0][0]
______________________________________________________________________________________________
block5_sepconv1_bn (BatchNormal (None, 25, 19, 728)  2912        block5_sepconv1[0][0]
______________________________________________________________________________________________
block5_sepconv2_act (Activation (None, 25, 19, 728)  0           block5_sepconv1_bn[0][0]
______________________________________________________________________________________________
block5_sepconv2 (SeparableConv2 (None, 25, 19, 728)  536536      block5_sepconv2_act[0][0]
```



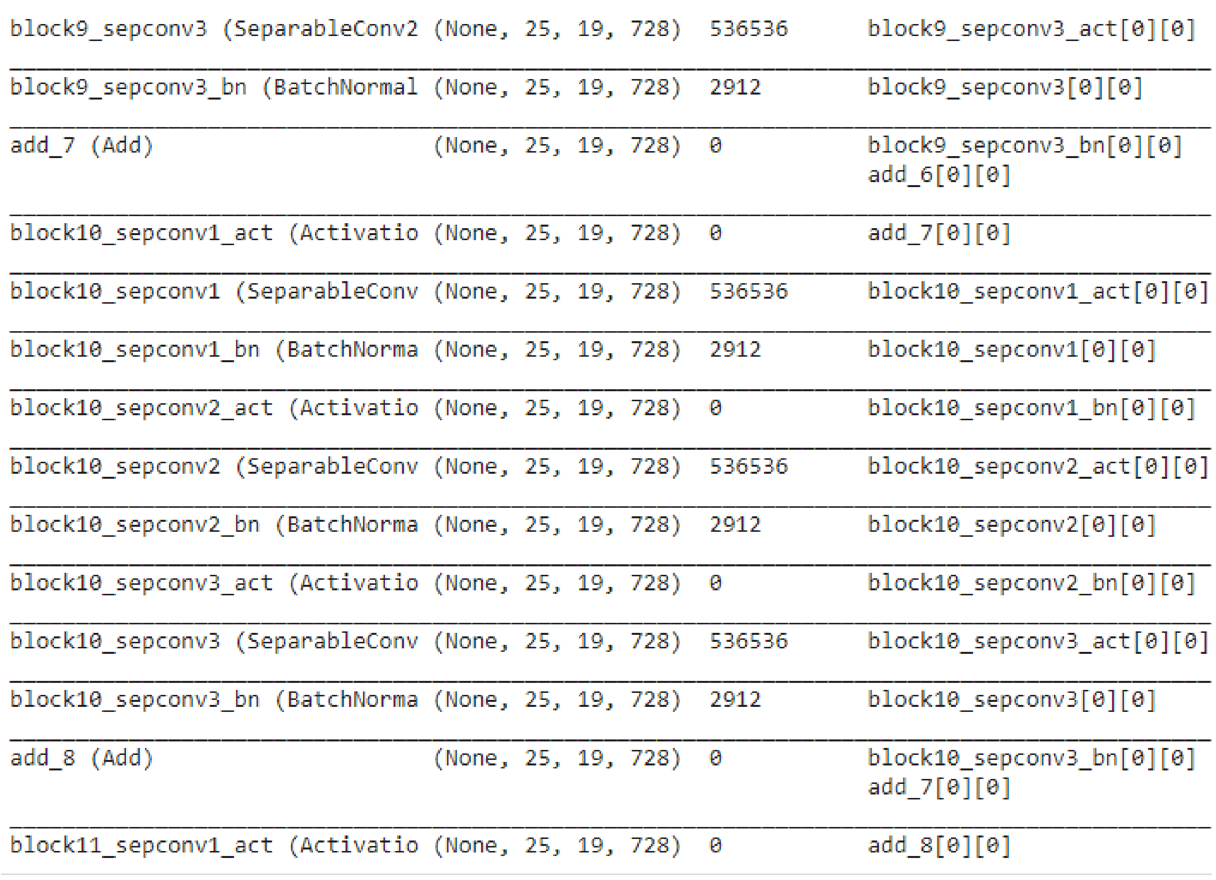
```
block5_sepconv2_bn (BatchNormal (None, 25, 19, 728)  2912        block5_sepconv2[0][0]
__________________________________________________________________________________________________
block5_sepconv3_act (Activation (None, 25, 19, 728)  0           block5_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block5_sepconv3 (SeparableConv2 (None, 25, 19, 728)  536536      block5_sepconv3_act[0][0]
__________________________________________________________________________________________________
block5_sepconv3_bn (BatchNormal (None, 25, 19, 728)  2912        block5_sepconv3[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 25, 19, 728)  0           block5_sepconv3_bn[0][0]
                                                                 add_2[0][0]
__________________________________________________________________________________________________
block6_sepconv1_act (Activation (None, 25, 19, 728)  0           add_3[0][0]
__________________________________________________________________________________________________
block6_sepconv1 (SeparableConv2 (None, 25, 19, 728)  536536      block6_sepconv1_act[0][0]
__________________________________________________________________________________________________
block6_sepconv1_bn (BatchNormal (None, 25, 19, 728)  2912        block6_sepconv1[0][0]
__________________________________________________________________________________________________
block6_sepconv2_act (Activation (None, 25, 19, 728)  0           block6_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block6_sepconv2 (SeparableConv2 (None, 25, 19, 728)  536536      block6_sepconv2_act[0][0]
__________________________________________________________________________________________________
block6_sepconv2_bn (BatchNormal (None, 25, 19, 728)  2912        block6_sepconv2[0][0]
__________________________________________________________________________________________________
block6_sepconv3_act (Activation (None, 25, 19, 728)  0           block6_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block6_sepconv3 (SeparableConv2 (None, 25, 19, 728)  536536      block6_sepconv3_act[0][0]
__________________________________________________________________________________________________
block6_sepconv3_bn (BatchNormal (None, 25, 19, 728)  2912        block6_sepconv3[0][0]
```



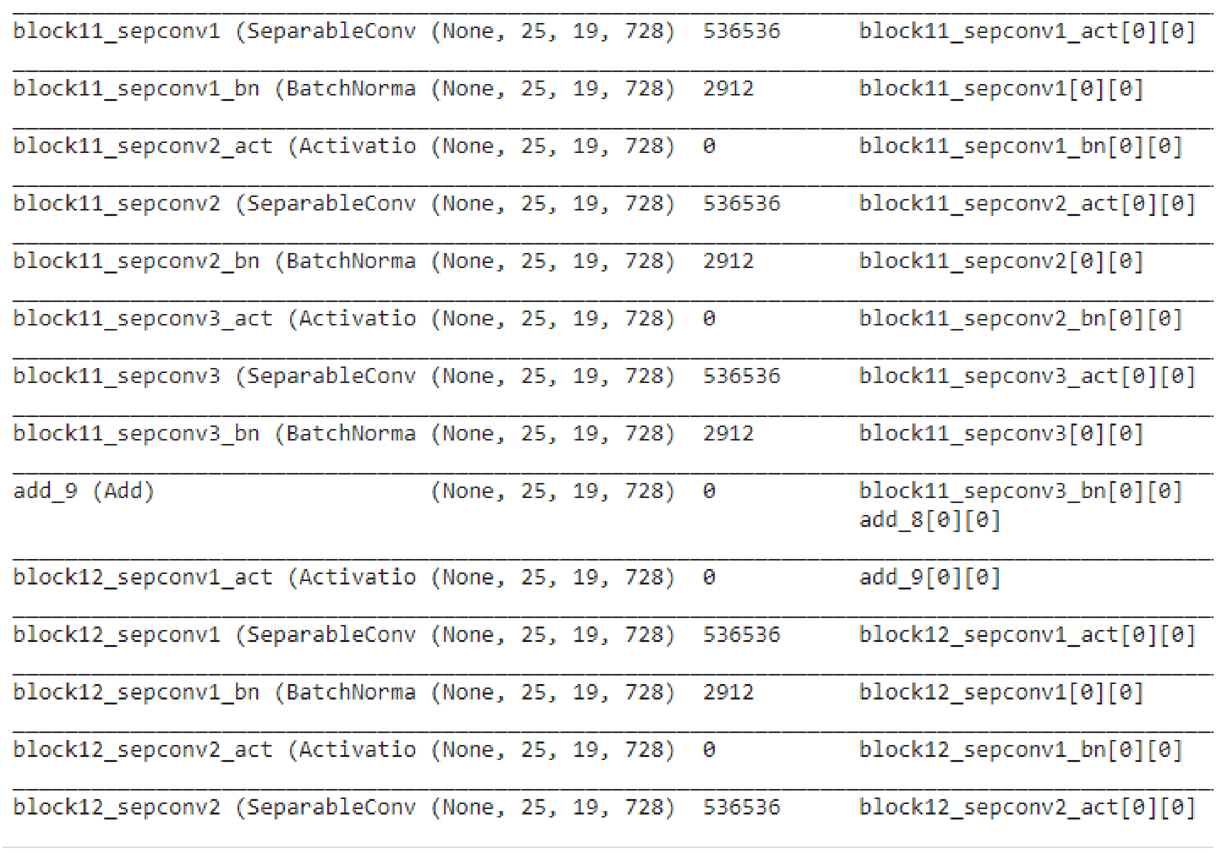
```
add_4 (Add)                     (None, 25, 19, 728)  0           block6_sepconv3_bn[0][0]
                                                                 add_3[0][0]
__________________________________________________________________________________________________
block7_sepconv1_act (Activation (None, 25, 19, 728)  0           add_4[0][0]
__________________________________________________________________________________________________
block7_sepconv1 (SeparableConv2 (None, 25, 19, 728)  536536      block7_sepconv1_act[0][0]
__________________________________________________________________________________________________
block7_sepconv1_bn (BatchNormal (None, 25, 19, 728)  2912        block7_sepconv1[0][0]
__________________________________________________________________________________________________
block7_sepconv2_act (Activation (None, 25, 19, 728)  0           block7_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block7_sepconv2 (SeparableConv2 (None, 25, 19, 728)  536536      block7_sepconv2_act[0][0]
__________________________________________________________________________________________________
block7_sepconv2_bn (BatchNormal (None, 25, 19, 728)  2912        block7_sepconv2[0][0]
__________________________________________________________________________________________________
block7_sepconv3_act (Activation (None, 25, 19, 728)  0           block7_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block7_sepconv3 (SeparableConv2 (None, 25, 19, 728)  536536      block7_sepconv3_act[0][0]
__________________________________________________________________________________________________
block7_sepconv3_bn (BatchNormal (None, 25, 19, 728)  2912        block7_sepconv3[0][0]
__________________________________________________________________________________________________
add_5 (Add)                     (None, 25, 19, 728)  0           block7_sepconv3_bn[0][0]
                                                                 add_4[0][0]
__________________________________________________________________________________________________
block8_sepconv1_act (Activation (None, 25, 19, 728)  0           add_5[0][0]
__________________________________________________________________________________________________
block8_sepconv1 (SeparableConv2 (None, 25, 19, 728)  536536      block8_sepconv1_act[0][0]
__________________________________________________________________________________________________
block8_sepconv1_bn (BatchNormal (None, 25, 19, 728)  2912        block8_sepconv1[0][0]
```



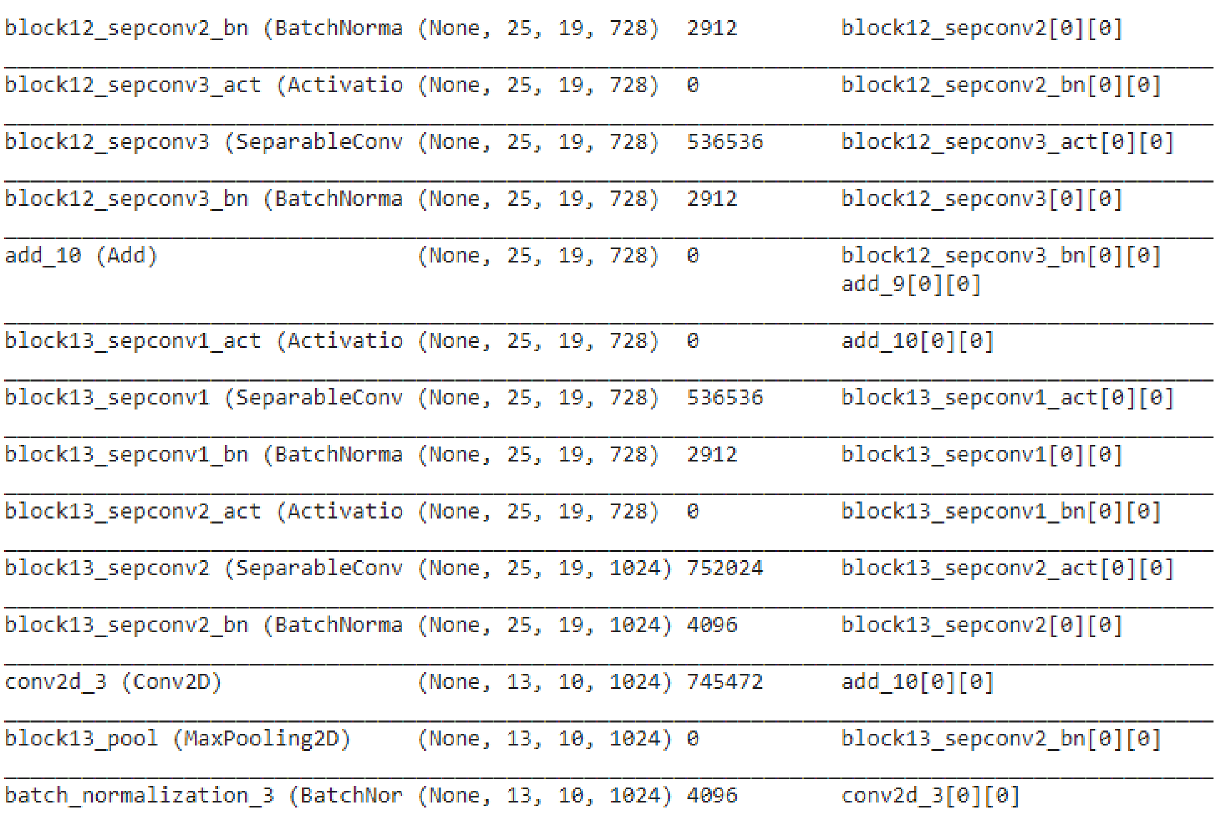
```
block8_sepconv2_act (Activation (None, 25, 19, 728)  0           block8_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block8_sepconv2 (SeparableConv2 (None, 25, 19, 728)  536536      block8_sepconv2_act[0][0]
__________________________________________________________________________________________________
block8_sepconv2_bn (BatchNormal (None, 25, 19, 728)  2912        block8_sepconv2[0][0]
__________________________________________________________________________________________________
block8_sepconv3_act (Activation (None, 25, 19, 728)  0           block8_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block8_sepconv3 (SeparableConv2 (None, 25, 19, 728)  536536      block8_sepconv3_act[0][0]
__________________________________________________________________________________________________
block8_sepconv3_bn (BatchNormal (None, 25, 19, 728)  2912        block8_sepconv3[0][0]
__________________________________________________________________________________________________
add_6 (Add)                     (None, 25, 19, 728)  0           block8_sepconv3_bn[0][0]
                                                                 add_5[0][0]
__________________________________________________________________________________________________
block9_sepconv1_act (Activation (None, 25, 19, 728)  0           add_6[0][0]
__________________________________________________________________________________________________
block9_sepconv1 (SeparableConv2 (None, 25, 19, 728)  536536      block9_sepconv1_act[0][0]
__________________________________________________________________________________________________
block9_sepconv1_bn (BatchNormal (None, 25, 19, 728)  2912        block9_sepconv1[0][0]
__________________________________________________________________________________________________
block9_sepconv2_act (Activation (None, 25, 19, 728)  0           block9_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block9_sepconv2 (SeparableConv2 (None, 25, 19, 728)  536536      block9_sepconv2_act[0][0]
__________________________________________________________________________________________________
block9_sepconv2_bn (BatchNormal (None, 25, 19, 728)  2912        block9_sepconv2[0][0]
__________________________________________________________________________________________________
block9_sepconv3_act (Activation (None, 25, 19, 728)  0           block9_sepconv2_bn[0][0]
```



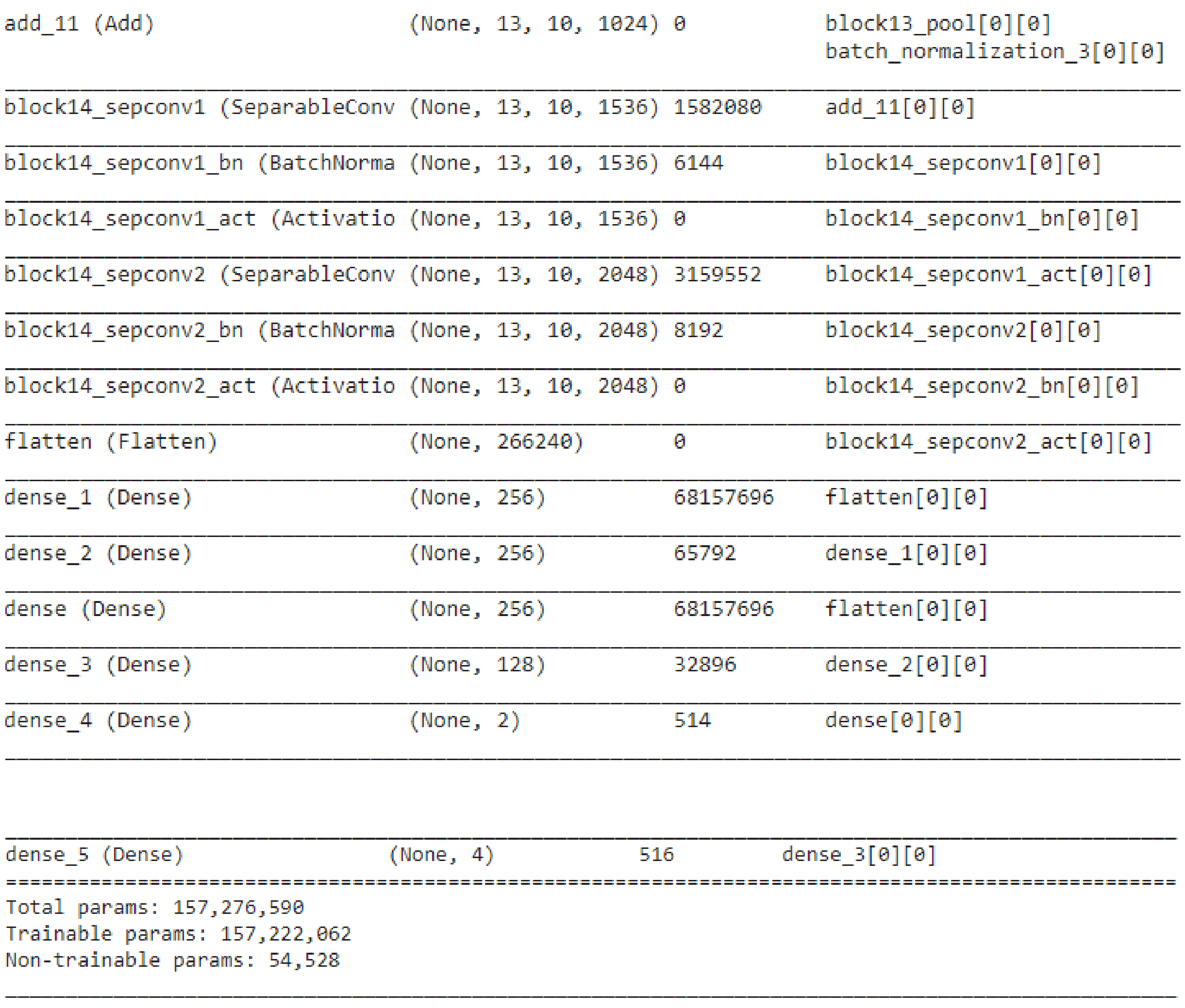
```
block9_sepconv3 (SeparableConv2 (None, 25, 19, 728)  536536      block9_sepconv3_act[0][0]
__________________________________________________________________________________________________
block9_sepconv3_bn (BatchNormal (None, 25, 19, 728)  2912        block9_sepconv3[0][0]
__________________________________________________________________________________________________
add_7 (Add)                     (None, 25, 19, 728)  0           block9_sepconv3_bn[0][0]
                                                                 add_6[0][0]
__________________________________________________________________________________________________
block10_sepconv1_act (Activatio (None, 25, 19, 728)  0           add_7[0][0]
__________________________________________________________________________________________________
block10_sepconv1 (SeparableConv (None, 25, 19, 728)  536536      block10_sepconv1_act[0][0]
__________________________________________________________________________________________________
block10_sepconv1_bn (BatchNorma (None, 25, 19, 728)  2912        block10_sepconv1[0][0]
__________________________________________________________________________________________________
block10_sepconv2_act (Activatio (None, 25, 19, 728)  0           block10_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block10_sepconv2 (SeparableConv (None, 25, 19, 728)  536536      block10_sepconv2_act[0][0]
__________________________________________________________________________________________________
block10_sepconv2_bn (BatchNorma (None, 25, 19, 728)  2912        block10_sepconv2[0][0]
__________________________________________________________________________________________________
block10_sepconv3_act (Activatio (None, 25, 19, 728)  0           block10_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block10_sepconv3 (SeparableConv (None, 25, 19, 728)  536536      block10_sepconv3_act[0][0]
__________________________________________________________________________________________________
block10_sepconv3_bn (BatchNorma (None, 25, 19, 728)  2912        block10_sepconv3[0][0]
__________________________________________________________________________________________________
add_8 (Add)                     (None, 25, 19, 728)  0           block10_sepconv3_bn[0][0]
                                                                 add_7[0][0]
__________________________________________________________________________________________________
block11_sepconv1_act (Activatio (None, 25, 19, 728)  0           add_8[0][0]
```



```
__________________________________________________________________________________________________
block11_sepconv1 (SeparableConv (None, 25, 19, 728)  536536      block11_sepconv1_act[0][0]
__________________________________________________________________________________________________
block11_sepconv1_bn (BatchNorma (None, 25, 19, 728)  2912        block11_sepconv1[0][0]
__________________________________________________________________________________________________
block11_sepconv2_act (Activatio (None, 25, 19, 728)  0           block11_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block11_sepconv2 (SeparableConv (None, 25, 19, 728)  536536      block11_sepconv2_act[0][0]
__________________________________________________________________________________________________
block11_sepconv2_bn (BatchNorma (None, 25, 19, 728)  2912        block11_sepconv2[0][0]
__________________________________________________________________________________________________
block11_sepconv3_act (Activatio (None, 25, 19, 728)  0           block11_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block11_sepconv3 (SeparableConv (None, 25, 19, 728)  536536      block11_sepconv3_act[0][0]
__________________________________________________________________________________________________
block11_sepconv3_bn (BatchNorma (None, 25, 19, 728)  2912        block11_sepconv3[0][0]
__________________________________________________________________________________________________
add_9 (Add)                     (None, 25, 19, 728)  0           block11_sepconv3_bn[0][0]
                                                                 add_8[0][0]
__________________________________________________________________________________________________
block12_sepconv1_act (Activatio (None, 25, 19, 728)  0           add_9[0][0]
__________________________________________________________________________________________________
block12_sepconv1 (SeparableConv (None, 25, 19, 728)  536536      block12_sepconv1_act[0][0]
__________________________________________________________________________________________________
block12_sepconv1_bn (BatchNorma (None, 25, 19, 728)  2912        block12_sepconv1[0][0]
__________________________________________________________________________________________________
block12_sepconv2_act (Activatio (None, 25, 19, 728)  0           block12_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block12_sepconv2 (SeparableConv (None, 25, 19, 728)  536536      block12_sepconv2_act[0][0]
```



```
block12_sepconv2_bn (BatchNorma (None, 25, 19, 728)  2912        block12_sepconv2[0][0]
__________________________________________________________________________________________________
block12_sepconv3_act (Activatio (None, 25, 19, 728)  0           block12_sepconv2_bn[0][0]
__________________________________________________________________________________________________
block12_sepconv3 (SeparableConv (None, 25, 19, 728)  536536      block12_sepconv3_act[0][0]
__________________________________________________________________________________________________
block12_sepconv3_bn (BatchNorma (None, 25, 19, 728)  2912        block12_sepconv3[0][0]
__________________________________________________________________________________________________
add_10 (Add)                    (None, 25, 19, 728)  0           block12_sepconv3_bn[0][0]
                                                                 add_9[0][0]
__________________________________________________________________________________________________
block13_sepconv1_act (Activatio (None, 25, 19, 728)  0           add_10[0][0]
__________________________________________________________________________________________________
block13_sepconv1 (SeparableConv (None, 25, 19, 728)  536536      block13_sepconv1_act[0][0]
__________________________________________________________________________________________________
block13_sepconv1_bn (BatchNorma (None, 25, 19, 728)  2912        block13_sepconv1[0][0]
__________________________________________________________________________________________________
block13_sepconv2_act (Activatio (None, 25, 19, 728)  0           block13_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block13_sepconv2 (SeparableConv (None, 25, 19, 1024) 752024      block13_sepconv2_act[0][0]
__________________________________________________________________________________________________
block13_sepconv2_bn (BatchNorma (None, 25, 19, 1024) 4096        block13_sepconv2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 13, 10, 1024) 745472      add_10[0][0]
__________________________________________________________________________________________________
block13_pool (MaxPooling2D)     (None, 13, 10, 1024) 0           block13_sepconv2_bn[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 13, 10, 1024) 4096        conv2d_3[0][0]
```



```
add_11 (Add)                    (None, 13, 10, 1024) 0           block13_pool[0][0]
                                                                 batch_normalization_3[0][0]
__________________________________________________________________________________________________
block14_sepconv1 (SeparableConv (None, 13, 10, 1536) 1582080     add_11[0][0]
__________________________________________________________________________________________________
block14_sepconv1_bn (BatchNorma (None, 13, 10, 1536) 6144        block14_sepconv1[0][0]
__________________________________________________________________________________________________
block14_sepconv1_act (Activatio (None, 13, 10, 1536) 0           block14_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block14_sepconv2 (SeparableConv (None, 13, 10, 2048) 3159552     block14_sepconv1_act[0][0]
__________________________________________________________________________________________________
block14_sepconv2_bn (BatchNorma (None, 13, 10, 2048) 8192        block14_sepconv2[0][0]
__________________________________________________________________________________________________
block14_sepconv2_act (Activatio (None, 13, 10, 2048) 0           block14_sepconv2_bn[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 266240)       0           block14_sepconv2_act[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          68157696    flatten[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          65792       dense_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          68157696    flatten[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 128)          32896       dense_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 2)            514         dense[0][0]
__________________________________________________________________________________________________


__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 4)            516         dense_3[0][0]
==================================================================================================
Total params: 157,276,590
Trainable params: 157,222,062
Non-trainable params: 54,528
__________________________________________________________________________________________________
```

Figure 4.3: Final architecture

4.4.2 Evaluation metric&Loss function

The selection of a performance measure is important in machine learning project. In our project we used a complex cost function to train the network as we will see later, and for evaluating it we used both the accuracy for classification metric which is good if we get a value above 95% and IoU (intersection over union) for regression which is fine if we get a value above 50%.

- Intersection over union (or Jaccard index) it is a common metric to evaluate how well the model can predict the bounding box, it is the area of the intersection of the predicted bounding box and the true bounding box divided by their union, the larger the area of overlap the greater the IoU.

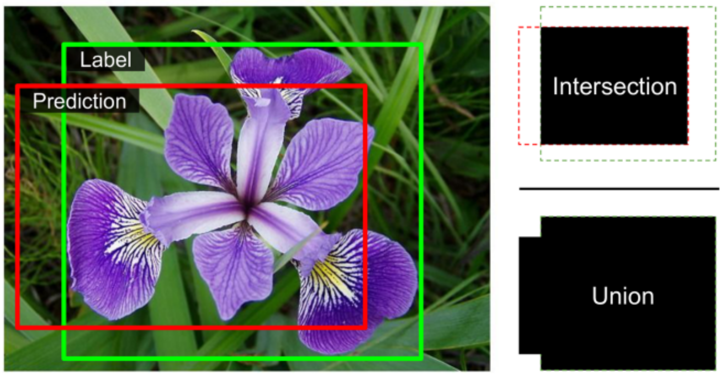


Figure 4.4: Intersection Over Union.[6]

- Accuracy : The term of accuracy is usually meant the classification accuracy, it is used only for classification tasks. It is a metric to evaluate how well the model perform on unseen data, and it is the ratio of number of correct classification to the total number of input samples.

  $$\text{accuracy} = \frac{numberofcorrectprediction}{Totalnumberofexamples}$$

- MSE (Mean Squared Error) is a typical performance measure for regression tasks. We can use it as a cost function to train the model in order to approximate the coordinate of the bounding box, and we can also use it as a metric to evaluate the model. All it does is just take the average of the square of the difference between the true values and the predicted values.

  $$\text{MSE} = \frac{1}{n}\Sigma_{i=1}^{n}\left(Y_i - \hat{Y_i}\right)^2$$ [6]

- Cross-entropy loss

  This loss is used for optimization classification models, it takes the predicted probabilities and measure the distance from the true probabilities, in other words it is used to measure how well the estimated class probabilities matches the target classes.

  $$I(\Theta) = -\frac{1}{m}\Sigma_{i=1}^{m}\Sigma_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- Objective function

  In our localization problem we deal with two tasks at the same time, so we need a complex loss function to minimize. The loss function that we used is a merge of of the MSE loss which it works well in the regression task in order to predict the coordinate of the bounding box and the cross-entropy loss function which is used for classification task in order to predict the category of the object in the image (either cat or dog). Each of these two loss functions are multiplied by coefficients. Tweaking these coefficients depends on what we care most about.

  Loss function $= \lambda_1 MsE() + \lambda_2 CrossEntropy()$

4.5 Development process

- Pre-processing the dataset :

After we downloaded Oxford-IIIT Pet Dataset which contains 3685 images with their associated annotations, we preprocessed the data before we fed it to the network. We extracted the information including the labels from each annotations which are in XML files using the XML library then we put the labels in a numpy array and we got an array of 3685 rows and 6 columns, the first 4 columns represents the coordinate of the bounding box that are normalized by the size of the image and the 2 last columns is for the category of the image because we used one-hot-vector normalized to encode the classes that means [1,0] represents a cat and [0,1] represents a dog. we read the images using PIL library, then we resize the images into 400 width and 300 height to reduce the using memory, then we made a normalization to the images by deviding them by 255 because the TensorFlow library that is used for training accepts the input image in the range 0 to 1. After that we put them in a numpy array and we got an array of [3685,400,300,3] (3685 images, 400 width, 300 height and 3 columns). Finally we split the processed data into 90% for training and 10% for test using train-test-split of the scikit-learn library, and we save the 4 array into nympy files, so we don't repeat the pre-processing again after each time we want to train a model.

- Creating the network & training it :

  We loaded the saved training data, and also split it into training data validation data, then we created the architecture of the network using the functional API of Keras library, after that we chose the optimize, loss function and evaluation metric using the compil method. Finally we started the training by the fit method.

- Fine tuning : we said earlier that our objective function that we want to minimize is a merge of MSE loss and cross entropy loss and we and that we must try several combination of the hyperparametres in order to minimize it we started by connecting the two output layers with predefined network.

  We experimented 3 predefined model VGG16, VGG19 and Xception and turn out that Xception performed better.

  Optimizer : We trained the models using RMS prop optimizer with learning rate $2e^{-5}$.

  Batch size : We set the batch size to 20.

  Epochs : we tried many epochs 30,60,70,100 and 130, it turns out that after 100 epochs the learning algorithm converges.

  Layers & neurons : We added fully connected layers before each output layer, and we got a good performance on classification when we added one dense layer that contains 256 neurons. And for regression we tried to tune the number of layer and the number of neurons and we ended adding 3 dense layers (256, 128 and 64) respectively to get a higher IoU.

  Activation : We kept ReLu a the activation function for every layer that we added.

  Loss-weights : It is an important hyper-parameter which represents the two coefficients of two loss function we tried many combinations and turned out that if we set the coefficient of MSE to 0.9 and the coefficient of cross-entropy to 0.1 we got a good results.

4.6 Results and Discussion

In order to improve the performance we trained a lot of models each time with different combination of hyperparameters. In this section we will talk about an important model that made an obvious improvement in the performance while highlighting the hyperparameters we tweaked to improve it.

In the first model we reused the VGG16 architecture for the extraction features and connected its outputs directly to the regression output layer but we added a dense layer that contains 256 neurons before the classification output layer, we started with 30 epochs.

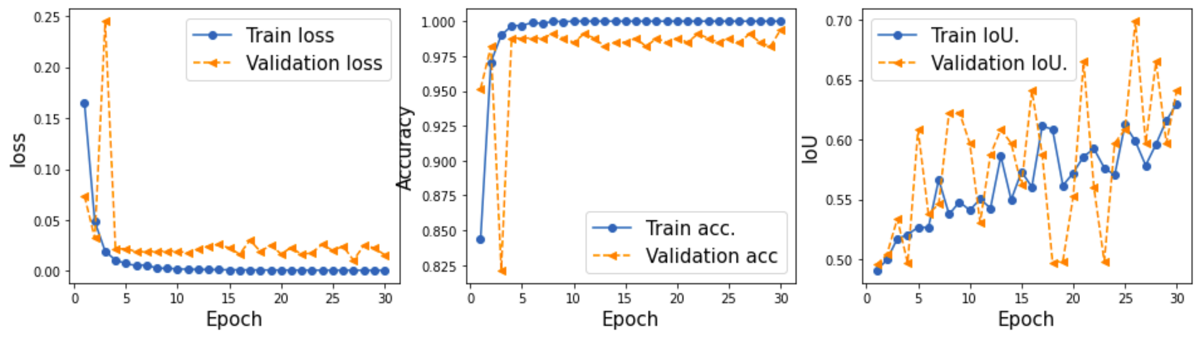
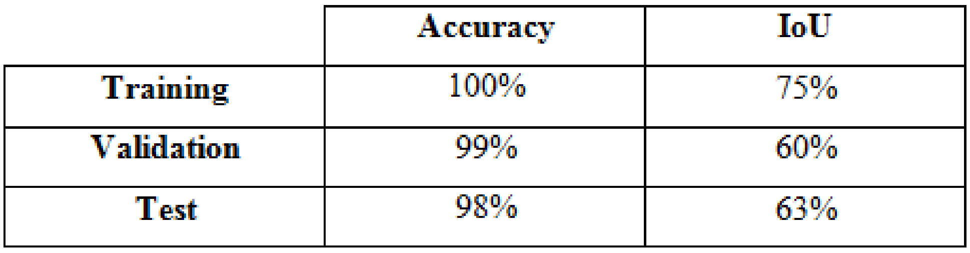


Figure 4.5: The training and validation losses,IoUs and accuracies



	Accuracy	IoU
Training	100%	75%
Validation	99%	60%
Test	98%	63%

Figure 4.6: Accuracy and IoU metrics.

We noticed that our model did well for the classification, we got more than 98% accuracy on validation data , but for regression we got only 63% on IoU, which need to be increased .so we only focused on improving the IoU. Also, we tried to tweak the loss-weights hyperperamters and we got a good performance when we set the coefficients at 0.9 for mse loss function and 0.1 for cross-entropy.

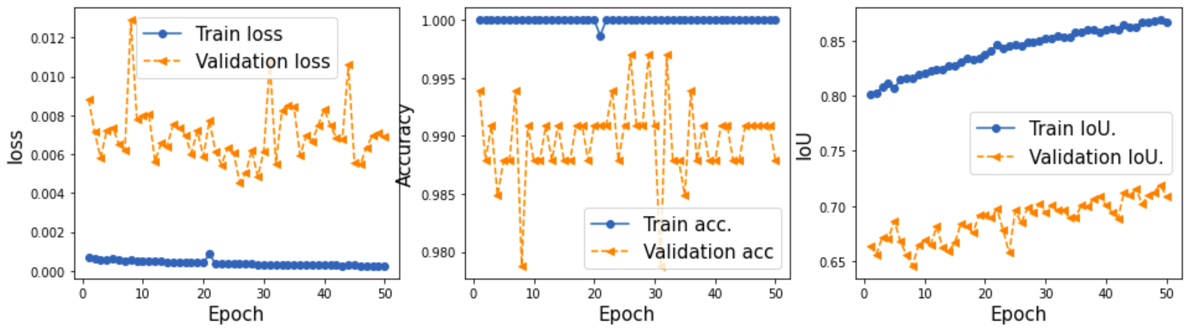
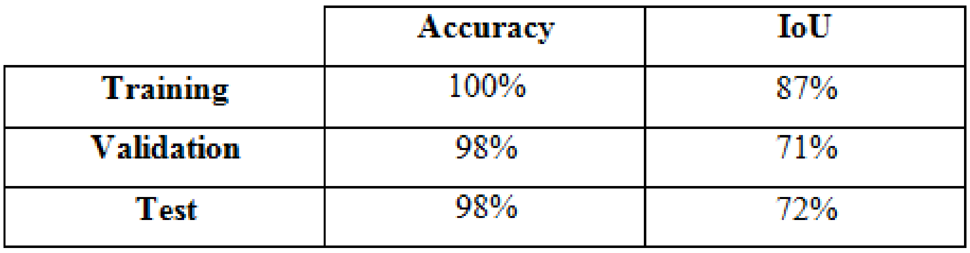


Figure 4.7: The training and validation losses,IoUs and accuracies



	Accuracy	IoU
Training	100%	87%
Validation	98%	71%
Test	98%	72%

Figure 4.8: Accuracy and IoU metrics.

Then we noticed that the accuracy is still 99% but the IoU increased a little bit and reached 71%, so we tuned the layers and the neurons. We saw an improvement when we added 2 dense layers with 256,128 neurons respectively.

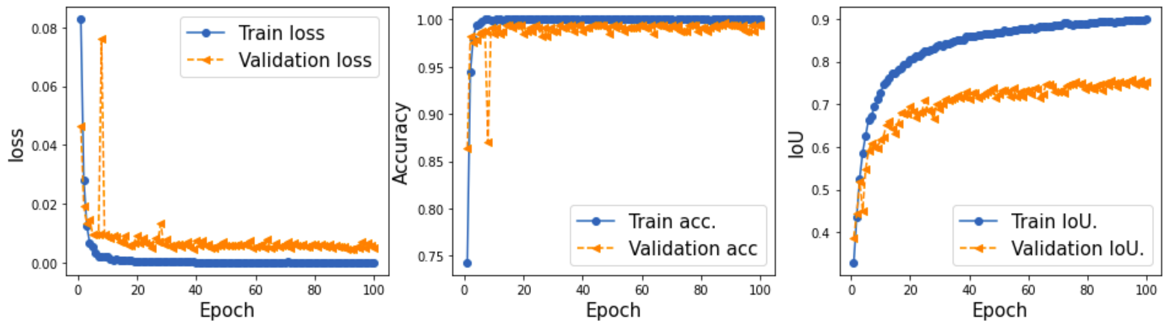
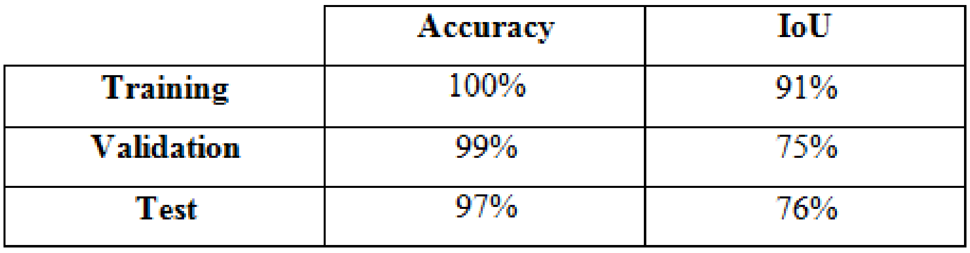


Figure 4.9: The training and validation losses,IoUs and accuracies



	Accuracy	IoU
Training	100%	91%
Validation	99%	75%
Test	97%	76%

Figure 4.10: Accuracy and IoU metrics.

We can see that IoU reached 76%, so we continued tweaking the layers and neurons with increasing the number of epochs each time,but it didn't make a big progress, so we replaced the VGG architecture by the Xception architecture . We tweaked the layers and neurons again which lead us to our final model that got 100% accuracy and 78% IoU.

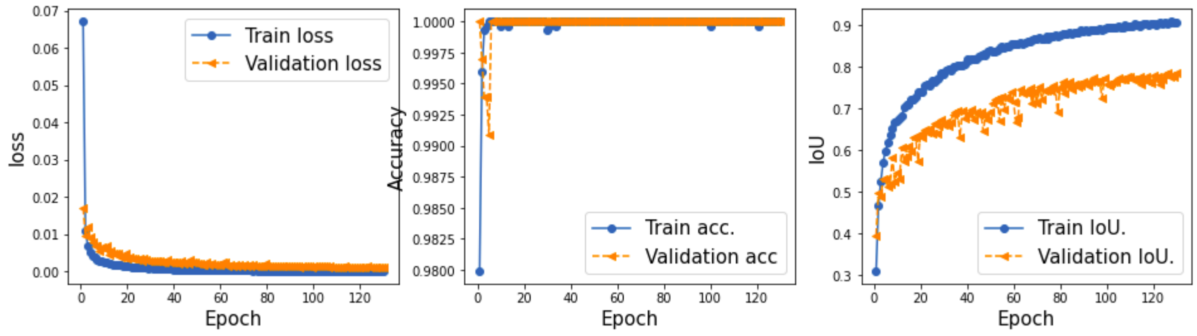
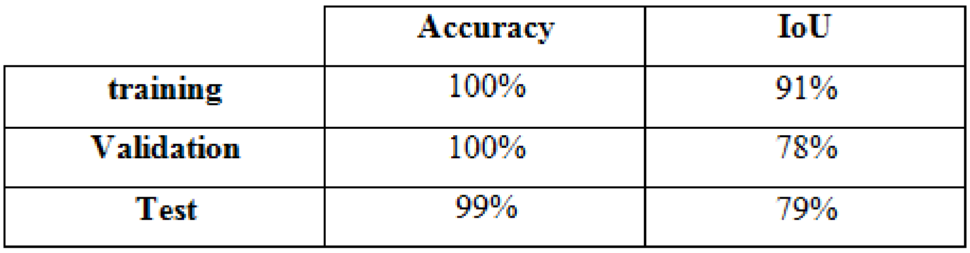


Figure 4.11: The training and validation losses,IoUs and accuracies.



	Accuracy	IoU
training	100%	91%
Validation	100%	78%
Test	99%	79%

Figure 4.12: Accuracy and IoU metrics.

We tested our final model on our test dataset, we got the results below figure [see figure 4.13] the yellow rectangle is the target bounding box, and the red is the predicted one.

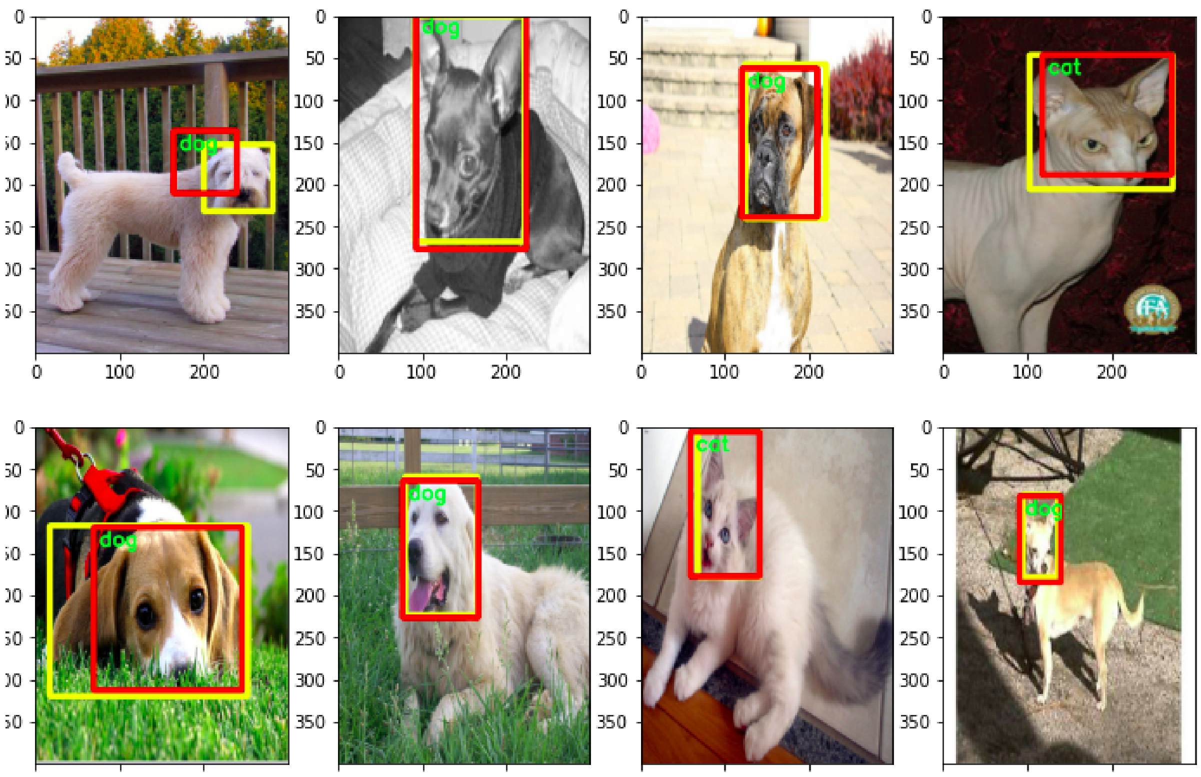
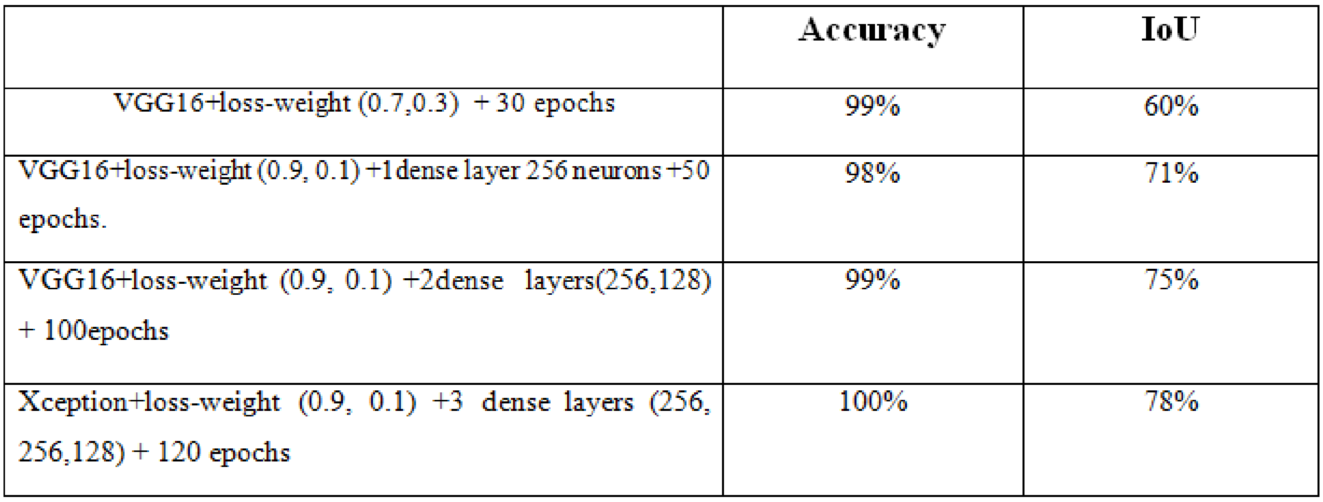


Figure 4.13: Prediction vs target.

The following table figure [see Figure 4.14] represent the summary of the results (IoU validation) scores over validation :



	Accuracy	IoU
VGG16+loss-weight (0.7,0.3) + 30 epochs	99%	60%
VGG16+loss-weight (0.9, 0.1) +1dense layer 256 neurons +50 epochs.	98%	71%
VGG16+loss-weight (0.9, 0.1) +2dense layers(256,128) + 100epochs	99%	75%
Xception+loss-weight (0.9, 0.1) +3 dense layers (256, 256,128) + 120 epochs	100%	78%

Figure 4.14: IoU Summary

4.7 Conclusion

In this chapter, we presented the process of developing a neural network to tackle the localization problem. We mentioned that it is not a good idea to train a deep neural network from scratch, instead we can use pre-trained models(transfer learning). Finally we saw how we tuned the hyperparmeter in order to improve the performance and realized that some hyperparmeter effect more.

General conclusion

In this work, we have introduced the computer vision discipline that includes several tasks, such as object recognition, segmentation and detection. solving the object localization task usually needs a set of features such as HAAR cascade or HOG these features must change according to the type of the localized object and this create an additional overhead. With the arrival of deep learning era it will be possible to both learn the representation subproblem (feature extraction) and the regression/classification subproblem. We leverage the concepts offered by deep learning and use them to tackle our localization problem which consists of recognizing the category of the main object (in our case cat or dog), in addition to the drawing of a bounding box around the detected object. our model mainly consists of a feature extractor termed Xception and a set of dense layers that capture the categories and the bounding box. We also notice that the proposed model has achieved a classification rate of 100% and an IoU rate of 78% on the validation dataset.
As a perspective of this work, we can compare our results with more elaborated feature extractors such as SeNet,ResNet; Additionally, we can extend the presented model to tackle the detection task which assume that the image contains several objects of different classes.

Bibliography

[1] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In Computer VisionECCV 2008, pages 2-15. Springer, 2008. 4

[2] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” arXiv preprint arXiv: 1610.02357 2016.

[3] F.Chollet . Deep learning with Python. Manning Publications Co,2017.

[4] N.DALAL and B.TRIGGS. *Histograms of oriented gradients for human detection.* In : 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05). Ieee, 2005. p. 886-893.

[5] G.E. Hinton et al., “Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors,” arXiv preprint arXiv:1207.0580 2012.

[6] A.GÉRON, . *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O’Reilly Media, 2019.

[7] I.GOODFELLOW, Y,BENGIO and A,COURVILLE. Deep learning (adaptive computation and machine learning series). Cambridge Massachusetts, 2017, p. 321-359.

[8] R. C.GONZALEZ, R. E. WOODS, *Digital Image Processing Prentice.* 2006.

[9] K.HAN,Y. WANG,H.CHEN, et al. A survey on visual transformer. arXiv preprint arXiv:2012.12556, 2020.

[10] R.HECHT-NIELSEN. Theory of the backpropagation neural network. In : Neural networks for perception. Academic Press, 1992. p. 65-93.

[11] A.Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks,” Proceedings of the 25th International Conference on Neural Information Processing Systems 1 2012: 1097–1105.

[12] D.R.MARTIN, FOWLKES, Charless C., et MALIK, Jitendra. *Learning to detect natural image boundaries using local brightness, color, and texture cues.* IEEE transactions on pattern analysis and machine intelligence, 2004, vol. 26, no 5, p. 530-549.

[13] B.Planch , E.Andres *hands computer vision with tensorflow 2.*Packt Publishing Ltd,2019.

[14] S.JD.PRINCE. *Computer vision: models, learning, and inference.* Cambridge University Press, 2012.

[15] S.RASCHKA and V.MIRJALILI. Python Machine Learning: Machine Learning and Deep Learning with Python. Scikit-Learn, and TensorFlow. Second edition ed, 2017.

[16] O.RUSSAKOVSKY, J.DENG, H.SU, *et al. Imagenet large scale visual recognition challenge. International journal of computer vision*, 2015, vol. 115, no 3, p. 211-252.

[17] J.E.SOLEM, . *Programming Computer Vision with Python: Tools and algorithms for analyzing images.* " O'Reilly Media, Inc.", 2012.

[18] K.Simonyan and A.Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recog- nition," arXiv preprint arXiv:1409.1556 2014.

[19] C. Touzet. LES RESEAUX DE NEURONES ARTIFICIELS, INTRODUCTION AU CONNEXIONNISME : COURS, EXERCICES ET TRAVAUX PRATIQUES. EC2, 1992, Collection de l'EERIE, N. Giambiasi. hal-01338010

[20] P.VIOLA, M.JONES. Rapid object detection using a boosted cascade of simple features. In : *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition.* CVPR 2001. Ieee, 2001. p. I-I.

[21] ICHI.PRO, "Comprendre la descente de gradient de manière simple." https://ichi.pro/fr/comprendre-la-descente-de-gradient-de-maniere-simple-206129876149300.

[22] learnopencv,"https://learnopencv.com/histogram-of-oriented-gradients/".

[23] Wikipedia, "Réseau de neurones artificiels." https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neur