Report (Draft version)

LE Van Linh and BEURTON-AIMAR Marie

September 21, 2018

Abstract

This document is working on the morphormetry and machine learning studied. In morphometry part, we introduce the techniques which are in image processing and morphometry analysis such as segmentation methods or dominant points method. At the end of this part is MAELab, a software is constructed to distribute the studied method on the insect. In the machine learning part, we present the basic knowledge of machine learning and deep learning, specially convolutional neural network (CNN). We also introduce about a CNN that we use to classify the image in context of this thesis.

Contents

Part I

Morphometry

Morphometry (or morphometrics) is a norm refers to the analysis of form, specifics size and shape of the object in digital image. Morphometry analyses are commonly performed on organisms, and are useful in analyzing the structure of the organisms. Morphometry can be used to extract the general information of creatures, or reconstruct the shape of the organism based on the analysed information that we had. Besides, morphometry can also detect the changes on creatures. Based on these information, we can statically the hypotheses about the factors that affect to the changes of shape.

We have three distinct approaches are usually use: traditional morphometry, landmark-based morphometry and outline-based morphometry.

1. Traditional morphometry: measure the size of the object such as the length, width, angle, ratio and areas on object. The traditional morphomentry is using many measurement of size that most will be highly correlated; as a result, there are few independent variables despite the many measurements.

2. Landmark-based morphometry: finding enough landmark to provide a comprehensive description of shape. From the set of beginning landmarks, we can estimate the the missing landmarks.

3. Outline-based morphometry: a mathematical functions is fitted to points sampled along the outline.

In this part, we will describe the methods to analysis the morphometry based on the segmentation, dominant points of the image. The method is through the several technique in image processing. At the end of this part, a software had built to verify the linkage of the steps.

Chapter 1

Segmentation

Segmentation is an importance process in computer vision, the goal of segmentation is to change the representation of an image into another way with more meaningful and easier to analyze. In another point of view, segmentation is process to assign the label to every pixel in an image to detect the pixels that have the same characteristics. The result of image segmentation is a set of distinct pixels with difference characteristics, or a set of contours extracted from the image.

In computer vision, we have a lot of methods to segment an image. In the context of this chapter, we will mention the commonly method for segmentation. Besides, we also introduce the ways to record the information extracted from the contours of the image.

1.1 Threshold

Threshold is a basically method in segmentation, but it is widely used in image processing to remove the unnecessary characteristics as well as keep the information interested. The main parameter in threshold method is **threshold value**. This value seems a border to seperate the necessary and unnecessary characteristics. Depending on the requirements of application, the user will be choose difference types of threshold.

1.1.1 Types of threshold

Let's consider that we have a source image with pixels with its intensity `src(x,y)`, threshold value `thresh`, a maximum value (`maxValue`) to assign to the result.

Binary threshold

The new intensity of pixel will be set to `maxValue` if the intensity of pixel `src(x,y)` is greater than `thresh` value. Otherwise, the new intensity will be set to `0`.

$$dst(x,y) = \begin{cases} maxValue & \text{if } \texttt{src(x,y)} > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Binary threshold inverted

The new intensity of pixel will be set to `0` if the intensity of pixel `src(x,y)` is greater than `thresh` value. Otherwise, the new intensity will be set to `maxValue`.

$$dst(x,y) = \begin{cases} 0 & \text{if } \texttt{src(x,y)} > \texttt{thresh} \\ maxValue & \text{otherwise} \end{cases} \tag{1.2}$$

Truncate threshold

The new intensity of pixel will be set to `thresh` value if the intensity of pixel `src(x,y)` is greater than `thresh` value. Otherwise, the new intensity is not change. It means in this case, the maximumvalue for the pixels is `thresh` value, if `src(x,y)` is greater than `thresh`, then its value is truncated.

$$dst(x,y) = \begin{cases} \texttt{thresh} & \text{if } \texttt{src(x,y)} > \texttt{thresh} \\ \texttt{src(x,y)} & \text{otherwise} \end{cases} \tag{1.3}$$

Threshold to zero

The intensity of destination pixels are kept the same with the source if its intensity `src(x,y)` is greater than `thresh`. Otherwise, new pixel value will be set to `0`.

$$dst(x,y) = \begin{cases} \texttt{src(x,y)} & \text{if } \texttt{src(x,y)} > \texttt{thresh} \\ \texttt{0} & \text{otherwise} \end{cases} \tag{1.4}$$

Threshold to zero inverted

The intensity of destination pixels will be set to 0 if if its intensity `src(x,y)` is greater than `thresh`. Otherwise, new pixel value is keep the same with the source.

$$dst(x,y) = \begin{cases} \texttt{0} & \text{if } \texttt{src(x,y)} > \texttt{thresh} \\ \texttt{src(x,y)} & \text{otherwise} \end{cases} \tag{1.5}$$

1.1.2 Threshold value parameter

The key of threshold method is how to decide the threshold value ? In most of application, the threshold value is setted by manual. Besides, this value can be detect automatically. In the context of thesis, we applying a method to determine the threshold value by analysing the histogram of the gray-scale image. For the histogram, we indicate the peaks and valley of the histogram. The threshold value is mean between two values at the histogram slopes(between peak and valley).

1.1.3 Post process

In threshold method, if we can choose a good threshold value, the segmentation is very good. However, removing wrong the interested characteristics is unavoidable. So, after thresholing the image, we continue post-process to evaluate the result of segmetation. The post-process method is based on "line scan" idea. It is used to fixed the wrong hole on the object in the image. The steps of method are followed:

1. Find the begin of the hole (initialize line)

- Find the first endpoint of line: pixel that moves from the object to backgound (hole).
- Find the second endpoint of the line: pixel that moves form the background (hole) to the object.
- Check the lines is the first line of this hole or not? If it is not correct, go to step 4.

2. While not end the hole
   - Recored the line
   - Increase a row
   - Find the limit endpoints of the new line
   - Check the new line is the end of hole or not? If it is not the end, repeat step 2. Otherwise, go to step 3.
3. For each line recored, fill the hole's pixel with value of object.
4. Clear the lines and continue with next pixel in the image
5. The algorithm will stop when all pixels are considered.

Figure 1.1: A result of threshold method

In figure 1.1, the left image shows result after applying binary threshold on gray-scale image; the right image shows the result of binary threshold and post-process.

1.2 Canny algorithm

In 1986, **John F.Canny** had proposed a method to determine the edge in image. This is a technique to detect the useful structure of the object in digital image. Until now, the Canny algorithm[1] is used widely for the segmentation in computer vision. The process of Canny algorithm can be described in 4 steps as follows:

1. Smoothing the image to reduce the noises by using Gaussian filter
2. Finding the intensity and direction gradient of each pixel in image
3. Eliminating the weak edge by using the edge thinning technique.
4. Applying double threshold to determine the potential edges

1.2.1 Gaussian filter

To smooth the image, a Gaussian filter is applied to convolve with the image. This step will help to reduce the effects of the noises on the edge detector. Normally, the equation of a Gaussian kernel with size $(2k+1)$ x $(2k+1)$ is computed as:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}); 1 \leq i, j \leq (2k+1) \tag{1.6}$$

where k is the size of kernel, and it should be a odd number.
For example, a 3x3 Gaussian filter with $\sigma = 1$ as followed:

$$G = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.7}$$

The selection of the size of the Gaussian kernel is important, it will affect the performance of the detector. If the size of the kernel is large, the detector can be sensitive to noise; otherwise, if the kernel's size is small, the detector can be destroy many strong edge. In the practice, this step is combined into Sobel convolution with a 3x3 kernel, which used to finding the intensity and direction gradients at each pixels of image.

1.2.2 Sobel convolution

The points belong to the edge in an image can stay in any direction, so the Canny algorithm uses four filters to detect the edges (vertical, horizontal and two diagonal edges) in the image. And the Sobel operator is used to detect the edges. This operator returns a value for the first derivative in horizontal direction (G_x) and the vertical direction (G_y). From these values, the gradient and direction of edge at each pixel are determined:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.8}$$

$$\phi = atan2(G_y, G_x) \tag{1.9}$$

In this case, the kernel of Sobel convolution is 3x3, and it is also combined the Gaussian filter to smooth the image. The kernels are used to convolute the horizontal direction and vertical direction as follows:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.10}$$

The edge direction angle is rounded to one of four angles which were presented for four directions: vertical, horizontal, and two diagonals $0^o, 45^o, 90^o$ and 135^o.

1.2.3 Non-maximum suppression

Non-maximum suppression is applied to thin the edge in an image. Thus, this operation is used to suppress all the gradient values to 0 except the local maximal. At every pixel, it suppress the gradient value of the center pixels if its magnitude is smaller than the magnitude of one out of two neighbors in the gradient direction. In details:

- If the gradient direction angle is **0** degree, the point will be considered to be on the edge if the gradient magnitude is greater than the magnitude at pixels in the **east** and **west** directions.

- If the gradient direction angle is **45** degree, the point will be considered to be on the edge if the gradient magnitude is greater than the magnitude at pixels in the **north east** and **south west** directions.
- If the gradient direction angle is **90** degree, the point will be considered to be on the edge if the gradient magnitude is greater than the magnitude at pixels in the **north** and **south** directions.
- If the gradient direction angle is **135(-45)** degree, the point will be considered to be on the edge if the gradient magnitude is greater than the magnitude at pixels in the **north east** and **south west** directions.

1.2.4 Double threshold

After applying the non-maximum suppression, the edges pixels are presented. However, there are still some edge pixels effected by noise. Double threshold will filter out the edge pixels with the weak gradient value and preserve the edge with the hight gradient value.

- A pixel called strong pixel (hence, it belong to the edge), if the edge pixel's gradient value is higher than the high threshold value.
- A pixel will be suppressed, if the edge pixel's gradient value is smaller than the low threshold value.
- A pixel called weak pixel (can be belong to the edge or not), if the edge pixel's gradient value is larger than low threshold value and smaller than high threshold value. A weak pixel can be belong to the edge if it connected with a strong pixel in 8-connected; else, it will be suppressed.

Thus, the accuracy of algorithm is depended on two parameters: the kernel of Gaussian filter and thresholds value. As said before, if we choose incorrect the kernel size of Gaussian filter, we can not reduce the noise or we can remove the real edge. Besides, the values of double threshold is also important to filter out the edge pixels. In practice, 1:3 is the good ratio between lower threshold and upper threshold in Canny.

1.2.5 Summary

With applying double thresholding in the last stage, Canny had provied a strict condition to consider the weak edge as well as remove the pixels which were not belong to the edge. So far, Canny algorithm is good method to determined the edge in image, it is used by many application in image processing.

1.3 Suzuki algorithm

The Canny algorithm had detected the edge in the image. We can apply many difference methods to track the edge. **S. Suzuki** and **K. Abe**[2] had proprosed a method to get the border of object in image. This method is based on the topological structure analysis on binary image.

Following this method, it detects two kinds of border in image. The first is outer border, which is defined by a set of border points between an arbitrary 1-component and the 0-component which surrounds it directly; another type is hole border which refers to the set of border points

between a hole and the 1-componet which surrounds the hole directly. In this case, the 1-component (or 0-component) is connected component of 1-pixels (or 0-pixels).

In our case, our purpose is getting the edges which were detectecd by Canny[1]. An edge is consider as an outer border or a hole border does not important. So, the Suzuki algorithm could make some changes to fit with our aim. The processes of algorithm is desribed as follows:

Let an input binary image is $F = \{f_{ij}\}$. Set initially $NBD = 1$ (denoted the sequence number of border.)

1. Select one of the following:
   (a) If $f_{ij} = 1$ and $f_{i,j-1} = 0$, increment NBD, $(i_2, j_2) \leftarrow (i, j-1)$ (pixel (i, j) is the starting point of an outer border).
   (b) If $f_{ij} \geq 1$ and $f_{i,j+1} = 0$, increment NBD, $(i_2, j_2) \leftarrow (i, j+1)$ (pixel (i, j) is the starting point of an hole border).
   (c) Otherwise, go to step (3)
2. From the starting point **(i,j)**, the process to trace the edge is done by substeps following:
   2.1 Starting from point (i_2, j_2), look around clockwise the pixels in the neighborhood (8-connected) of (i, j) and find the first non-zero pixel (i_1, j_1). If no non-zero pixel is found, assign -NBD to f_{ij} and go to step (3)
   2.2 $(i_2, j_2) \leftarrow (i_1, j_1)$ and $(i_3, j_3) \leftarrow (i, j)$
   2.3 Starting from the **next element of the pixel** (i_2, j_2) in the counterclock-wise order, check the pixels neighborhood of current pixel (i_3, j_3) to find the first non-zero pixel (i_4, j_4).
   2.4 Chang the value f_{i_3,j_3} of the pixel (i_3, j_3) as follows:
      (a) If the pixels $(i_3, j_3 + 1)$ is a 0-pixel examined in the substep (2.3) then $f_{i_3,j_3} \leftarrow -NBD$. Else, $f_{i_3,j_3} \leftarrow NBD$ unless (i_3, j_3) is on an already border.
      (b) If the pixels (i_3, j_3+1) is not a 0-pixel examined in the substep (2.3) and $f_{i_3,j_3} = 1$ then $f_{i_3,j_3} \leftarrow NBD$
      (c) Otherwise, do not change f_{i_3,j_3}.
   2.5 If $(i_4, j_4) = (i, j)$ and $(i_3, j_3) = (i_1, j_1)$ (coming back to the starting point), then go to step (3); otherwise, $(i_2, j_2) \leftarrow (i_3, j3), (i_3, j_3) \leftarrow (i_4, j_4)$ and go back to step (2.3)
3. Resume the scan from the pixel $(i, j + 1)$. The algorithm is stop when the scan reaches the lower right corner of the image.

The obtained results from Suzuki algorithm are list of the edges of the object in the image. Each edge is list of connected points. This result is very important for the next steps of automatic detection landmarks.

1.4 Approximated lines

The list of points from Suzuki algorithm can used to present the object. But in some case to consider the feature of the object, the information from the list of points is not perfect. Instead of, we use another kind of the geometric object to represent the object. This is the line. In this section, we will describe the duration to get the approximated lines of object from the list of edge (each edge is represented by the list of points).

The method is used to fragment the edge into a list of approximated line is a recursive algorithm[3], which is a new improved version with the method proposed by Lowe[4] except the stop condition is considered as a parameter λ. The steps of algorithm are followed:

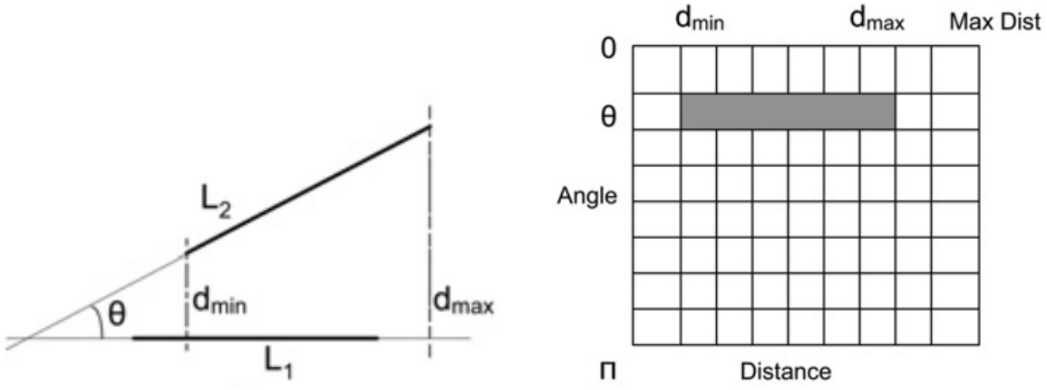
1. Create a straight line l between two endpoints of the edge
2. Calculate the perpendicular distance from each point on edge to line l, and identifying the maximum point p_m.
3. If the perpendicular distance from maximum point p_m is greater than the stop condition($d(p_m, l) > \lambda$), then the edge is split at this point and both parts are reprocessed. Otherwise, if we do not have any p_i that the perpendicular distance from they to l greater than λ then the edge can be represented by l.

1.5 Pairwise Geometric Histogram

In image processing, we have many techniques to describe the features of the image. With expect represent the image's features into the variant information for compare and representation, pairwise geometric histogram (PGH) method is chosen. PGH is constructed based on the geometry relative of the image's features. With an image is presented by the list of lines, the angle and perpendicular distance between two lines are importance characteristic to consider for re-constructor the image. Moreover, PGH is also fitwell when we apply some variants on the image such as translation or rotation because the angle and perpendicular distance between two lines are invariant. The process to calculate the PGH of an image which is represented by a list of lines as follows:

- Choose arbitrary line as reference lines l_f
- For each other lines l in image, calculate the angle between l and l_f, and perpendicular distance from two endpoints of l to l_f.
- The process will stop when all line in the image are considered as reference line.

An importance note, during the process calculate the PGH, we need to keep the information of PGH to reconstruct the image or compare with other image.



(a) The geometric relationship between two lines

(b) The information of PGH is kept into a matrix

Figure 1.2: Example about geometric features and the pairwise geometric histogram

To detect the similarity between two images, we can use the pairwise geometric histogram

matching via the similar distance of their probability distribution on histogram. The Bhattacharyya metric is common distance to compare two models. The form of Bhattacharyya is:

$$d_{Bhattacharyya}(H_i H_j) = \sum_{\theta}^{\pi} \sum_{d}^{d_{max}} \sqrt{H_i(\theta, d) H_j(\theta, d)} \tag{1.11}$$

The significance of parameters in the formula 1.11, as follows:

- θ: angle value, range of θ in angle axis from 0 to π.
- d: the perpendicular distance, range of d in perpendicular distance from 0 to the maximum distance of arbitrary lines of shape.
- $H_i(\theta, d)$ is an entry at row θ and column d in PGH of image i
- $H_j(\theta, d)$ is an entry at row θ and column d in PGH of image j

1.6 Bounding box detection

In this section, we will discuss about the methods to detect the bounding box of the object. Besides, we also mention about the projection histogram technique which used to detect the distribution of pixels in many ways and applying the projection histogram for detecting the bounding box around object in image.

1.6.1 Bounding box detection

The bounding box of an object is a rectangle surround the object and detecting the bounding box of object is finding the smallest box that can contain the object. Figure 1.3 show an example about the bounding box of a mandible of beetle. In this example, the mandible has two

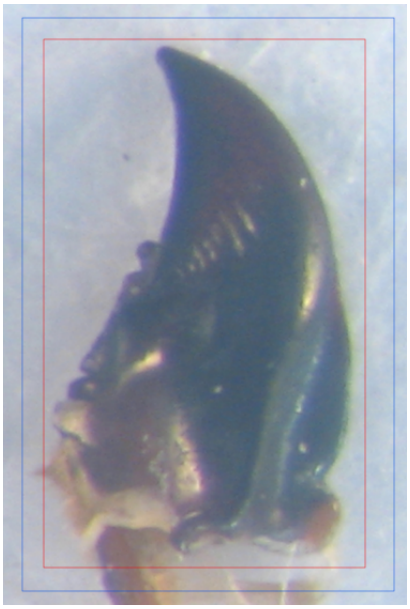


Figure 1.3: An example about projection histograms

bounding boxes, one is box with red color, another with blue color. In the context of detecting the bounding box, hence we want to indicate the smallest bounding of the object. Mostafa S. Ibrahim et al [5] has proposed a method based on image superpixelization. The image superpixels are used to determine the small subset of candidate sub-windows to have the object of

interest. Then, they used the *OLIS-BF* algorithm and *OLIS-ESS* algorithm to represent the most confident location to have the object. (**...need to talk more about the authors on this field**).

1.6.2 Histgoram projection

Projection histogram is one of the techniques that used to extract the feature of the image. The information of projection histogram was extracted from the distribution of the pixels on an arbitrary axis. In mathematics, a projection is a mapping of a set into a subset which is correspondence with its square for mapping composition.

To find the projection histograms of a color image; we need to process on each channel color of the image and combine the result at the end of process. With a channel of image, we can create a lot of projection histogram i.e projection on x-axis, y-axis. The number of the projection and the combining the projections are is depended on the goal of the works; the projection histogram of the image is calculated on many difference axes such as y=x, y=-x,.... The purpose of this work is increasing the accuracy of the method to find exactly the feature of the image. Figure 1.4 decribes the x-axis projection and y-axis projection of an objection in the image.

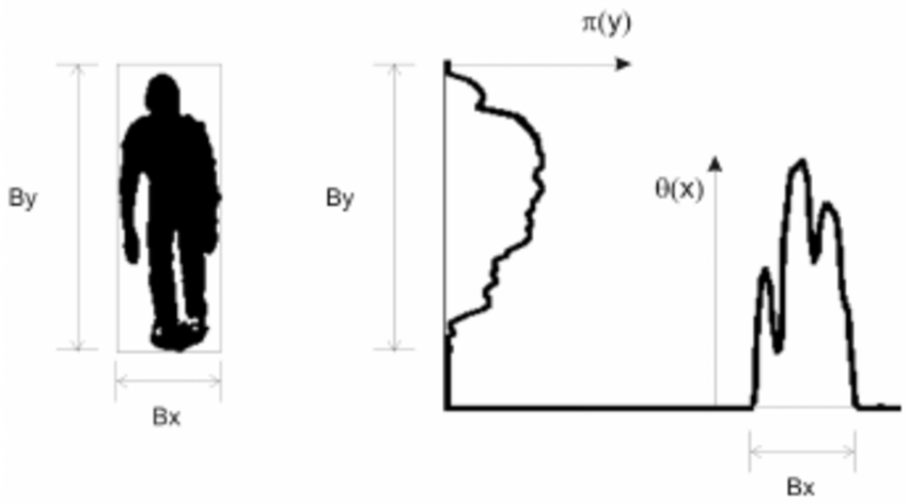


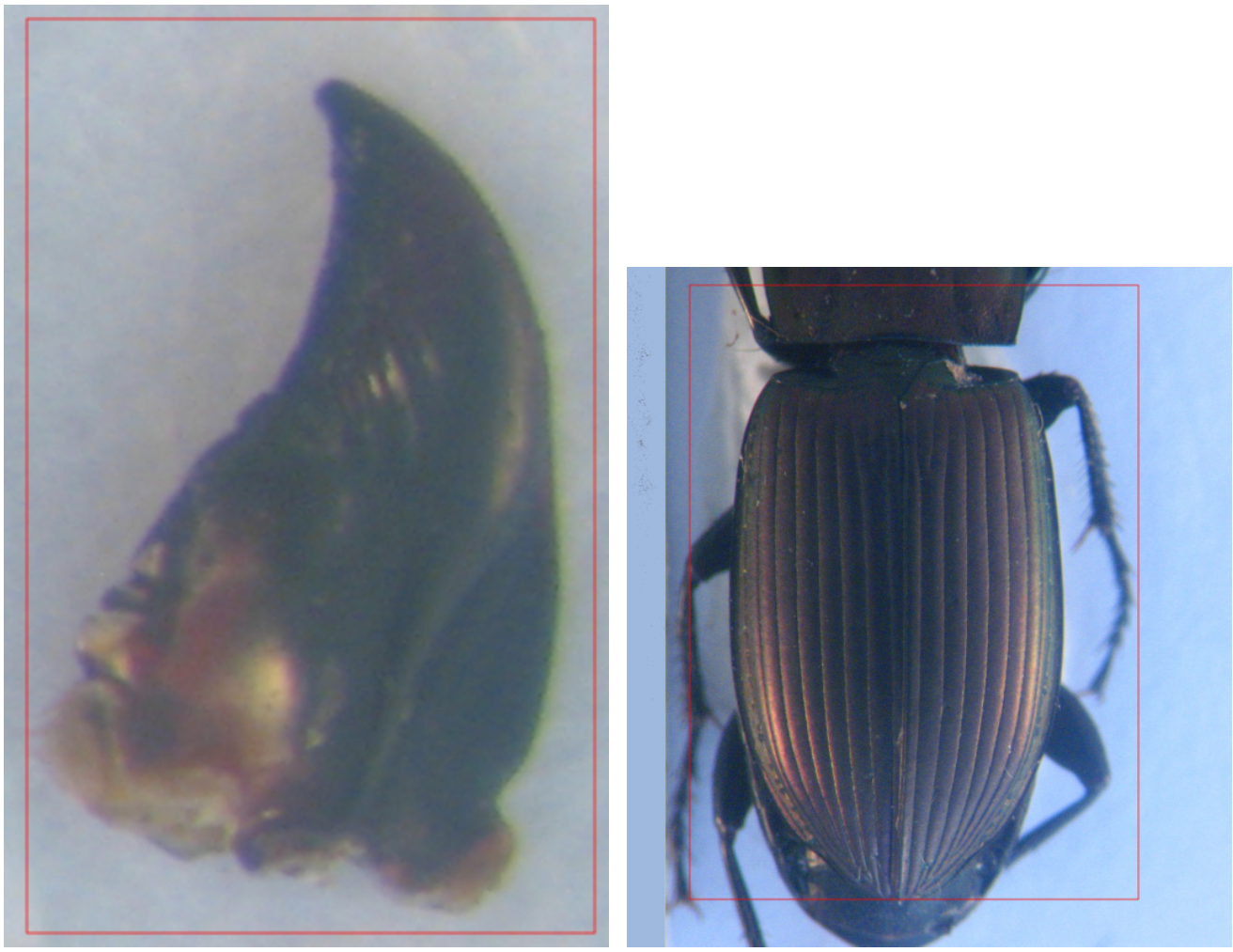
Figure 1.4: An example about projection histograms

The method that we use to detect the bounding box is based on the projection histogram of the image by applying the analysis on projection histograms. Actually, to determine the position of bounding box that we just need to indicate the position of two (or four) corners ((top-left, bottom-right)or (top-left, top-right, bottom-left, bottom-right) of the box.

The information of the corners was determined by combining and analysing the horizontal and vertical projection of the image: Firstly, the image is quantized into N color to decrease the number of color on the image. Secondly, the horizontal and vertical histogram projection are calculated on the width and height of the image. Finally, finding the lower and upper limit of histogram around the the peak value. The corners of the bounding box are indicated by intersecting of the limit values.

1.6.3 Result

The experiment is done on the set of the mandibles and body part of beetle. The bounding box on mandibles is detected clearly and more exactly than the box on body of beetle. The difference of the results are coming from the analysis the projections.



(a) Bounding box on madible of beetle

(b) Bounding box on body of beetle

Figure 1.5: The result of bounding box on beetle

1.6.4 Bounding box on Pronotum

The method that used to detect bounding box on pronotum includes 3 main steps:

1. Threshold the image (binary threshold) and fill the holes inside the object.
2. Detecting the contours points of the object.
3. Bounding box detection

Firstly, image was thresholded by applying binary threshold. The threshold value was detected by analysing the histogram of image. Then, holes which inside the object are filled by using the "line-based" method(section 1.1.3).

Secondly, the contours points of the object are indicated as following:

1. A point insides the object is chosen, called *origin*.
2. Find the intersection between horizontal, vertical line via *origin* and the background.
3. Finding the median point for each pairs of intersection points.
4. Finding the intersection between median line (line via origin point and median point) and the background.
5. The method will be stopped after a number of iteration which setted by user.
6. The intersection points are the contours points of the object.

Finally, four locations of bounding box are detected by analysing the contours points.

1. Dividing the contours points into two groups. The first group stays on the left of origin point, remaining group stays on the right of origin point.
2. For each group, two corners ((top-left, bottom-left) or (top-right, bottom-right)) of bounding box are detected:
   (a) The points are sorted following y-direction. Then, the point that has the maximum distance with vertical line (when we compute the contours points) is detected.
   (b) Next, the points are continue split into two groups. The first group begins from beginning point to the maximum point, and the second group is remaining points. The minimum point is determined in each group (the point that has the minimum distance to vertical line).
   (c) The coordinates of the bounding box are combined from the maximum and minimum points of each group. The value of x-coordinate is the x-coordinate of maximum point. The y-coordinates are the y-coordinate of minimum points.

Result

1.7 Texture segmentation

Texture is an important characteristic for the analysis the image. It is widely used to segment the image into regions and classify these regions. In another field, texture can be defining the characteristic of regions and critical in calculating the correct information. In intensity information side, texture provides for us the intensities of the image by the information of the pixels on the image. Texture segmentation is the methods which used to detect the border of the regions in the image. Those methods can be classified into three classes: *region-based* includes the methods that detect the regions in the image; *boundary-based* includes the methods that detect the border between the regions; and *a hybrid* of two classes. Even using any methods, the textures firstly must be characterize the distribution of the pixels on the image. This section will present the methods to describe and segment the texture.

1.7.1 Texture description

Texel-based texture descriptions

A texture can be seen as a set of primitive texel in a particular relationship. The structural description of texture will include the description of the texel and the relationship between the texels. The texels are regions that can be extracted by applying some simple procedure. The

information of spatial relationships is obtained from a Voronoi tesselation of texels.

Assume that we have a set of extracted texels and **S** is the set of meaningful points(i.e centroid) that each point is represented for texels of the image. With two arbitrary points in **S**, we construct the perpendicular bisector of the line joining them. The perpendicular bisector will divide **S** into two parts(P,Q), one of which is closest with the first point(P), other of which is closest with the second point (Q). The *Voronoi polygon* of P is the polygonal region consisting of all points that closer to P than to any other points of S. It is defined by:

$$V(P) = \bigcap_{Q \in S, Q \neq P} H^Q(P) \tag{1.12}$$

Where: $H^Q(P)$ is the part of points that closer to P with perpendicular bisector of P and Q. The texel based description is easy to understand and implementation, but it will be difficult when we apply for the real image instead of artificially general patterns.

Local binary pattern

Another method to describe texture is local binary pattern. For each pixels in the image, 8-connected pixels are examined by comparing the intensity of each pixel with the intensity of the center pixel. The result of this step is compute the digit matrix of 8-connected pixels; the corresponding will be 1 if the value of pixel $p_i(i = 1..8)$ is greater than value of center pixel and 0 if otherwise. The process is repeated for all pixels of the image. At the end, a histogram of these numbers is used to present the texture of the region. Two regions of the image are compared by using L1 distance between their histogram. Besides, LBP is also combined with the contrast of the regions in many texture segmentation methods. Figure 1.6 shows an example

7	6	5
7	7	4
7	9	8

1	0	0
1		0
1	1	1

Figure 1.6: Computation of local binary pattern

of LBP computation. The left matrix is a pixel in image and its neighbors; the right matrix is digit matrix when we compare the value of the neighbors with the center value.

1.7.2 Texture segmentation methods

Voronoi polygons

Timo's method

Chapter 2

Local features and landmark estimation

2.1 Local features

2.1.1 Image gradient

The simplest feature can be used as the local feature is image gradient. It is computed directly from the intensity or color of the image. Because the image is composed of the discrete pixels, so the gradient of a pixel can be computed as approximation of its value and the neighbourhood value. The common way to calculate the image gradient is convolve the image with an kernel, such as Gaussian operator [6], Robert operator [6], Sobel operator [6] or Prewitt operator[6]. The gradient feature have been used to detect the contour[1], corner [7], point of interest [8].

When talk about the gradient of the image, we usually ear to the magnitude and direction of the gradient which given by the formula:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix}, \tag{2.1}$$

$$\theta = tan^{-1}(\frac{g_y}{g_x}) \tag{2.2}$$

Where:

- g_x: is the gradient in x direction
- g_y: is the gradient in y direction
- ∇f: is the gradient magnitude
- θ: is the gradient direction

2.1.2 Local binary pattern

Local Binary Pattern (LBP) is a model of texture analysis based on the texture unit (a small patch of the image). It is introduced the first time by Wang and He [9] with 6561 possible texture units in a 3×3 patch. Timo Ojala et al [10] proposed another level (two-level) for the texture descriptor, there are only $2^8 = 256$ possible texture units instead of 6561. In the binary case, the elements in 3×3 patch are thresholded by the value of center pixel. A weight patch corresponding with the patch is created. The value at each position in weight patch is exponential of two following the clockwise direction or counter-clockwise direction. The value of the pixels in the thresholded patch are multiplied with the corresponding pixels in weight patch. Finally, the value of eight pixels are summed to obtain the LBP value of the texture unit. The LBP method is a gray-scale invariant and it can be easily combined with other features to use in the application of texture.

2.1.3 Contrast

Contrast is the difference in luminance or color that make the object distinguishable. Depend on the situations of using, the definitions of contrast are defined in different.
Weber [11] defines contrast as (equation 2.3):

$$C = \frac{I - I_b}{I_b} \tag{2.3}$$

Where:

- C: is the contrast of the image,
- I, I_b: are luminance of the features and the background, respectively.

Michelson [12] use the highest (I_{max}) and lowest (I_{min}) luminance to calculate the contrast (equation 2.4). This calculation is commonly used for patterns.

$$C = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} \tag{2.4}$$

The Root Mean Square (RMS) [13] contrast is defined as the standard deviation of pixel intensities:

$$C = \sqrt{\frac{1}{MN} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (I_{ij} - \overline{I})^2} \tag{2.5}$$

Where:

- C: is the contrast of the image,
- M, N: are the size of the image
- I_{ij}: is intensity at position (i, j).
- $\overline{I}$: is the average intensity of all pixel values in the image

2.1.4 Center-symmetric covariance measures

SCOV is a measure of the pattern correlation. It was introduced by David Harwood et al [**?**]. It measures covariance of any local center-symmetric pattern. The measure is abstract measures of texture pattern and grey-scale, providing the discriminating information about the amount of local texture. The SCOV of a 3×3 neighbourhood (see fig 2.1) for center-symmetric pairs of the pixels are calculated by the equation (2.6):

g_2	g_3	g_4
g_1		g'_1
g'_4	g'_3	g'_2

Figure 2.1: A 3×3 neighborhood for center-symmetric pairs.

$$SCOV = \frac{1}{4} \sum_{i}^{4} (g_i - \mu)(g'_i - \mu) \tag{2.6}$$

Where:

- g_i: is grey-level of pixel i
- g'_i: is the grey-level of pixel that symmetric with pixel i
- μ: is the local mean

2.2 Features integration

In most case, a single texture feature is not enough information to present for amount and spatial of local texture. The better way is considering the combination between two or more features. As an example, Lowe [8] considered the use of gradient magnitude and direction to detect the dominant point in the image; Timo Ojala et al [10, 14] used LBP/Contrast and LBP/SCOV to classify the texture in the image.

2.3 Local features and landmark estimation

In this section, we show a way to combine the local features and evaluate the correctness of them for estimating the landmark on beetle's pronotum. Local binary pattern (LBP) and contrast(C) have been selected.
By using two images (source and target image) and their manual landmarks, method shows the way to evaluate the landmarks on the others. It includes two steps: (1) create the texture descriptor; (2) compare the similarity among the descriptors. Each landmark of the source image will be evaluated with all the landmarks of the target image.

For each manual landmark, a patch centered at the landmark is created with size s. Then, for each pixel in the patch, a 3×3 neighborhood (Fig. 2.2a) is used to calculate the LBP and contrast. The original 3×3 patch is thresholded by the value of the center pixel (Fig. 2.2b). The values of the pixels in the thresholded neighborhood are multiplied by the binomial weights given to the corresponding pixels(Fig. 2.2c). The LBP of texture unit is summed of all obtained values (Fig. 2.2d). The contrast of the texture unit is defined as the difference between the average gray-level of the 1-pixels and 0-pixels. The contrast then is mapped from the continuous value to the discrete value by quantization. The pair LBP/C after that is presented into a two-dimensional accumulator of size $256 \times b$, where b is number of discrete contrast. The process is continued until all pixels in the patch are considered.
The comparing between the descriptors is done by using $L2$ distance.

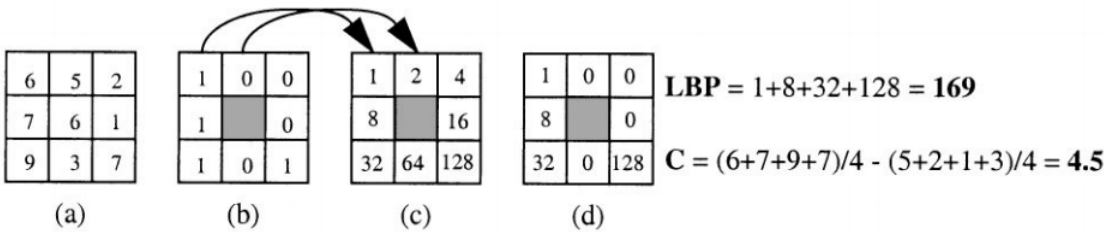


Figure 2.2: LBP and contrast (C) computing.

2.4 Result

Timo Ojala et al [14] have proposed a method to segment the texture using LBP/C. The evaluation is done on Mosaic #2 (a 512×512 image containing four texture made by a GMRF

process). The segmentation error is 4.2% error for the first sweep, it is quite decent. The final error is 1.2% (after 23 sweeps); Mosaic #3 (a 512×512 image with a background made by a GMRF process and four distinct regions) with final error is 1.9% after 13 sweeps; Mosaic #4, #5 are composed of textures taken from outdoor scene. The segmentation errors are 3.3% and 2.1%, respectively. From this result, the combining between LBP and contrast could be a good pair for the classification application. An advantage of this method that it does not require any knowledge about how many texture or regions in the image.

In the context of combining the local features, we apply LBP/C to evaluate the effect of this feature pair on the texture around the landmark (as described before). The result shows that the distance of pair at landmark $3^{rd}, 4^{th}, 6^{th}, 7^{th}$ is better than other ones. This mean that the pair LBP/C is worked well on the pattern which have the distinctable pixels.

In another examine, the scene image is segmented by applying Canny algorithm. Then, a patch around each contour point is created. The descriptor is computed and compared with the descriptor of the manual landmark of the model. The process keep the point of the contour that has minimum distance with the manual descriptor. Following this way, the evaluation is done on all manual landmarks of model. The result of examination shows that this method can estimate the landmark at the position $3^{rd}, 4^{th}$, and in some case of landmark $6^{th}, 7^{th}$.

2.5 Conclusion

In this works, we have studied the local features and their combination. We have applied the LBP/C to examine the sensitive of this pair with on the gray-scale of pronotum image. In the next, we will try to evaluate the other pair of local features on pronotum such as LBP/SCOV.

Chapter 3

Dominant points

In shape analysis, extracting features from the curves is an important step because in another way, we can re-construct the shape from the features. The term dominant points, also called as siginficant points, points of interest, corner points or landmarks is assigned to the points which have the high effect on boundary of object; their dectection is a very important aspect in contours methods because these concentrate the information of a curve on the shape.

Dominant points can be used to produce a presentation of a shape contour for futher processing. The representation ... In the content of this chapter, we will discuss about the methods to determine the dominant in digital image.

There are many approaches developed for detecting dominant points and the methods can be classified into three groups follows:

- Dectermine the dominant points using some significat measure other than curvature
- Evaluate the curvature by transforming the contour to the Gaussian scale space.
- Search for dominant points by estimating directly the curvature in the original image space.

3.1 Hough Transform

One of the challenges in image processing is detecting the characteristic of the object for recognition. Shape recognition is done by searching or detecting a class of simple geometric object such as line, curves in the image and comparing with the model. The matching score of the shapes is calculating by a measurement distance (such as Bhattacharyya). Instead of comparing between the geometric classes from the shapes, we can detect the presence of a shape in anther shape by searching each feature of the shapes. To sovle this problem, Hough Transform (HT)[15] is varied. At the beginning, HT[16] is used to detect the line. It converts the space of parameters from x and y (coordinate of points in line) to space of slope and y-intercept of the line by voting process. For each object statisfying with equation of a line, it votes for the bin have correspondence slope and y-intercept. The set of bins is called the accumulator.

3.1.1 Generalizing Hough Transform

Until now, HT is still a good method for line detection or object recognition. But one of the weakness of HT is cannot determine the end points of the line segments. For this reason, the Generalized Hough Transform (GHT), introduced by Ballard[17] is a generalization of HT to

detect non-parametric curves. The process includes two phases: learning and recognition. In learning phase, a R-table is construct for model object. R-table is constructed based on the geometric information of each points in curves of object model with a reference point. The reference point can be arbitrary point in the model. Each row in R-table includes the gradient direction of each point which was chosen as index of table; and the polar coordinate values of each point. This mean that a gradient direction can be having many polar coordinate values. During recognition phase, an accumulator is created, called Hough Space. For each point in the scene object, finding the correspondce gradient direction in the R-table and voting at all the coordinate values. The peak in accumnulator is position of reference point of the model object in the scene object. And the peak value is equal to the number of boundary points of the object when the model and the scene match perfectly.

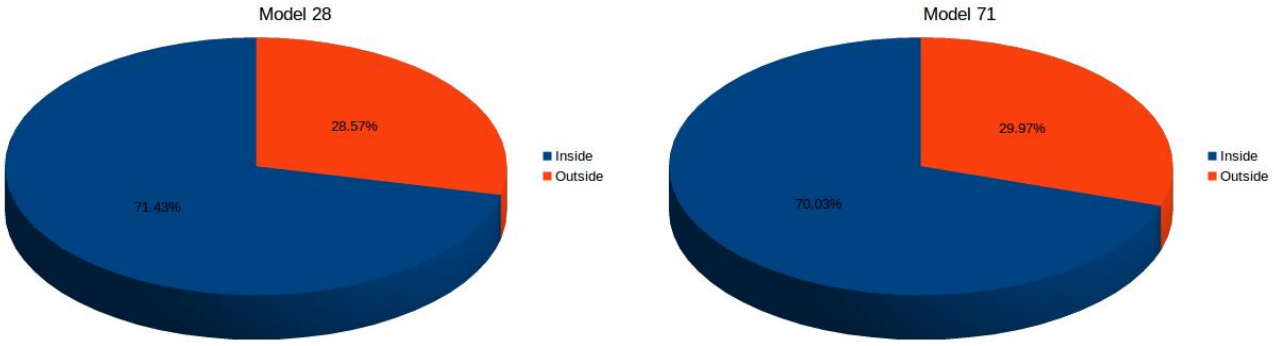
During recognition phase, the translation and rotation between model object and scene object is determine by principal component axis. Based on the curve points of the object, the centroid of each object is calculated. Then, the principal axis of each object is indicated. The translation between two objects is difference of two centroid points. The angle to rotate is the angle difference of two axes.

When model and scene are matching, the dominant points (landmarks) of scene object is estimated from landmarks of model object by applying the translation, rotation from the centroid points. The last result is verifying by apply template matching (which will discuss as section 3.2).

Result

Using GHT to extract the landmarks on beetle is experiment on 287 images of right mandible of beetle. To compare the matching between the location of manual landmarks and estimated landmarks, the centroid size is compute for each set of landmarks.

Figure 3.1 display the accuracy of the centroid size of estimated landmarks when we compare



(a) Model 28 (b) Model 71

Figure 3.1: The accuracy of centroid size on mandible

with the centroid size of manual landmarks. We use 2 images (Md28.JPG and Md71.JPG) as model. The number of landmarks be detected on scene object is 100%. For model 28, 71.43% the centroid of estimated landmarks is placed inside the standard deviation (SD) of manual landmarks, 28.57% is outside the SD. The ratio for model 71 are 70.03% and 29.97%.

3.1.2 Probabilistic Hough Transform

To speed up Hough Transform, instead of processing on all data set, we can consider a subset of data points. The popular method is **Probabilistic Hough Transform**. Probabilistic Hough

Transform (PHT) is used to detect the presence of a model image in a scene image based on the group of features. The hypothesised location of the model image in the scene image is indicated based on the conditional probability that any pair scene lines agreement about a position in model image. Applying PHT can be separated into two steps: firstly, recording the information of model image and try to find the presence of the model image in scene image (called training process); secondly, predicting the pose of model image in the scene image (called estimating process).

During training process, choose an arbitrary point in the model image, called reference point. For each pair of lines in model image, the perpendicular distance and angle from each line to reference point is recording (angle is calculated as angle between line and a horizontal line begin from reference point). The presence of model image in scene image is detected by PHT with *"vote"* procedure. Finally, we choose the similar pair lines between model image and scene image. The chosen pair is obtained from best *vote* when we consider each pair of line in scene image with each pair of lines in model image.

In estimating process, the reference point in model image is estimated in scene image by extending the perpendicular lines of the pair of scene lines at the appropriate position. There, we can estimate the pose of the model in the scene image.

3.2 Template matching

Template matching is a technique for finding areas of an image that match to a template image (template) by sliding the template over each pixel on the image (commonly cross-correlation). At each position, the sum of products between two images is calculated. The position is considered similar if the sum value at this position is maximal. The equation of cross-correlation is as follows:

$$R_{ccorr}(x,y) = \sum_{x',y'}[T(x'.y').I(x+x',y+y')] \tag{3.1}$$

Where:

- T is template which use to slide and find the exist in other image.
- I is image which we expect to find the template image
- (x', y') are coordinates in template where we get the value to compute.
- $(x+x', y+y')$ are coordinates in image where we get the value to compute when template T sliding.

However, if we use the original image to compute and find the similarity, the brightness of the template and the image might change the conditions and the result. So, we can normalize the image before applying the cross-correlation to reduce the effect of lighting difference between them. The normalization coefficient is:

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x'.y')^2 . \sum_{x',y'} I(x+x',y+y')^2} \tag{3.2}$$

The value of this method when we normalized computation as below:

$$R_{ccorr\_norm}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)} = \frac{\sum\limits_{x',y'}[T(x'.y').I(x+x',y+y')]}{\sqrt{\sum\limits_{x',y'}T(x'.y')^2.\sum\limits_{x',y'}I(x+x',y+y')^2}} \tag{3.3}$$

3.3 Image registration

Image registration is process of transforming difference data sets into the same space and comparing or integrating the data from them. The object in image registration may be the images, time series or viewpoints. It is having many application in medical, military or satellites. In recent years, image registration is applied for both 2D and 3D objects with many methods. These methods may be classified following the characteristics of the input such as *intensity-based and feature-based*, **transformation**, *spatial and frequency*,... In the context of this section, we want to discuss around the methods of linear transformations which include rotation, translation and scaling. Besides, we use these method to generate the general model from several objects or detect the landmarks on the object.

3.3.1 Principal component analysis (PCA)

Principal component analysis is computed based on principal directions of the datasets (model and scene). The input of this method is the list of points on curves of model and scene object (called model points and object points). The origin of the axes is centroid of all points on the curves. One of the axes is the line over the origin and having the minimum distance to all points in the curves; another axis is perpendicular axis with the first axis. The translation between two objects is different distance of centroid point coordinates; the rotation is different angle of two coordinate systems. The steps in PCA are followed:

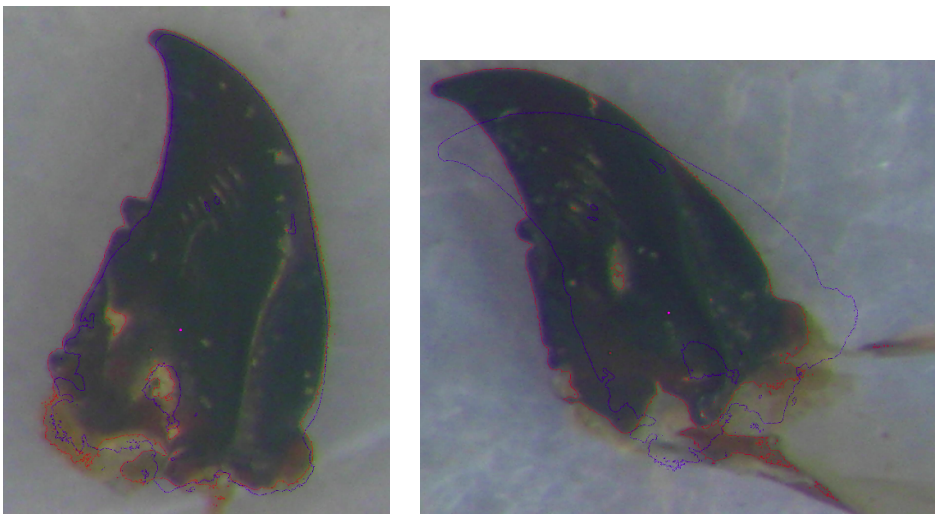
- Compute the centroid of model and scene object,
- Calculate the principal axes of model and scene,
- Compute the translation and rotation
- Translate the model to the scene that they have the same centroid.
- Rotate the model followed the different angle to match with the scene.

Result

The method is experiment with the set of right mandibles. Most of model can be detected its position on the scene by PCA. It also determine the translation and rotation information (see figure 3.2a). But in the case the input has more the noises, the centroid may be missed with correct position, following it is wrong translation and rotation(see figure 3.2b). In these examples, the red line is presented for the scene object and blue points is presented for the model object, which we want to align with the scene.

3.3.2 PCA Iteration (PCAI)

PCA method is a simplest method to align two images. However, PCA is more influenced by noises(see figure 3.2). Based on the idea of PCA method, PCAI try to apply the PCA on the interested data of the input.



(a) PCA with less noises (b) PCA with noises

Figure 3.2: The result after applying the PCA

The solved problem in PCAI is the same with other difference registration methods. With two set of data input, specify curve points, we want to register two images with best matches. In PCAI method, firstly, PCA is applied on all two set of data for having the first sight of data. Secondly, the data is sorted followed one of coordinates of data points. The interested data is taken up with a half of data points which are sorted. Thirdly, an iteration will be executed to match the data. For each iteration, we re-compute the principal component of scene data and compare with the model. The iteration will be terminated when the difference position between two images is smallest.

Beside applying PCAI to make the images are matched. PCAI can combine with other technique to estimate the landmarks, such as template matching. At beginning, PCAI is applied to match the images. At the end, for each manual landmarks on the model, we apply the template matching to estimate the location of landmarks on scene image.

Result

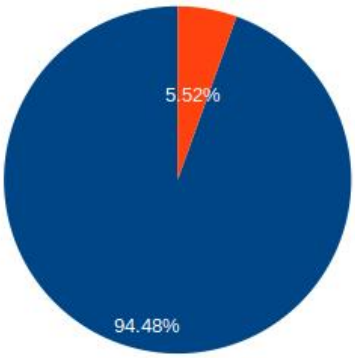
The combining between PCAI and template matching is experimented on two datasets of mandible (left and right mandible). For each set of data, it was divided into two sub-sets followed the size of objects. The model is chosen for each sub-set. The estimated landmarks coordinates are used to compute the centroid size of the mandibles. The results are compared with the centroid size of manual landmarks. In general, the success rate of the method is **90.56%** for left mandible and **94.48%** for right mandible (figure 3.3).

In a different side, if we have statistics on each set of mandible, the success rates of left mandible are *87.18%* for sub-set 1 and *97.80%* for sub-set 2 (figure 3.4). For right mandible are *94.52%* and *94.37%* for sub-set 1 and sub-set 2, respective (figure 3.5).

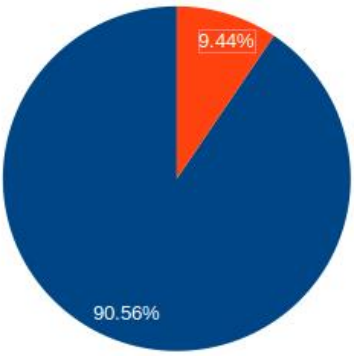
Although the result from the method is good but it depends much on the result of segmentation. If the segmentation is not good and have many noises, the result of the method will be affected.

3.3.3 Singular value decomposition (SVD)

The PCA method is more effected by the noise, instead of using all the curve points, SVD just using a subset of points by optimal alignment between corresponding points of model and

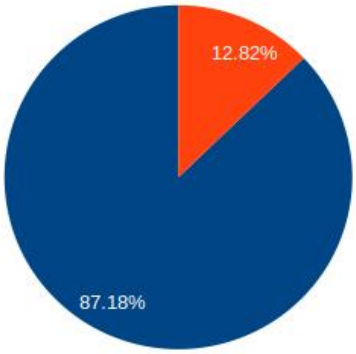


(a) Right mandibles

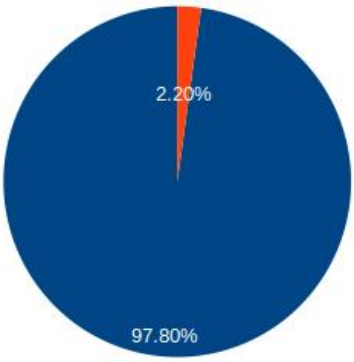


(b) Left mandibles

Figure 3.3: Success rates of estimating landmarks on mandibles

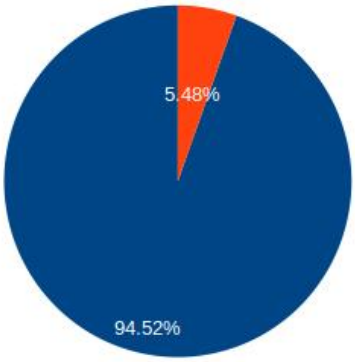


(a) Sub-set 1

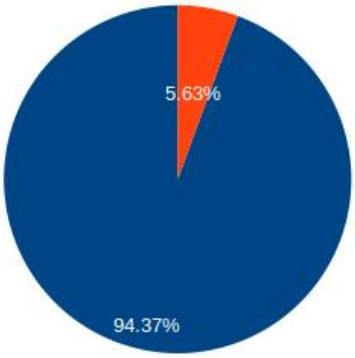


(b) Sub-set 2

Figure 3.4: Success rates of estimating landmarks on each sub-set of left mandibles



(a) Sub-set 1



(b) Sub-set 2

Figure 3.5: Success rates of estimating landmarks on each sub-set of right mandibles

scene. Assume that `M` is a subset of the model points, **S** is a subset of the scene points and $p_i \in M, q_i \in S$ are two corresponding points. We would like to find the matrix transformed **R** so that the pair-wise distances between the corresesponding points is minimum. The pairwise distance is indicated by equation (3.4).

$$E = \sum_{i=1}^{n} \|q_i - p_i^{'}\| \tag{3.4}$$

Where:

- n: is number of corresponding points
- q_i: point of scene
- $p_i^{'}$: point of model which corresponding with q_i

In detail, SVD method includes the following steps:

- Calculate the cross covariance matrix: $M = P.Q^T$, where $P(Q)$ are matrix with i-th column is vector $p_i - c_T$ $(q_i - c_S)$,
- Compute the singular value decomposition of matrix M: $M = U.W.V^T$. Where:
  - U,V are m x m orthonormal matrices
  - W is a diagonal m x m matrix with non-negative entries.
- Indicate the orthonormal matrix (rotation matrix) $R = V.U^T$

Result

SVD method is solving the noise problem of PCA by using a set of corresponding points. The result obtained by applying the SVD also better PCA. But a disadvantage of SVD is requiring a accurate correspondences set of points which are usually not available.

3.3.4 Iterative closest point (ICP)

Based on the advantage and disadvantage of PCA and SVD. ICP combines two previous methods. The idea of ICP is using PCA to intial guess of correspondences and repeating SVD to improve correspondences. The steps of ICP are:

- Transform the model by PCA aligment
- For each transformed model point, assign the closest scene point as its corresponding point. Align model and scene by SVD
- Repeat the step (2) until a termination criteria is met.

3.4 Patch-based

In recent year, beyond using the pixels to extract and compare the features of the images, patch-based is more and more catch on because the advantages of processing time and less memory use of program. The behind idea of this technique is extracting all patches (with(out) overlap) on the input image. The size of the patches are fixed and smaller than the size of the image. Most of the methods with patch-based spend more time to extract and compare the relation between them. In this section, we will mention to comparing methods between patches; next, some processing methods on patches are introduced; finally, we want to present some application based on patch-base. During this section, we will use some notations as follows: **Z, X** are source and target image; **M, N** are two patches on Z and X with the size *nxn*.

3.4.1 Comparing method between patches

In this part, we will mention to the methods which are used to measure distance between two patches M and N.

Pixel-based distance(L1 distance)

The similarity measure between patches is determined by sum of the distance between each pair of pixels in the patches.

$$d(M, N) = \sum_{d}(M_d - N_d) \tag{3.5}$$

Where: d is the pixels on patches.

However, L1 distance (equation 3.5) is very sensitive with the changing of pixels such as rotation, translation or scale. Instead of using the L1 distance, we can use another norm (L2 distance) on the pixels as:

$$d(M, N) = \sum_{d}(M_d - N_d)^2 \tag{3.6}$$

Correlation

$$d(M, N) = \frac{\sum_d (M_d - \overline{M_d})(N_d - \overline{N_d})}{\sigma_1 . \sigma_2} \tag{3.7}$$

Where: $\overline{x_i}, \sigma_i$ is the mean and standard deviation of the pixels on patch x_i.

Descriptor distance

In this way, the characteristic of the patch is presented into other dimension spatial (i.e vector). Then, the measure distance between two patches will be moved to distance between two descriptors.

Probabilistic matching

In this case, we do not measure the distance between patches directly. Instead, we calculate the probability to the patches **M, N** is belong to the same group.

3.4.2 Methods on patches

This section, we will investigates recent articles that follow these themes. Most of patch-based methods can be divided into three groups: in the first group, the feature on patches is presented into other dimensional and comparing [18, 19, 20, 21]; the second group includes the methods that directly compare the pixels in the patches [22, 23, 24]; the last group is used dictionary to compare the patches[25].

The first group, we have to say about the *kd-tree* (k is the number of dimension in tree). This method is introduced by [18]. Bentley used the binary tree to present the relationship between the records in a file. Firstly, attributes of the record is presented to other data in other dimensional (vector). Then, the datas is used to construct a binary tree. And patch matching will be done in this tree. In this method, we can spend more time and cost to build the tree but we can reduce the cost during searching a patch. Based on the idea of kd-tree, Sproull and Robert propose PCA-tree [19]. The idea of this medthod is reduce the number of dimension of data by using eigenvector and eigenvalue of patches. Besides, we have also the methods based on the tree as TSVQ [20] or vp-tree[21].

In the scheme of methods which based on comparing of pixels in the patches. Connelly[23] was given the concepts of Nearest Neighbor Field (NNF) to compare the distance between two patches by using a function $f : S \to R^2$. This method includes 2 steps: (1) the matching patches is randomly initialization with NNF; (2) an iteration of propagation and randomization

is applied to determine the best patch match for searching patch. In a different side, instead finding the best matching of a patch, we would like to have a set of patches that matching with the searching patch[22]. In this method, they try to extend on patch matching include k-nearest neighbors, the matching is done through rotation, scale and matching descriptor. They also mention a new search strategy, called "enrichment" and parallel searching based on multiple core of GPU. Besides, Xiao[24] propose a method that the authors evalutate this is a fast method. Xiao's method refers to move and calculate the distance between the patches. It was also improved by reducing the complexity of the calculating distance of patches.

The last group on patch-based is used dictionary such as Locality Sensitive Hashing (LSH)[25]. Firstly, the dimension of each patch is reduced and stored into **PatchTable**. Follow with this way, the similar patches will be stored into a record in PatchTable. Patch-matching is finished by checking each cell in PatchTable with searching patch to indicate the best matching location.

3.4.3 Application on patches

Patch-based is widely used in 2D, 3D images and video application [26]. This section will introduce some domain in 2D image where we can apply the patch-based.

Inpainting and reshuffling

Image inpainting [27] image is technique that removes a region (foreground) in the image by replacing it with other region(background) in the image. The backround can be extract from this image or from another image. Meanwhile, reshuffling [28] is moving a region to new postion in the image. The aim of this work is create a new image consistent with the user constraints. Image reshuffling can be treated as an extension of image inpainting by initializing the regions to be synthesized by user specified contents.

Denoising

Image denoising[22] is popular application of patch-based (i.e Gaussian filter, Sobel filter). A patch is moved over the pixels of image. At each position, a weighted average is calculated to denoise the pixels.

Chapter 4

Software

The architecture of program is followed 3-tier model. There-tier architecture is an architecture that each tier is designed, developed and maintained as independent. The advantage of this architecture is intended to allow any upgraded or replaced independent between the tiers. When user want to change the requirements or technology of a tier, it will non-affect to other tiers.

The architecture of three-tiers includes:

- **Data tier**: includes the classes which were designed for the data structure of program. It also provides the persistence mechanism to access the data.
- **Model tier**: controls the functionality of application by performing detailed processing.
- **Presentation tier**: displays information related to user. It is a layer which received the require from user to program or return the result from program to user.

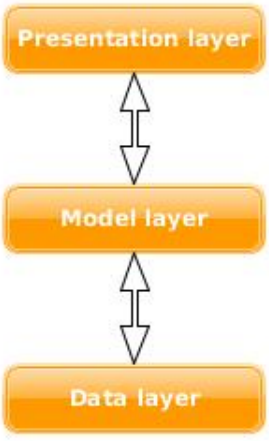


Figure 4.1: Three-tiers model

4.1 The design of the software

The MAELab software mainly includes four modules: **segmentation**, **histograms**, **pht** and **correlation**. Besides, the software also includes the other modules to support for the main modules. The relation between the modules in the software is shown in figure 4.2. The functions of each modules is describing as followed:

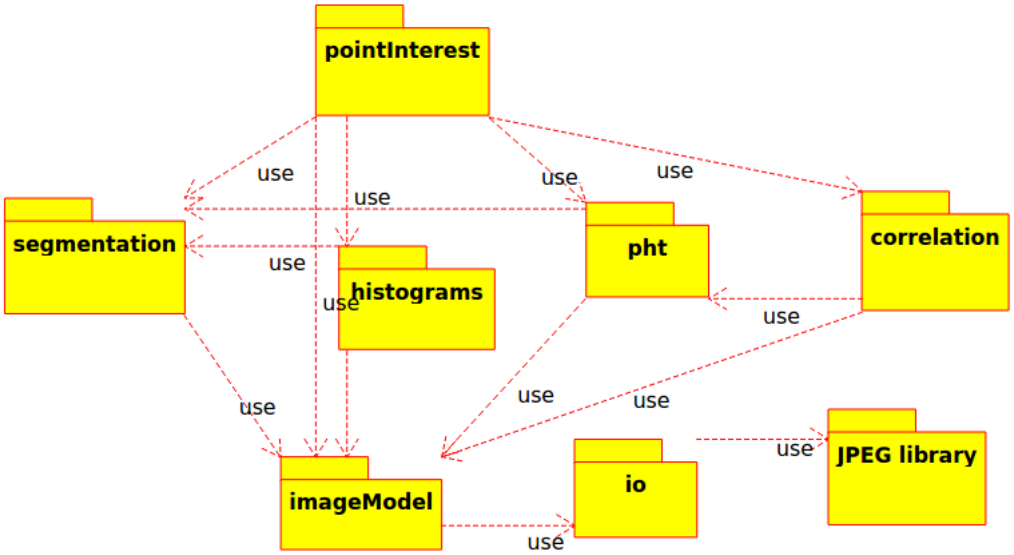


Figure 4.2: Three-tiers model

- **io** module: Implement the functions to read and write file. It includes the **JPEG library** that used to decode and encode the JPEG image.
- **imageModel** module: Represent the data structure of the image.
- **segmentation** module: Implement the segmentation methods on image.
- **histograms** module: Contains the methods to compute the geometric histogram of the image.
- **pht** module: Describe the probabilistic hough transform duration.
- **correlation** module: Includes the template matching methods.
- **pointInterest** module: Combine the result of the modules such as segementation, histograms,... to provide the adapter to other module or other software.

4.2 The classes architecture

Figure 4.3 describes the classes diagram of the software. Based on structurally software, the classes is divided into two parts: one, describing for the data structure and another, describing the functions of software. The data structure classes are the classes that used to represent the structure of the image in program. These classes are using through all the functions of the software.

- `Point` class describes a point in mathematics, its attributes include the coordinate of the point in Cartesian coordinate system.
- `Line` class describes for a straight line. It includes two endpoints and the line's methods, such as: *length of line, perpendicular distance, angle between two lines*. *Line* has been used in more functions of software.
- `Edge` class is an intermediary class. It stores the information of the image at the beginning; after that, its information is used to construct the approximated lines of the image.

- `Matrix` class is used to store the information of the image after decoding.
- `Image` class presents the information of an image (i.e file name, list of edges, matrix that represent the image). It also provides the methods on image such as computing histogram of image, converting image, reading the manual landmarks.

The function classes are contains the methods to implement the processes on the image. They are separated into four groups: segment the image, calculate the pairwise geometric histogram of the image, apply the probabilistic hough transform on image and estimate the landmarks on image.

- `Segmentation` classes implement the method to apply the segmentation methods on image such as threshold method, Canny algorithm. It also includes the methods to extract the edge of the image or change the display form of the image into approximated lines.
- `Pairwise geometric histogram` classes provide the method to compute the geometric histogram of the image and measure the difference metric between two images. These classes include `LocalHistogram`, `ShapeHistogram`, `GeometricHistogram`.
- `Probabilistic Hough Transform` classes are implemented to detect the presence of the scene image in the model image. At the beginning, the classes detect the reference point of the model in the scene, and the last result of this process is estimating the manual landmarks of model image in the scene image. The classes include `PHTEntry`, `PHoughTransform`, `ProHoughTransofrm`.
- `Estimating the landmarks` classes (`LandmarkDetection`)provides the methods to verify the estimated landmarks that are detected by probabilistic hough transform. These classes have also methods to evaluate the correctness of the estimated landmarks position.

4.3 Experiments

The dataset is two set of biological images: *left mandible* and *right mandible*. Each dataset contains 293 images (3264 x 2448). However, the datasets are filtered by suppressing the "bad images" that are the empty images or the image contains the broken object. As the result, the dataset includes 290 right mandible and 286 left mandible. The experiments are consider on two aspects: the runtime and the accuracy of the estimated landmarks. The software had

Machine	No of images	Segmentation(second)	Estimation(second)
Machine 1	1	0.844	31.4245
Machine 2	1	0.27782	10.4392
Machine 1	290	571.576	13000.9131
Machine 2	286	171.589	4665.79

Table 4.1: The runtime of program on two machine

implemented in several steps to estimate the landmarks. The runtime can be calculate in separated step to improve the runtime of all processes. As in table 4.1, we show the runtime on two stages of method on two machine[1]. On each system, we compute the runtime on one image

[1]
- Machine 1: Intel(R) Core (TM) 2 Duo CPU T8100 2.1GHz, 2GB of RAM
- Machine 2: Intel(R) Core (TM) i7-47900 CPU 3.6GHz, 16GB of RAM

and a set of images (table 4.1).

The accuracy of estimated landmarks is evaluated by comparing the coordinates of estimated landmarks and manual landmarks. The manual landmarks are indicated by biologist (by hand) with 18 landmarks for each right mandible and 16 landmarks for each left mandible. The automated landmarks are indicated based on the learning from the manual landmarks of model image. Figure 4.4 show the model image and its manual, and figure 4.5 shows a scene image and its landmarks that are estimated from model's landmarks.

Besides, the accuracy of the system can be determined by comparing the differences(in pixels) between the landmarks located by this method and the manual landmarks which was indicated manually by the biologist. The charts in the image 4.6 and 4.7 show the comparison between the manual and automated landmarks on two sets of data (*right mandible, left mandible*). The **blue** line describes the size of the manual landmarks on set of images. The **orange** line presents for the size of the automated landmarks by software. It is clearly that the automated landmarks is near with the manual landmarks (about 5% is exactly, 75% is near and 20% is far). Based on the processes, this method has to pass several steps, the result of each step will effect on next steps. Thus, to evaluate the accuracy of this method, we can evaluate the result of each step.

To evaluate the method, we carry out the experiments on the random models and 62 scene images. The centroid of estimated landmarks is quite near with the manual centroid. According the result, we can use the estimated centroid to classify the image on left and right mandibles. The figure 4.8 show manual centroid and estimated centroid from some random model of mandibles: the "blue" color and "other" are presented for manual centroid and estimated centroid of each image, respective.
However, as can see that the estimated landmarks are depending on the chosing of model image. If we choose good model, the landmarks will be indicated adequately. Table 4.2 shows the success rate of method on difference models. Besides, we also have the cases that method can

Model image	Success rate(%)
Md 19	82.26
Md 63	79.03
Md 152	16.13
Md 237	56.45

Table 4.2: The successfull rate of estimating

not determine all the landmarks on image. Following the result of estimating, the error cases can be come from arbitrary stage in the method such as:

- Bad segmentation
- The error about the probability when applying the PHT
- The size of the bounding box in template matching.

4.4 Parameters

As we have seen, the software has to pass the steps indicate the estimated landmarks. In each step, we have used the parameters to configure the software. As detail, the parameters that used in each stage have described as followed.

- The ratio between the *lower threshold* and *upper threshold* in *segmentation stage*. The default ratio in the software is **(1 * threshold value) : (3 * threshold value)**. The *threshold value* is identified by analysis the histogram of the image.
- The perpendicular distance from a point in edge to the **endpoints line** (endpoints line is the connected line of two endpoints in the edge) when we break the edge into list of approximated lines. This is the condition to stop the algorithm. In default, the distance is set to 3 pixels.
- The accuracy of the PGH matrix. This the size (height and width) of the matrix that used to record the angle and distance between each pair of lines in the image. The program has provided an enumeration for angle accuracy (i.e Haft Degree, Degree,...). The default value in program is **180** degree for angle accuracy and **500** for distance accuracy.
- In PHT stage, a pair of lines is considered *closet lines* if it has satisfied conditions:
  - Length of each line is greater than **60** pixels
  - Angle between two lines is greater than 15 degrees.
  - Perpendicular distance from one of two endpoints on a line to another is less than 5 pixels
- In PHT stage, the conditions for conclusion that two pair of lines are similar:
  - The subtraction between the angles is less than 1.
  - The subtraction between ratio couple of scene lines and reference lines is less than 1.
  - The subtraction between distance of two pair of lines is less than 2.
- The size of bounding box in *template matching* stage. The bounding box is a rectangle surround the landmarks and accepting the landmarks is center point of the rectangle. In default, the size of bounding box surround the reference landmarks (in model image) and estimated landmarks (in scene image) are **400** pixels and **1400** pixels, respective.

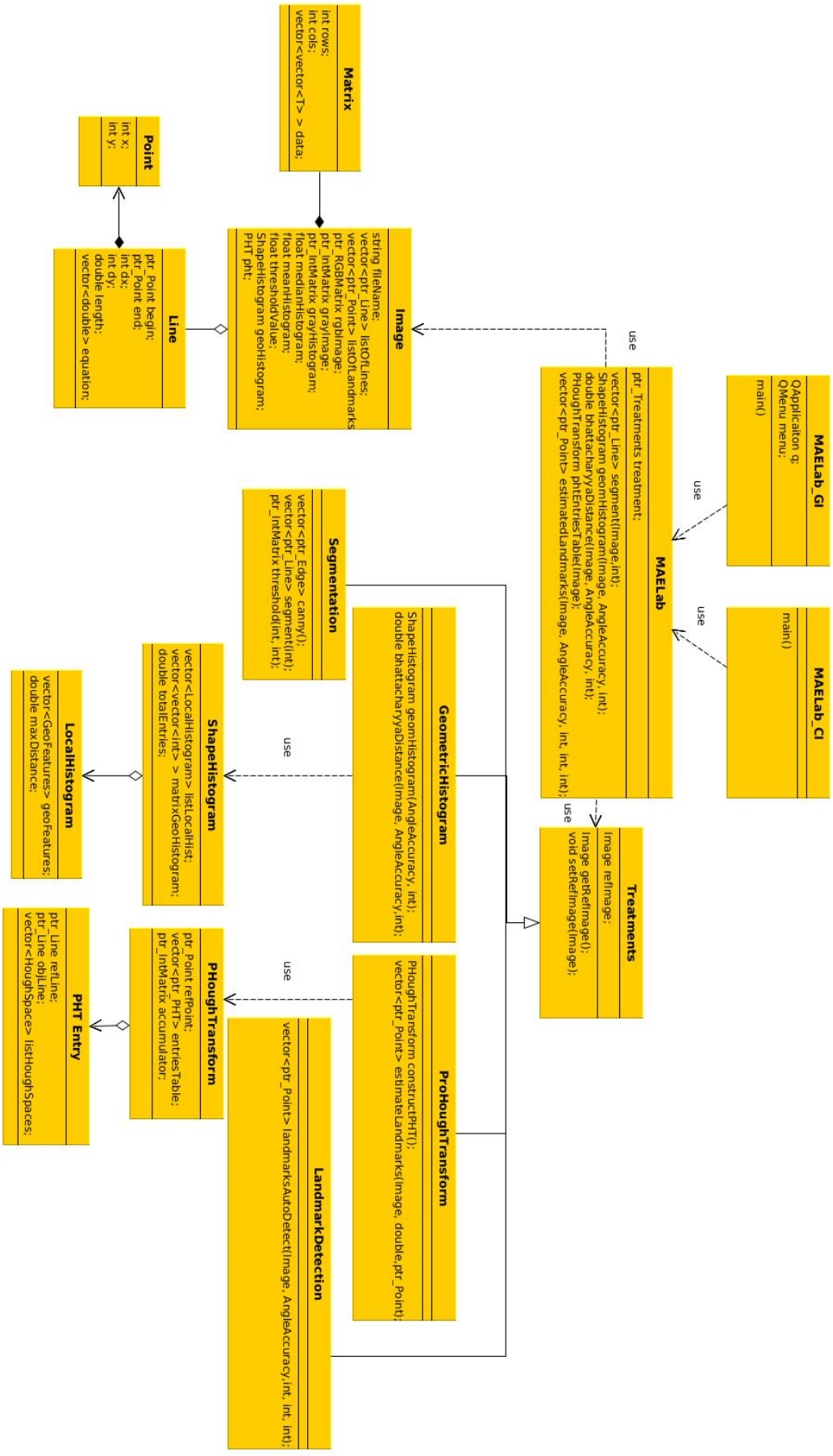


Figure 4.3: Classes diagram

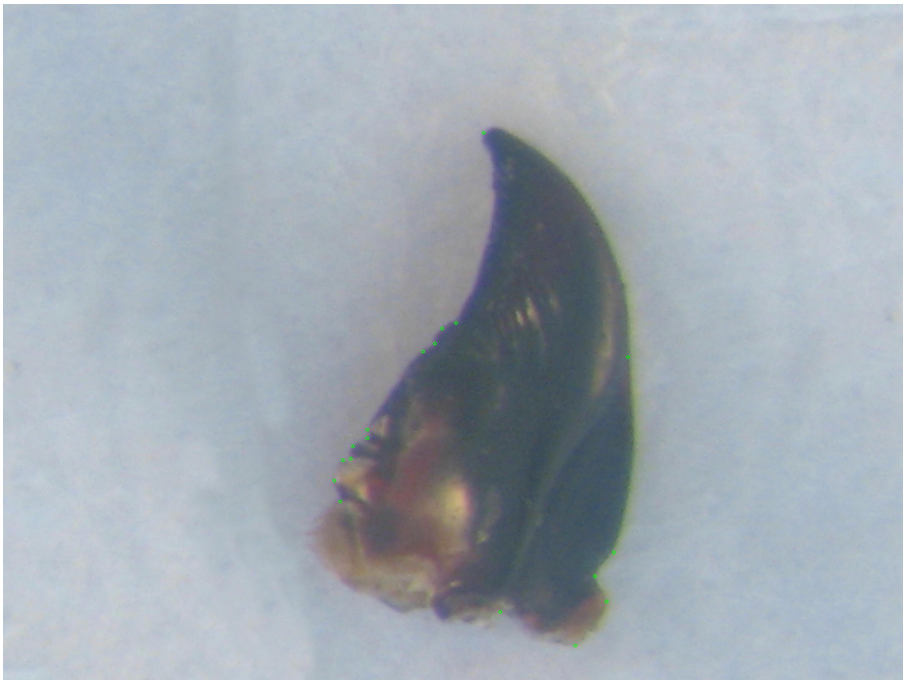


Figure 4.4: Model image with manual landmarks

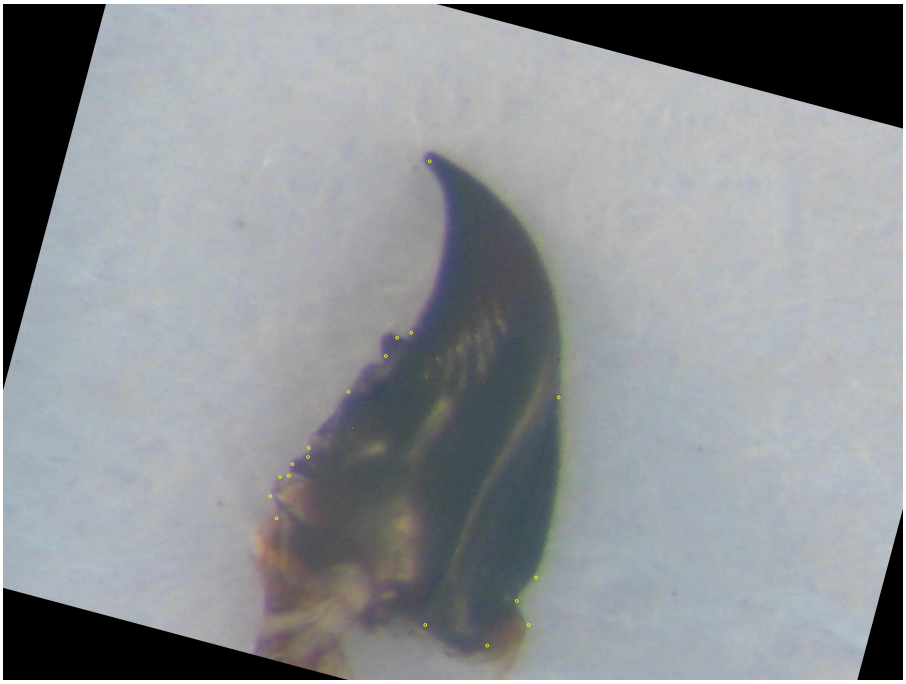


Figure 4.5: Automated landmarks indicated by our method

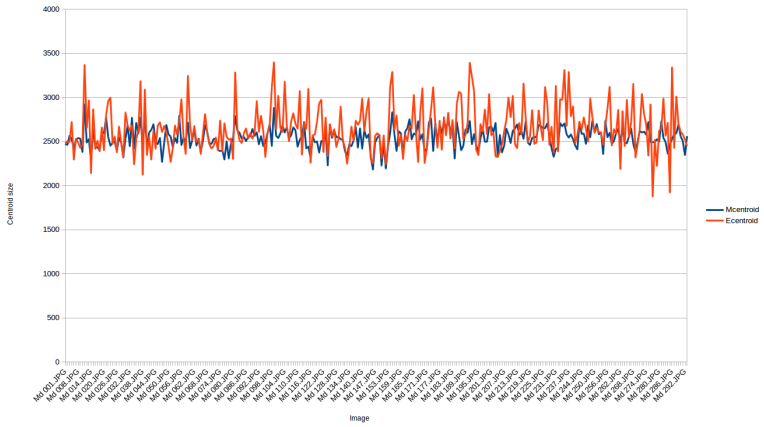


Figure 4.6: The chart presents the accuracy between manual and automated landmarks on right mandible

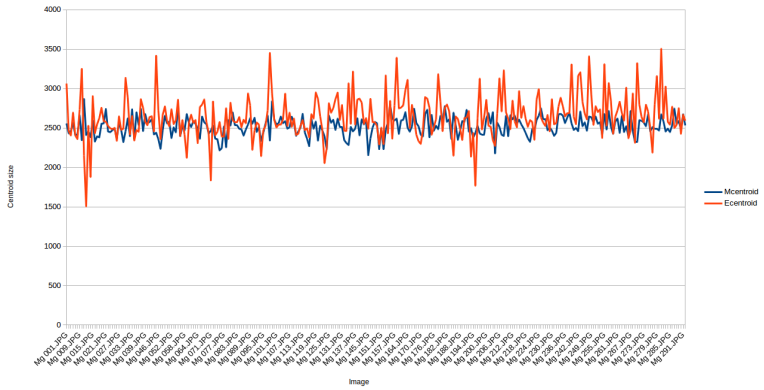


Figure 4.7: The chart presents the accuracy between manual and automated landmarks on left mandible

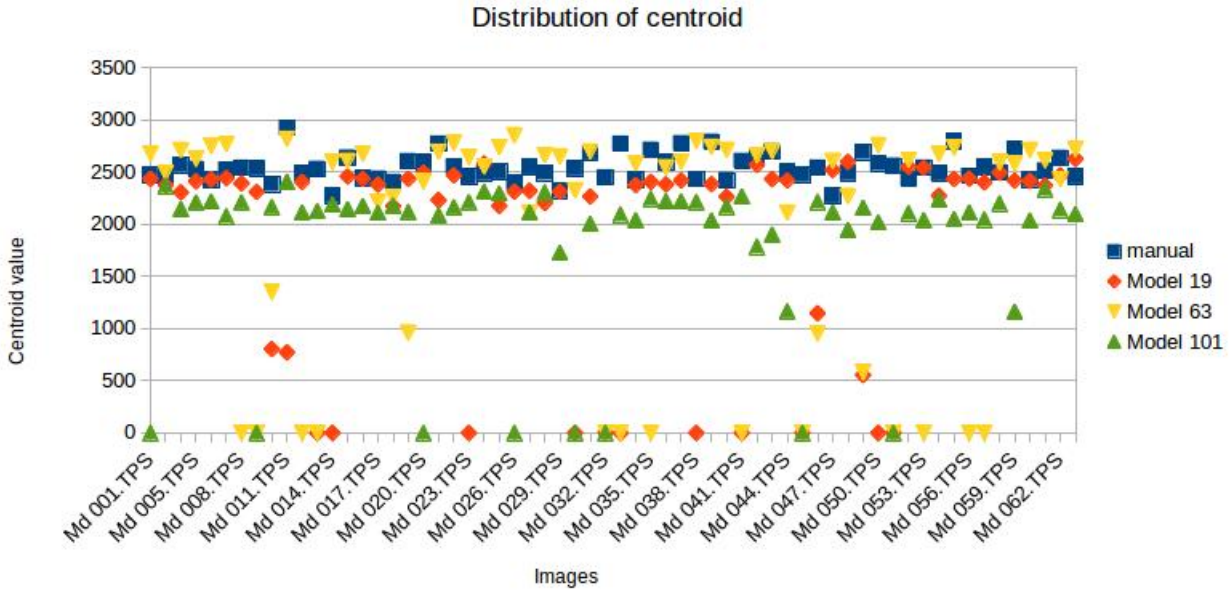


Figure 4.8: Distribution of centroid

Part II

Deep learning

In the previous of thesis, we have studied the basic methods in image processing field. Then, the methods have been applied to determine the landmarks on biological images. They are known as the very well methods in image processing. In this part of thesis, we introduce another field that we have also applied to image analysis in recent years, that is machine learning, specially deep learning.

Chapter 5 gives the principle context of machine learning. In this chapter, we present there problems of machine learning and we also give an overview about some machine learning algorithms.

Chapter 6 presents an sub-field of machine learning, **Deep Learning**. It displays an overview of deep learning problems. It has also present the convolutional neural network, a popular technique has been used in Deep Learning. At the end of the chapter, the libraries, which have been developed for Deep Learning, are also introducing.

Chapter 7 shows the applying of Deep Learing to predict the landmarks on biological images. This chapter presents the building process of the networks which have used to predict the landmarks. In this chapter also review some convolutional neural network, which have been employed to predict the facial keypoints.

Chapter 5

Machine Learning

Machine learning is a norm refer to teach the computer the abilities which are only done by the humans by applying the algorithms. The learning of computer is done through experience directly or observations of an algorithm on a database. In general, machine learning is mention to how to learning better in the future based on the experienced of current situation or the past. A machine learning algorithm is an algorithm that is able to learn from data, it was built based on the task for a machine learning system.

Machine learning has been applied to solve many problems in computer science, i.e. classification, regression, language translation, ... The machine learning task can be divided into two categories: *supervised learning* and *unsupervised learning algorithms*.

5.1 Supervised learning

Supervised learning is the machine learning tasks that the system try to find the output for an input based on the pair input-output examples that it has been seen before. It refers to a function can be learned from a labeled training data. In training data, each pair example includes an input and an output (which considered as the right output of the input). The supervised learning algorithm analyzes the training data and provides an inferred function, which can be used for finding the output of a new input. Three important elements of a supervised learning problem includes: training data, learned function and corresponding learning algorithm. In order to solve a problem by supervised learning, the steps can be followed:

1. Determine the training set of the problem, this includes finding the corresponding output for each input of training set

2. Choose the structure of the learned function and corresponding learning algorithm

3. Determine the features that they can be gathered and represented on the learned function. The accuracy of learned function will be improved during the training process.

4. Run the learning algorithm on training set. In some algorithm, the parameters can be adjusted to optimize the performance of remaining training process.

5. Evaluate the accuracy of the learned function on test set.

Given an input space X, an output space Y, a training set T includes N training examples $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ which $x_i \in X$ is the feature vector and $y_i \in Y$ is its label. A learning algorithm seeks a function $g : X \to Y$, it means that the learning algorithm tries to find a function g so that for each feature vector in the input space X will be correspond with a value in output space Y. Sometimes, the function g is replaced by a scoring function

$f : X \times Y \to R$ such as f function returns the score when we map an input feature vector to output space, i.e. the probability function. The learning algorithm will perform the score function to obtain the highest score. In order to measure the fitting of the function with the training data, a loss function $l : Y \times Y \to R^{\geq 0}$ is defined. For training example (x_i, y_i), the loss of predicting value $\hat{y}$ is $L(y_i, \hat{y})$.

In supvervised machine learning, we have many kinds of learning algorithms, i.e. nearest, k-, ... but most of them are based on a probability distribution $p(y|\mathbf{x})$).

5.2 Unsupervised learning

Unsupervised learning refers to the machine learning algrithms garther a function that describes the structure of ublabeled data. These algorithm are experienced only on features of data without any supervision. When talking about unsupervised learning, we often think about the clustering algorithms but we have also other algorithms along with clustering algorithms, such as anomaly detection or neural networks.

We have many kinds of task can be solved with machine learning. Some of common machine learning tasks include the following:

- *Classification*: In this type of task, the computer is asked to indicate a category in k category which the input belongs to. To solve this task, the learning algorithm uses a function $y = f(x)$, the model assigns the input described by vector x to a category identified by score y.

- *Classification without input*: A challenge of classification is missing the input vectors. In this case, to solve the classification task, the learning algorithm only has to define a single function mapping from a vector input to a category output. When some of inputs are missing, instead of providing a single classification function, the learning algorithm must learn a set of functions. Each function corresponds to classifying x with different subset of its inputs missing.

- *Regression*: the computer program is asked to predict a numerical value given some input.

- *Transcription*: machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form.

- *Translation*: The input already contains the sequence of symbols in some languages, the computer program must convert it into the sequence of symbols of other languages.

- *Structure output*: involve any task where the output is a vector with important relationships between the different elements.

- *Anomaly detection*

- *Synthesis and sampling*: The program is asked to generate the new example that are similar with the training data.

- *Imputation of missing value*: The algorithm mus provide a prediction of the values of the missing entries in a new example.

- *Denoising*

- *Density estimation or probability mass function estimation*

5.3 Supervised learning algorithms

5.4 Unsupervised learning algorithms

5.5 Stochastic Gradient Descent

Chapter 6

Deep Network

In recent years, deep learning has succeeded with many applications in different fields. In which, the neural network is the most popular method in deep learning to solve the problem of the high dimensions dataset. This chapter focuses on the discussion about the deep network and its parameters. Firstly, we overview the neural network, its basic components. Secondly, we focus on the main method to update the parameters of neural networks. Thirdly, we mention the objects of this method, dataset. In this part, we will describe the necessary to enlarge the data for the neural network. At the end of this chapter, we present the algorithms that are hired to optimize the learning process.

6.1 Deep learning

Deep learning is known as a part of machine learning. It includes the methods based on learning data representation by allowing the computation on the models that are composed of multiple layers. Each layer extracts the presentation of the input data from the previous layer and computes a new presentation as the input for the next layer. In the hierachy of a deep learning model, the higher layers of representation enlarge aspects of the input that is important for discrimination and suppress irrelevant variations. Each level of representations is corresponding to the different level of abstraction. Deep learning methods work on a large dataset using the backpropagation algorithm to improve the result after each step. The methods of deep learning have effectively improved the results in classification problems [], object recognition [], speech recognition [] and other domains [].

In deep learning, using neural networks is known as the most popular method. This is a computing-system based on a collection of connected units (called *neurons*). Each connection (called *synapse*) between the neurons can transmit the signal from a neuron to another neuron. The receiving neuron processes the signal that it received, then it sends the resulting signal to another neuron connected to it. Neurons and synaptes may have the weights as learnabled variables, which can used to increase or decrease the strength of signal that it sends to next units. Normally, neurons are organized in layers with different kinds of transformation inside. The signal is travelled multiple times from the first layer (input layer) to the last layer (output layer).

6.2 Neural network

6.2.1 Neural

The basic components of the brain is a neuron. For the ordinary man, we have billion neurons in the human nervous system, and they are connected by the billion of synapses. Each neuron

receives input signals from its dendrites and procedures output signals along its axon. At each neuron, the input signals are received, analysed and then given a decision. Fig. 6.1 shows the model of a neuron: the left side presents the connections of a biological neuron in human brain, the right side describes the mathematical model at the neuron.

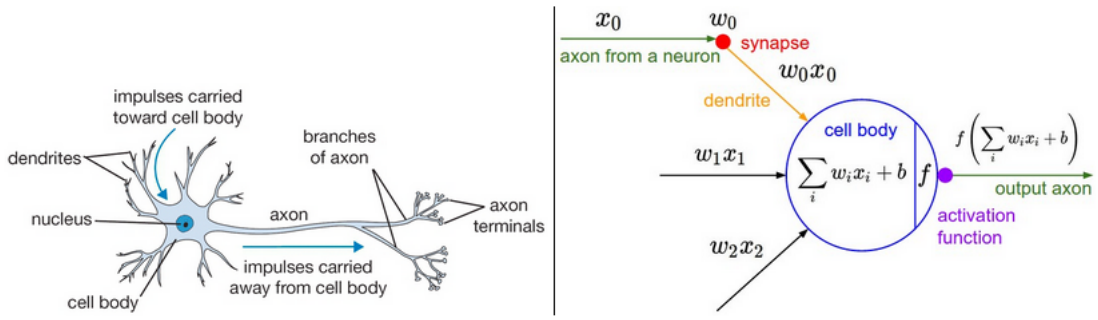


Figure 6.1: A drawing of a biological neuron and its mathematical model

In the computational model of a neuron, the signals travel along the axons interact multiplicatively with the dendrites of the other neuron based on the synaptic strength at the synapse. The synaptic strength are learnable and control the strength at influence or inhibitory of one neuron on another. In basic mode, the input signals are summed and compared with a threshold value (called *firing rate*). If the sum is greater than threshold value, the neuron can "fire", sending a spike along its axon. It look likes what we have seen in the real world, the human makes a decision when he take into acount all the conditions. The computation at a neural can be considered as following:

$$Y = \sum(\text{weight} * \text{input}) + \text{bias} \tag{6.1}$$

Depending on the values of the parameters, the output of Y can be from $-\infty$ to $+\infty$. So, how do we decide what does a neuron should do in a large range? Should it "fire" or not? Therefore, an "activation function" has been added to check the value of Y and decide what the neural should be "fire" (called *activated*) or not.

The first thing in usual is using a threshold for activation function. If the value of Y is greater than the threshold value, the neural will be declared as actived; otherwise, it is not activated. This kind of activation is called *Step* function. It seems that Step function is really work when we would like to create a binary classifier. *For example*, we would like to classify the samples of a dataset includes the samples of two classes (i.e. Class 1 and Class 2). Clearly that Step function works well in this case because it just provides a value to precise a sample will belong to "Class 1" or "Class 2". But the problem does not stop there, the question is raised as what will happen if we want to classify more than two classes (i.e three or four classes)? Of course, Step function can still be used in this case if we consider an activated class and other classes are non-activated. But it is really hard to train and converge follows this way. It will be better if we have "non-binary" activation which can provide the activated (or non-activated) probabilty for each class. The first solution is **Linear** function. This function gives a range of activations. The user can combine the output of some neurons before deciding. So, that is good too.

However, a neural can not stay alone, it need to connect to other neurons until the last one. In fact, the neurons are organized by the layers and they are connected. If each layer is activated by a linear funtion, the final activation function of the last layer is just a linear function of the input of the first layer. This means the layers that we have tried to create can be replaced by a single layer. Therefore, we have to use some "non-linear" activation function in the network if we would like to create a neural network with many layers, i.e. sigmoid, tanh or rectified linear Unit (ReLU).

Sigmoid function is one of the most widely used activation function (Eq. 6.2). Its output is always going to be in range $(0, 1)$. It means this activation can bound the range of the output instead of $(-\infty, +\infty)$ of linear function. Consider on Eq. 6.2, if x stays in the range around the origin (i.e $[-2, 2]$), then the value of activation has a change being considered. It means any small changes in the values of x in this region will make the output to change significatly. Additional, when x is bigger then $\sigma(x) \to 1$ and otherwise, $\sigma(x) \to 0$ if x smaller which means Sigmoid function makes clear distinctions on classification.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6.2}$$

Tanh is also another popular activation function (Eq. 6.3). It is known as a scaled of Sigmoid function (two times Sigmoid). The characteristics of Tanh is similar with the Sigmoid function. However, the output of this functions is zero centered. It means the boundary range is changed to $(-1, 1)$. This point mj

$$tanh(x) = 2\sigma(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{6.3}$$

Rectified Linear Unit (ReLU) function (Eq. 6.4) is another activation function which has become in the last of few years. The activation is threshold at zero. At first look this function likes the linear function but in fact they are different because ReLU outputs zero across half its domain. This makes the derivatives through a ReLU remain large and consitent whenever the unit is active. Comparing with Sigmoid or Tanh functions, ReLU was found to greatly accelerate the update parameters of the networks. Addition, ReLU can be easily implemented by threshoding at zero.

$$f(x) = max(0, x) \tag{6.4}$$

Besides, we have also some generalizations of ReLU such as **Leaky ReLU** (Eq. 6.5), **PReLU** (Eq. 6.6). They are also more broadly applicable.

$$f(x) = max(0.01x, x) \tag{6.5}$$

$$f(\alpha, x) = max(\alpha x, x) \tag{6.6}$$

Actually, we have many kinds of activation functions along with the mentioned functions above. Choosing an activation function for a neuron is depending on the objective of the user when they designed the network. For example, a sigmoid function works well for a classifier because approximating a classifier function as combinations of sigmoid is easier than ReLU for example.

6.2.2 Neural network

Fig. 6.2 shows architecture of a simple neural network. The leftmost layer in this network is called the *input layer*, the rightmost layer is called the *output layer*. The neurons within the input layer are called input neurons, the neurons from output layer are called output neurons. When design the network, the input and the output are often straightforward. It means that the neural networks is designed where the output form one layer is used as the input to the next layer, there are no loops in the network, it always feed forward, never feed back (called feedforward networks).

So, the neural network includes many layers are designed as an directred acyclic graph from the intput to the output layer. The output of previous layer is used as the input of the next

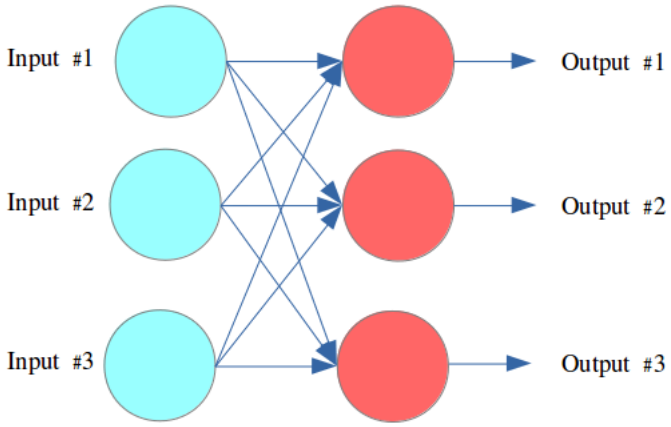


Figure 6.2: A model of neural networks

layer. At each layer excepts the output layer, the output is indicated by a activation function (i.e sigmoid, tanh,...). The size of a neural network can be to compute as the number of neurons, or the number of parameters.

6.2.3 Deep network

A deep neural network is a neural network with multiple layers between the input and the output layers. These layers are called hidden layers. Each layer tries to find the correct mathematical operator to turn its input into the next layer. The deep neural network forwards the data from the input layer to the output layer without looping back: The network creates the connections of neurons and assigns the "weight" for each connection. At each layer, the weights and its input are multiplied and return an output to transfer to the next layer. Further, an algorithm is used to adjust the weights so that make certain parameters more influential until it receives the correct mathematical manipulation on all dataset. Fig. 6.3 presents a deep neural network with multiple hidden layers between the input and the output.

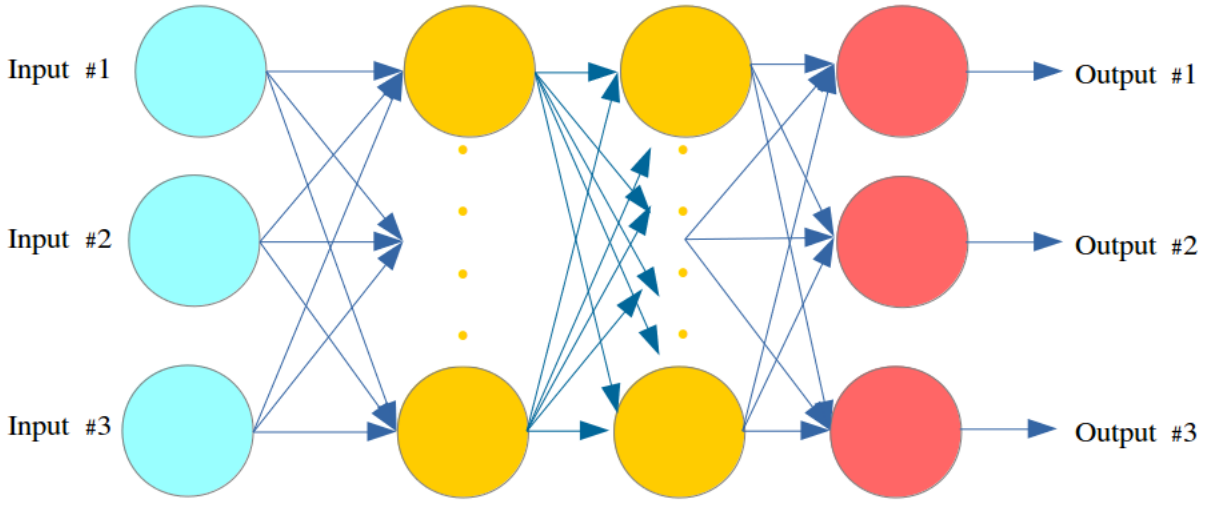


Figure 6.3: A deep neural network with multiple hidden layers

As other machine learning model, designing a deep neural network to solve a problem must be specified an optimization procedure, a cost function and a model family. In deep learning, the gradient-based learning is widely applied because it focus on the difference between the linear models and neural networks. In linear models, the solvers used to train the linear models with

global convergence guarantees; instead of, neural network uses gradient-based optimization to drive the cost function to a lowest value after each iteration (because neural network is usually trained by using iterative). Along with gradient-based optimization, a cost fucntion and the represetation of the output of model must be chosen.

A cost function of a neural network is used to compute the difference between the real data and the model's outputs. It is usually updated by each iteration of training process (or validation process). In most of case, we use the cross-entropy between the training data an the model's predictions as the cost function. It means that we define a distribution $p(y|x;\theta)$ and we simply use the priciple of maximum likelihood. But sometimes, we mergely predict some statistic of y conditioned on x. The total cost functions of a neural network often combines a primary cost function with a regularization term to make the learning algorithm intend to reduce its generation error.

In our mind, we think that we can separate the choice of the cost function and the output units but in fact, they are related together. For example, if we want to use cross-entropy as the cost function, we need to choose the way to represent the output so that the computing is easy and cheapest. Depending on the solved problem, the output units are chosen to fit with it. For example, we can use the Sigmoid function for a binary classification problem; the Linear function for a transformation with no nonlinearity; the Softmax function for a classifier over n different classes.

6.3 Back propagation

A feedforward neural network accepts an input x as the initial information, then it was propagated up to the hidden units at each layer and finally procedure an output $\hat{y}$. This is called *forward propagation*. During training, forward propagation can continue until the cost is stable. The *back propagation* algorithm receives the information from the cost (lost function) then flows backward through the network to compute the gradient. This process is repreatedly to discover the gradient for updateing the weights in an attempt to minimize the loss function.

The back-propagation in the form of an algorithm is written as followed:

1. **Input** x, **network** with L layers. Set the corresponding activation a^1 for the input layer
2. **Feedforward**: For each layer $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and activation $a^l = \sigma(z^l)$
3. Compute the error vector: $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. **Back-propagation the error**: For each layer $l = L-1, L-2, \ldots, 2$ compute $\delta' = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. **Ouput**: the gradient of the cost function given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\delta C}{\delta b^l_j} = \delta^l_j$

The term back-propagation refers only to the method for computing the gradient through recursive application of **chain rule**. For an arbitrary function f, the gradient can be computed from its set of variables whose derivative are desired and set of inputs. It means the process at each function can do independent. In the next, we will describe how to compute the gradient of a function $f(x)$ by using Chain Rule of Calculus which is used to compute the derivaives of functions formed by composing other functions whose derivatives are known. Let consider a simple multiplication function: $f(x, y) = xy$. The partial derivative of this function as followed:

$$f(x, y) = xy \rightarrow \frac{\partial f}{\partial x} = y \text{ and } \frac{\partial f}{\partial y} = x \tag{6.7}$$

The objective of derivative is indicating the rate of change of the function with respect to the variable surrounding a small region near a particular point. For example, if $x = 4, y = -3 \rightarrow f(x, y) = -12$, then the derivatives on x, y are:

$$\frac{\partial f}{\partial x} = -3 \text{ and } \frac{\partial f}{\partial y} = 4 \tag{6.8}$$

It means that if we increase the values of x, the value of function f will be decreased. Otherwise, if we increase the value of y, the ouput of function f is also increased. For an addition function, the derivatives are:

$$f(x, y) = x + y \rightarrow \frac{\partial f}{\partial x} = 1 \text{ and } \frac{\partial f}{\partial y} = 1 \tag{6.9}$$

And for a MAX operation, the gradient is 1 on the larger input and 0 on other inputs:

$$f(x, y) = max(x, y) \rightarrow \frac{\partial f}{\partial x} = 1(x \geq y) \text{ and } \frac{\partial f}{\partial y} = 0(y \geq x) \tag{6.10}$$

Consider example in Fig. 6.4, it presents the function $f(x, y, z) = (x + y)z$. Let see how to apply the derivative to compute the backward pass.

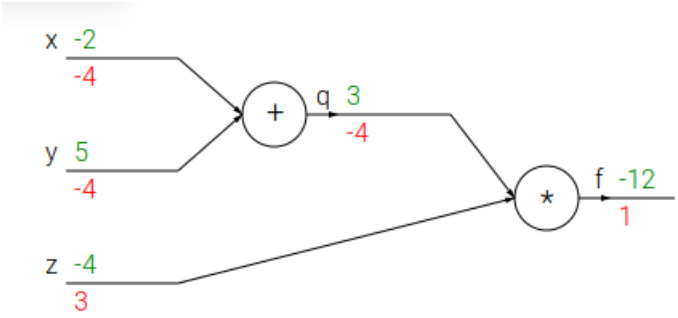


Figure 6.4: The computation graph of function f

The *green* values are the input to compute the forward pass. It is very easy to obtain the result: $f(x, y, z) = (x + y)z = (-2 + 5)(-4) = -12$. In the backward pass, the values are computed by applying derivative: At multiply gate, we can easy to compute the gradient of add gate and input z are -4 and 3, respectively. At the add gate, it takes the gradient and multiples it to all of the local gradient for its input. So, the gradient on both x and y are $1 * -4 = -4$. Clearly that if we increase the value of x and y, the value of add gate will be decreased and it will turn the ouput of multiply gate increase. So, back-propagation can see as gates communicating to each other to make a change on their outputs (decrease or increase) and to make the final value higher.

6.4 Data augmentation

The main problem of machine learning is to make an algorithm that not only perform on the training data but also on new inputs. Many plans have been used in machine learning to reduce the test error but not expense the training cost (re-trainning). A solution is given that the algorithm should be trained on a large dataset where it can covers all the issues of the attented problem.

More of the same, a deep neural network will be better if is is trained with more data. But in practice, we do not always have a large data instead the amount of data is very limitted.

A solution for this problem is to create the fake data and add it to the dataset. However, for different tasks, we may be need to apply the different augmentation methods. *For example,* in classification problem, a classifier needs to take a pair of input x and summary it with a single category y. This means that the task of a classifier is to be invariant when the input transforms. So, we can generate new (x, y) pairs just by transforming the x inputs in the training set. But this approach is not reality applicable with a *density estimation* task.

Data augmentation has been a particularly effective technique for object recognition problem [], speech recognition []. With the operations like translating the images a few pixels some directions, it can often generate the new images. Even the other operations like rotating or scaling the images have also proven effective.

Noise injection can also be as another form of data augmentation []. For many classification and regression tasks [], they have proven that the neural network can be improved the robustness when we train them with random noise applied to their input. They are also inserted into the hidden units which can see as an augmentation at multiple levels of consideration. [] shown that this approach can be highly effective provided that the magnitude of the noise is carefully tuned.

Data augmentation is considered as a part of machine learning algorithms. Usually, operations are generally applicable while other operations are specific to one application domain. So, choosing augmentation methods should be thoroughly examined before applying.

6.5 Optimization problem

The optimization for neural networks is a active area, it involves to deep learning in many contexts and one of all is finding the parameters of a neural network that significantly reduce the cost function and improve the accuracy of the algorithm. In the previous section, we have seen how to compute the gradient with back-propagation. They are used to perform the parameters of the network to reduce the cost of the model. There are several appoaches for performing the update. In this section, we present some established and common techniques which have used to optimize the neural networks.

In previous section, we have tried to calculate the gradient of the loss function. Based on that we can compute the best direction along which we should change our weights to guarante the direction of the stepest descent. This process will be repeated to evaluate the gradients and to perform the parameters updating. This procedure is called *Gradient Descent*. The simplest form of this procedure is to change the parameters along the negative direction. Assuming a vector of parameters x and the gradient dx, the update has the form:

$$x = x - learning\_rate * dx \text{ (}\textbf{learning\_rate}\text{ is a fixed constant)} \tag{6.11}$$

A neural network can be trained on a dataset of hundreds of millions of examples. It seems that wasteful to compute the full loss of the network to perform a parameter. So, we can just apply the Gradient descent over **batches** of training data to achieve the faster convergence. This process is called **Stochastic Gradient Descent (SGD)**. Using SGD to update training of a minibatch of m examples can be described in Algorithm 1.

In SGD algorithm, the learning rate is an important parameter. In previous, the learning rate is fixed as a constant but in practice, it is necessary to decrease the learning rate over time. Denote the learning rate at iteration k as ϵ_k. It is common to decay the learning rate linearly until iteration τ:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{6.12}$$

Where $\alpha = \frac{k}{\tau}$ and after iteration τ, it is common to leave ϵ constant. Usually, the learning rate is chosen by trial and error. When using the linear schedule, the parameters to choose are ϵ_0,

Algorithm 1 SGD update at training iteration k

Require: Learning rate ϵ
Require: Initial parameter θ
  while stopping criterion not meet **do**
    Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
    Compute gradient estimate: $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^i;\theta), y^i)$
    Apply update $\theta \leftarrow \theta - \epsilon\hat{g}$
  end while

ϵ_τ, and τ: τ parameter prefers to the number of iteration after that the learning rate will be decreased, ϵ_τ indicates the dropping of the learning rate, the last problem is how to choose the value for initial learning rate ϵ_0. If the learning is too large, the cost function often increases significantly. Otherwise, if the learning rate is too low, the learning process will be slow and learning may become stuck with a high-cost value. Experience indicates that the learning rate in the first 100 iterations should be higher than the next iterations. Therefore, setting up the first learning rate along with a decreasing schedule should be considered together to obtain the best result.

Besides SGD, the method of momentum [] is designed to accelerate learning. It accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. Acording that, a variable role of velocity υ is introduced, it presents the direction and speed that the parameters move through the parameter space. The velocity is set to an exponentially decaying average of the negative gradient. The SGD algorithm with momentum is describe in Algorithm 2.

Algorithm 2 SGD with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter θ, initial velocity υ
  while stopping criterion not meet **do**
    Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
    Compute gradient estimate: $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^i;\theta), y^i)$
    Compute velocity update: $\upsilon \leftarrow \alpha\upsilon - \epsilon\hat{g}$
    Apply update $\theta \leftarrow \theta + \upsilon$
  end while

Nesterov Momentum is another version of the momentum on SGD. The difference between Nesterov momentum and standard momentum is where the gradient is evaluated after applying the current velocity. It looks like we add a correlation factor to the standard momentum. The complete Nesterov momentum is presented in Algorithm 3.

The momentum algorithm can accelerate learning and mitigate the issues of the network, but it is expensive of including other hyperparameters. In which, the learning rate is one of the hyperparameters that is the most difficult to set because it has a significant impact on model performance. Tuning the learning rate for each trial is a very expensive process, so adjusting and adapting the learning rate throughout the training phase can be suitable to impale the cost because it is often highly sensitive to the directions in the parameter space. In this part, we highlight some common adaptive methods on learning rate.

Delta-bar-delta algorithm [] is the first approach to adaptive local learning rates for model parameters during training. The algorithm describes that we should increase the learning rate when the partial derivative of the loss with respect the parameters remains the same sign and

Algorithm 3 SGD with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter θ, initial velocity υ
while stopping criterion not meet **do**
Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha\upsilon$
Compute gradient: $\hat{g} \leftarrow +\frac{1}{m}\nabla_{\tilde{\theta}} \sum_i L(f(x^i; \tilde{\theta}), y^i)$
Compute velocity update: $\upsilon \leftarrow \alpha\upsilon - \epsilon\hat{g}$
Apply update $\theta \leftarrow \theta + \upsilon$
end while

otherwise, if the partial derivative of the loss with respect the parameters change sign, then the learning rate should decrease.

AdaGrad algorithm (Algorithm 4) adapts the learning rates by scaling them inversely proportional to the square root of the sum of all of their historical squared values. The parameters with the largest parital derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.

Algorithm 4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter θ
Require: Small constant δ ($i.e.10^{-7}$)
Initialize gradient accumulation variable r = 0
while stopping criterion not meet **do**
Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
Compute gradient: $g \leftarrow +\frac{1}{m}\nabla_{\theta} \sum_i L(f(x^i; \theta), y^i)$
Accumulate squared gradient: $r \leftarrow r + g \odot g$
Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$ (Division and square root applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

RMSProp algorithm [] (Algorithm 5) modifies AdaGrad to perform better in non-convex function by chaning the gradient accumulation into an exponentially weighted moving average.

Adam algorithm [] (Algorithm 6) is another adaptive learning rate optimization. It looks like RMSProp with momentum but in Adam, the momenum is incorporated directly as an estimate of the first order moment of the gradient. Additional, it includes bias corrections to the estimates of both the first-order moments and the second order moments to account for their initialization at the origin.

In this section, we have described a highlight algorithms related to optimize the deep models by adapting the learning rate for each model parameter. The question is what algorithm should we choose? There is currently no consensus on this point. Choosing an algorithm is depending on the user's familiarity with the algorithm. In practice, the most popular algorithms are used include SGD, SGD with momentum, RMSProp, and Adam.

Algorithm 5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter θ
Require: Small constant δ ($i.e.10^{-6}$)
Initialize gradient accumulation variable r = 0
while stopping criterion not meet **do**
Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
Compute gradient: $g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^i;\theta), y^i)$
Accumulate squared gradient: $r \leftarrow \rho r + (1-\rho) g \odot g$
Compute parameter update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{r}} \odot g$ (Division and square root applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

Algorithm 6 The Adam algorithm

Require: Step size ϵ (default suggestion 0.001)
Require: Exponential decay rates for moment estimates, ρ_1, ρ_2 in $[0, 1)$ (default suggestion 0.9 and 0.999 resprectively)
Require: Small constant δ ($i.e.10^{-8}$)
Require: Initial parameter θ
Initialize 1^{st} and 2^{nd} moment variables $s = 0$, $r = 0$
Initialize time step t = 0
while stopping criterion not meet **do**
Sample a minibatch of m example from training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $y^{(i)}$
Compute gradi ent: $g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^i;\theta), y^i)$
$t \leftarrow t + 1$
Update biased first moment estimate: $s \leftarrow \rho_1 s + (1-\rho_1) g$
Update biased second moment estimate: $r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$
Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
Compute parameter update: $\Delta\theta \leftarrow -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}+\delta}} \odot g$ (Division and square root applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

6.6 Conclusion

In this chapter, we have described the basic components of deep learning by using the neural network. Using any algorithms for deep learning is depending on the users but generally, the problems that we have mentioned are indispensable *i.e.* designing the algorithm, optimization problem. This chapter just provides to us a generally understood about deep learning but it is really important to help us have the first overviews about a method in this field before going to the further. In the next, we will enter into another method on deep learning, Convolutional Neural Network, and then is its application to predicting landmark on the beetle dataset.

Chapter 7

Classification

Classification is a most of important task in machine learning. In classification, a function is constructed to determine the category of the input. Generally, the model of classification as following:

The process of classification includes two steps:

1. **Training**: Use the **training set** to learn what every object of a class looks like. This duration is called training a classifier or learning a model. The training set is a set with the objects which have labeled with specific category.

2. **Evaluation**: To evaluate the quality of the classifier. We use a new set **(test set)** of the objects and try to ask the classifier predict the category of the object in the test set.

In the content of this chapter, we will discuss about the classification techniques, especially, linear classification which technique has used more in neural network and deep learning.

7.1 Nearest Neighbour Classifier

The first approach to Classifier, we will develop Nearest Neighbour Classifier. This classifier do not have any relation with deep learning or convolutional networks, but it will help us to have an overview about classification problem.

The idea of Nearest Neightbour Classifier is comparing each image in test data set with all image in training data set and predict the label of closet training image. And one of simplest methods to compare two images is comparing each pixels of two images and sum of all the differences. Assum that we have two vector I_1, I_2 presented for two images, the equation to compare two images is following (called **L1 distance**):

$$d_1(I_1, I, 2) = \sum_p |{I_1}^p - {I_2}^p| \tag{7.1}$$

Actually, we have many ways to compute the distances between two image. Instead of using L1 distance, we can use **L2 distance**, which has indicated by square root of euclidean distance between two vectors. The form of L2 distance as:

$$d_2(I_1, I, 2) = \sqrt{\sum_p ({I_1}^p - {I_2}^p)^2} \tag{7.2}$$

For example, this is a way to compute distace between two images (fig. 7.1):

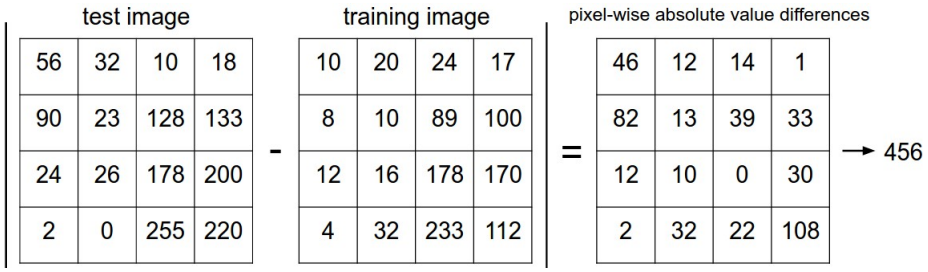


Figure 7.1: An example used **L1 distance** to compare two images

7.2 K-Nearest Neighbour Classifier

In the case of Nearest Neighbour Classifier, we just determine only one closest image in the training data with the test image when we wish to make a prediction. It means that we need some images in training data set that closest with the test image. In this case, we can use the **k-Nearest Neighbour Classifier**. The idea of this method is finding top **k** closest images instead of singel closest image (hence, when k = 1, we recover the Nearest Neighbour Classifier).

In practice, what is the best value of k that we should to use? Besides that, we have many choices for compute the distance between two images different with L1 distance, L2 distance. The method called **hyperparameters** is vary for this work. This method comes in the design of many Machine Learning algorithsm, and it is used to choose the setting values. We should try out many different values and see what works best. This is the idea, but we must be done vary carefully. Another noticed that, we do not try to evaluate on test data set with each k. After having the k, we evaluate on the test set only a single time, at the end of procedure.

The idea is spitting the training data set in two subsets: the first subset is used to training, the other subset is used to validate (called **validation set**). The validation set is used as the test set to indicate the value of k. At the end of procedure, we could determine values of k work best. We would then use this value and evaluate once on the actual test set.

In summary, split the training set into training set and validation set. use validation ton tune all hyperparameters. At the end run a single time on the test set and evaluate the result.

7.3 Linear Classification

The (k-)Nearest Neighbour Classifier had introduced about the problem of Image classification, which is predicting the label to an image from a fixed set of labels. But with the these methods, we must spend more time with large datasets an the cost for classifying is expensive. Another classification methods is known as **linear classification** which is the core of neural networks. The linear classification has two main components:

- **Score function**: which used to map the raw data to score of a category.
- **Loss function**: that quantifies the agreement between predict score and the truth category of the data.

The simplest function of a linear mapping is:

$$y = f(x_i, W, b) = Wx_i + b \tag{7.3}$$

Where:

- x_i is the raw data, *example: an image.*

- W: a matrix parameter, called **weight** matrix
- b: vector, called **bias** vector
- y: score when consider the data x_i belongs to a category.

In equation above, the input image x_i is fixed but we can control the setting of parameters **W and b**. Our goal is setting the parameters that the computing score match with the truth labels of image.

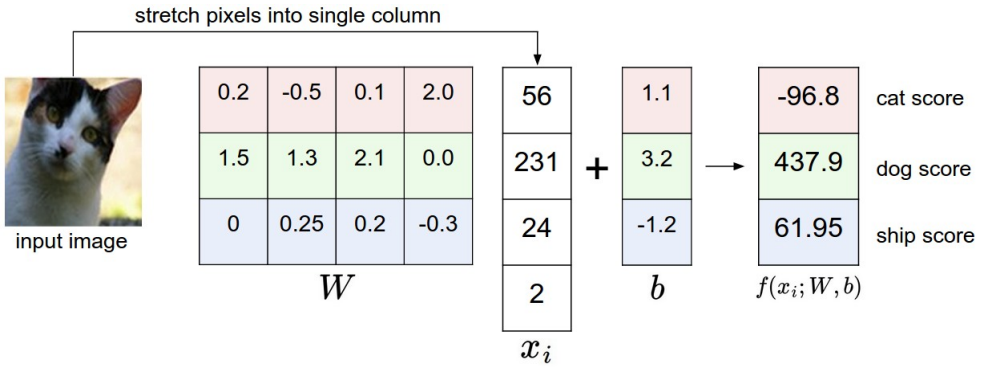


Figure 7.2: An example of mapping an image to a class scores

In training process, it is a little cumbersome to keep two sets of parameters (W,b) separately. A commonly trick is used to combine two sets of parameters into a single matrix that holds both of them by extending a vector x_i with one additional dimension and keep the constant defaut 1. Now, the new score function will be:

$$y = f(x_i, W, b) = Wx_i \tag{7.4}$$

Visualation of new score function:

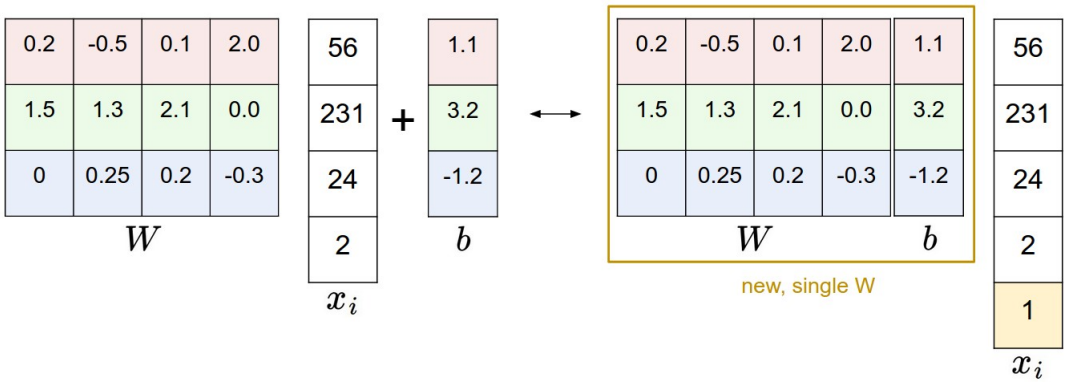


Figure 7.3: An example of bias trick

As described, we defined a score functions from a pixels value of an image to class scores with set of parameters **W**. Moreover, we need to control over parameters W such that the class scores are consistent with the ground truth labels in the training data. But not all cases are perfect, the class scores just near with the score of truth labels. So, we are going to measure the wrong with a **loss function**. Intuitively, the loss will be high if we are doing a poor classifier, ant it will be low if we are doing well. Multicalss Support Vector Machine (SVM) and Softmax function are two commonly methods for this purpose.

7.3.1 Multiclass Support Vector Machine loss

A commonly way to define the loss function called the **Multiclass Support Vector Machine** (SVM) loss. SVM loss is set up a margin Δ for the incorrect class scores. It means that SVM loss function wants the score of the correct class to be greater than the incorrect class (predict score) by at leat Δ. If this is not the case, we will accumulate the loss.

The SVM loss for the i-th is formalized as follows:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta) \tag{7.5}$$

Where:

- s_j is the score of x_i for j-th class
- s_{y_i} is the score of correct class
- Δ is margin
- $max(0, -)$ is thresholding to zero, called hinge loss.

For example, we have three predict scores of an image x_i like $s = [14, -9, 11]$, and the first class is true class of x_i. Assume that Δ is 10. The SVM loss of this case is following:

$$L_i = max(0, -9 - 14 + 10) + max(0, 11 - 14 + 10) = 0 + 7 = 7$$

7.3.2 Softmax classifer

The other popular choice to define the loss function is the **Softmax classifier**. Unlike SVM which treats the output of score function for each class, the Softmax classifier give a sightly more intuitive output and use the probabilistic description. Instead using theshold zero function as SVM, Softmax is using a **cross-entropy loss** for *hingle loss*, which has the form.

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{7.6}$$

Where: f_j is the j-th element of vector of class scores f.

In formula 7.6, the function $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is called the softmax function. This formula turns the predict scores into probabilistic values (Noticed that sum of all $f_j(z)$ is 1).

The cross-entropy between a correct distribution **p** and an estimated distribution **q** is defined as:

$$H(p, q) = -\sum_x p(x) log(q(x)) \tag{7.7}$$

The Softmax classifier is minimizing the cross-entropy between the estimated socre and true score. At the end, the loss of training process is the average of cross-entropy.

$$L = \frac{1}{N} \sum_i (H(p, q)) \tag{7.8}$$

The image 7.4 describes an example for a comparison between SVM and Softmax:

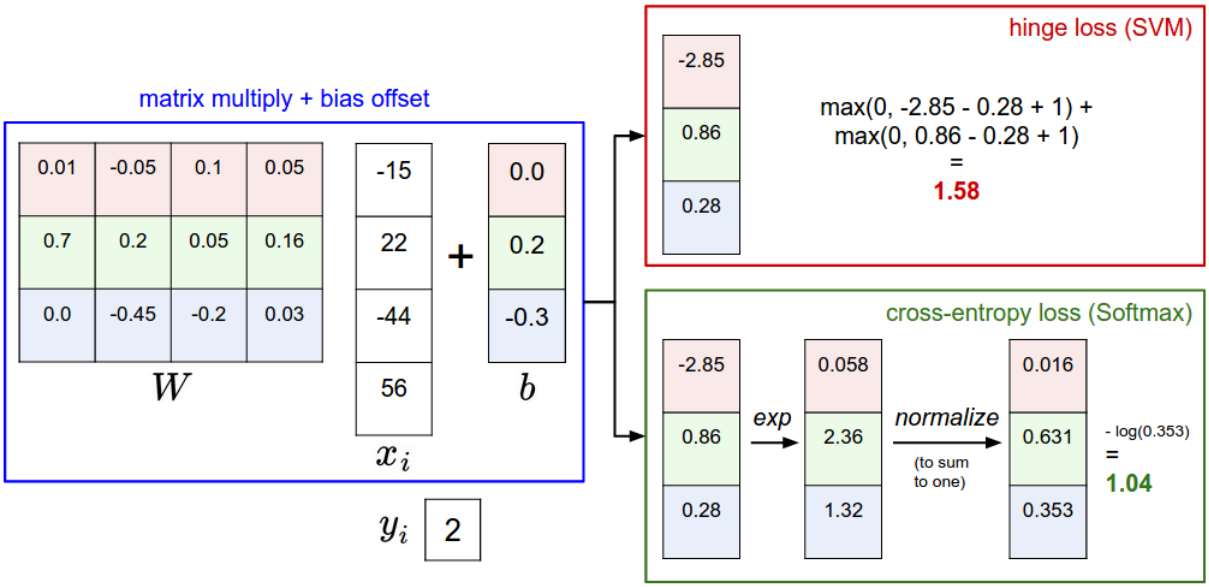


Figure 7.4: An example about SVM and Softmax classifiers

Both SVM and Softwax compute the same score of vector f. The difference is the way to present the score f: SVM uses the margin and Softmax uses probabilistic. In practice, the SVM and Softmax are usually used and compared in the machine learning systems.

7.4 How to determine the value of W and b?

As we have seen in previous section, the loss function is used to quantify the quality of any set of weights **W**. So, the choosing (or optimizing) the weights W is really important to obtain a good prediction. The goal of this process is to find W that minimize the loss function. In this section, we will describe some strategy to optimize the W.

7.4.1 Random search and random local search

The core idea is finding the best set of weights W. We begin with the random weights W, try out many different random weights and keep the W what works best.

Another way to try with random search is to try to extend the random direction . We also start with a random **W**, generate a random noise δ**W** to it and if the loss at the concern **W** + δ**W** is lower, we will perform an update. Following that method, the accuracy of classifier is getting on **21.4%** (by experiment).

7.4.2 Following the Grandient

Another method is usually used to optimize the W that following the best direction which we should change our weight (steepest increase or descend). This direction is related to the gradient of the loss function.

Gradient Descent is the procedure of repeatedly evaluating the gradient and then performing a parameter updated. In an application with the training data set is large, we must wasteful to compute the full loss function to perform the parameter. A very common way is addressing this challenge with a batches (a subset) of the training data. The extreme case of this is a setting where the mini-batch contains only a single example. This process is called **Stochastic Gradient Descent (SGD)**

7.5 Backpropagation

With a classifier, we can use SVM or Softmax to compute the loss of classifier, and the input are both the training data and the parameter weights **W** and biases **b**. Clearly, the value of training data is fixed, so we can control the value of loss function via controling the weight parameters.

In this section, we will discuss about a method to compute the gradient of a function **f(x)** at **x** (i.e $\nabla f(x)$), called backprobagation. The core of backprobagation is computing gradients of function through recursive application of **chain rule**.

Let consider a simple function: $f(x, y) = xy$. The partial derivative for this function as followed:

$$f(x, y) = xy \rightarrow \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x \tag{7.9}$$

The purpose of derivative is indicated the rate of change of the function with respect to that variable surrounding a small region near a particular point. Example, if **x = 4, y = -3** then $f(x, y) = -12$. The derivate on x is $\frac{\partial f}{\partial x} = -3$, it means if we increase the value of x by a tiny amount, the value of this funciton will be to decrease it. Otherwise, the derivate on y is $\frac{\partial f}{\partial y} = 4$, if we increase the value of y, the function also increase the output.

The derivaties for the addition operation:

$$f(x, y) = x + y \rightarrow \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1 \tag{7.10}$$

And for *max* operation, the gradient is 1 on the input that was larger and 0 on the orther input:

$$f(x, y) = max(x, y) \rightarrow \frac{\partial f}{\partial x} = 1(x \geq y), \frac{\partial f}{\partial y} = 1(y \geq x) \tag{7.11}$$

The image 7.5 show an example about applying derivative to calculate the backward pass of the fucntion $f(x, y, x) = (x + y)z$:

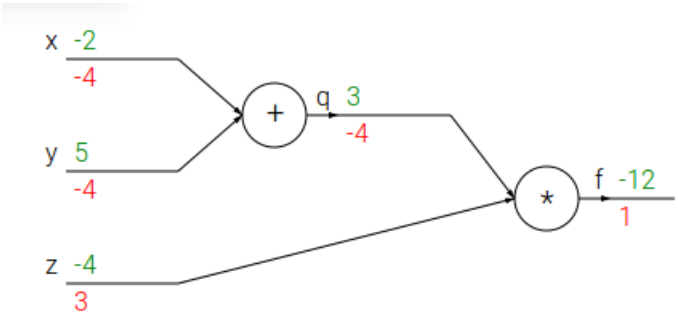


Figure 7.5: A backpropagation example

The backpropagation is good local process. Each gate of the circuit gets some inputs and compute two things (1) its output value and (2) the local gradient of its inputs with respect to its output value. Morever, the process at each gate can do independent. However, one the forward pass is over, during backpropagation the gate will eventually learn about the gradient of its output value on the final output of the entire circuit. Chain rule says that the gate should take that gradient and multiply it into every gradient it normally computes for all of its inputs.

Another kind of function at gate of circuit which we use to compute the gradient of function is *sigmoid activation* function. The form of sigmod function as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow \frac{d\sigma(x)}{dx} = (1-\sigma(x))\sigma(x) \tag{7.12}$$

Chapter 8

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are similar with the original of Neural Networks. It means that CNNs has also the score function and the loss function at the end of network. Neural Networks receive an input and pass it through a series of hidden layer but in CNNs is deeper. Each hidden layer is made from a set of neurons, where each neuron is full connected with all neurons of previous layer. The layers of the CNNs have neurons arranged in 3 dimensions: **width, height, depth**. CNNs transform the original image layer by layer from the original pixel value to the final class score. In CNNs, some layers contain the parameters but other don't. This chapter will describe the architecture and the detail of each layer in the CNNs.

8.1 Architecture

A CNN is made from the layers. The common layers in CNN are convolutional, nonlinear, pooling and full connected layers. CNN takes image as an input, pass it through the series of layers and get an ouput. Each layer has a difference function to transform the input to another layer.

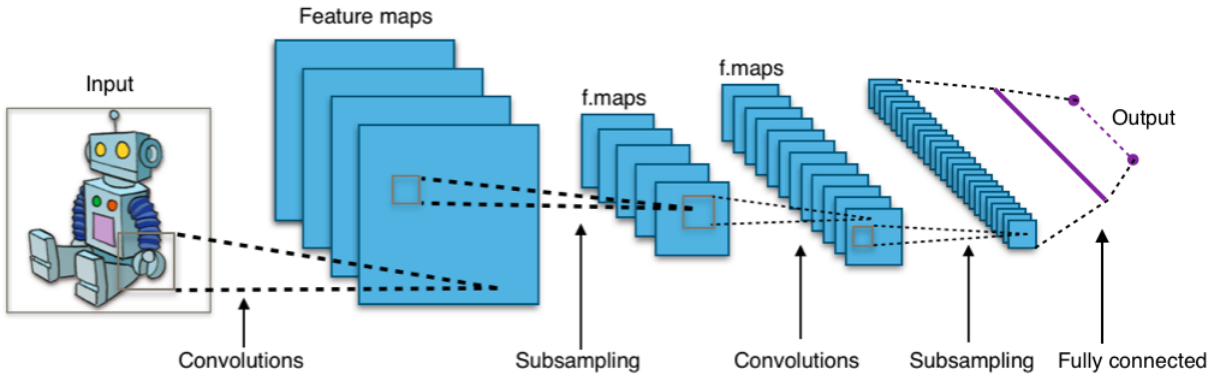


Figure 8.1: An architecture of convolutional neural network

8.1.1 Convolutional layer

Convolutional (CONV) layer computes a dot product between their weights and a small region in the input image for each small region in the input. At the output of neurons is combining the result of the connected to local regions.

CONV layer uses a set of learnable filters as parameters. Each filter is small spatially but extends the depth of the input. During the forward pass, the filter is slided over each pixel of

the input (from left to right, top to bottom) and calculate dot product between the entries of the filter and the input at this position. During the process, we can see the response of input for each filter such as the orientation of the edge or a blotch of some color on the firt layer. With an entire set of filters in each CONV layer, we will stack these activation maps along the depth dimension and procedure the output volume.

Instead of connecting a neurons to all neurons in the previous layer, CONV connect each neurons to only a local region of the input. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equal with the number of the filter). The extent of connectivity has the depth axis is equal to the depth of the input. For example, if the input has size [32x32x3] and the filter size is [5x5] then each neuron in the CONV layer will have the weights to a [5x5x3] region in the input, and total of $5 * 5 * 3 = 75$ weights. This is the way that each neuron in CONV layer connected to the input; but how many neurons that we have in the output and how the order between the neurons. With 3 hyperparameters **depth, stride** and **zero-padding** will help us control the size of the CONV output.

- **Depth**: corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- **Stride**: which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. If the stride is 2 (or more), then the filter will jump 2 (or more) at a time when we slide the filter.
- **Zero-padding**: pad the input with zeros arund the border.

We can compute the spatial size of the output through the equation:

$$N = \frac{(W - F + 2P)}{S} + 1 \tag{8.1}$$

Where:

- **W** is the input size
- **F** is the filter size of CONV layer neurons
- **P** is amount of zero padding on the border
- **S** is the stride.

The important of equation 8.1 is the constraint on stride. If we choose the stride inadequate, the result could be not an integer, it means that the neurons do not fit neatly and symmetrically across the input. Besides, using zero-padding also affects to the spatial size of the output. Therefore, the setting of the hyperparameters is considered to cheap, we can throw an exception or use zero pad the rest or crop the input to make it fit,etc. Easy to see that if a CONV layer received the input of size $[wxhxd]$, then the number of neurons is $(w * h * d)$; and if the size of the filter on each neuron is k, then we have $k * k * d$ weights for each neuron. And the total parameters that we need to keep on the layer is $(w * h * d) * (k * k * d)$, this number is clearly high. To reduce the number of the parameter on the layer, we can assume that if the filter is useful to compute at a position (x_1, y_1), then it should be useful to compute at different postion (x_2, y_2). With this way, we just need to keep unique set of weights for each depth slice(single 2-dimensional slice of depth). This technique is called parameter sharing.

In general, the CONV layer:

- Accept a input of size W_1 **x** H_1 **x** D_1

- Requires four hyperparameters:
  - Number of filters **K**
  - Size of filter (spatial extent) **F** (commonly F = 3)
  - The stride **S** (commonly S = 1)
  - The number of zero padding **P** (commonly P = 1)
- The output with size of W_2 **x** H_2 **x** D_s where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- With parameter sharing, CONV layer store $F * F * D_1$ weight per filter, for a total of $(F * F * D_1) * K$ weights and K biases.
- In the output, the **d**-th depth slice (size $W_2 x H_2$ is the result of performing a valid convolution on the **d**-th filter over the input volume with a stride of S and then offset by **d**-th bias.

Consider an example below (figure 8.2, 8.4), with the input is image of size [5 x 5 x 3]. The parameter of CONV layer are **K = 2, F= 3, S = 2, P = 1**, that is using two filter of size **[3x3]** and applying the stride of 2. Therefore, the size of the output volume is $(5 - 3 + 2)/2 + 1 = 3$ ([3x3x2]), and the zero padding P = 1, so making the outer border of the input with zero value. The figure 8.2 and 8.4 describe the process of convolution. The each the element in the output (green) is computed by elementwise multiplying the highlight input (blue) with the filter(red), summing it up, and then offsetting the sum result by the bias.

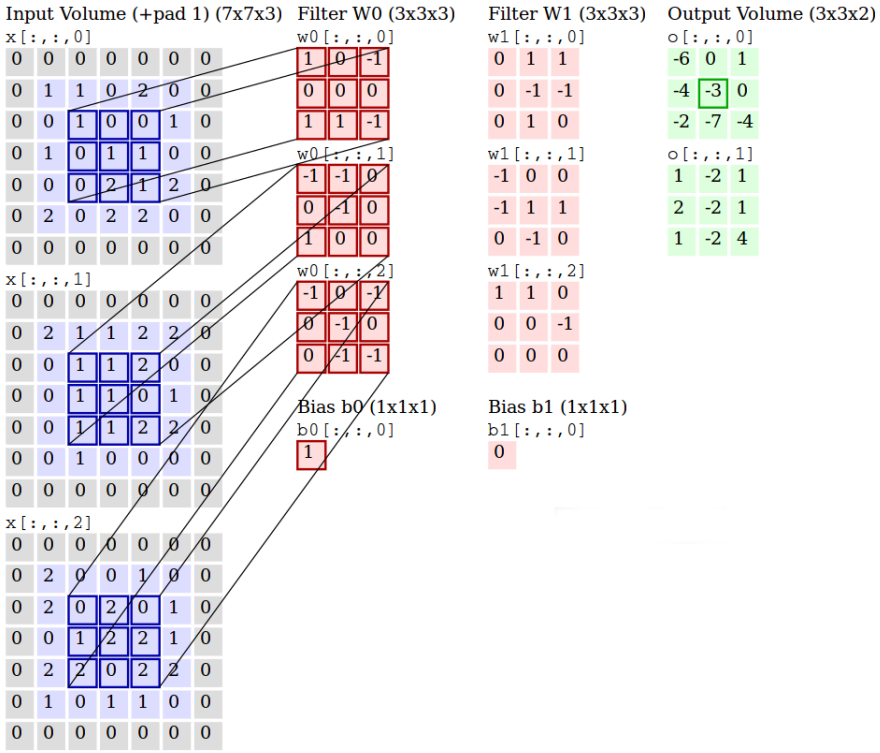


Figure 8.2: Convolutional input with W0 filter

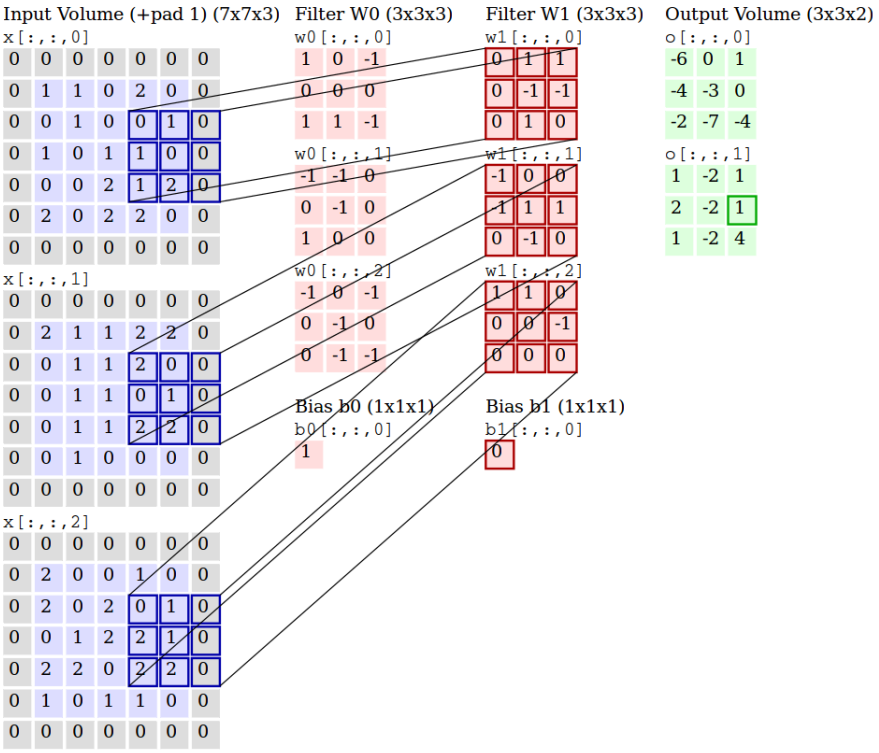


Figure 8.3: Convolutional input with W1 filter

8.1.2 Pooling layer

Pooling (POOL) layer is another common layer in CNNs network. It is used to reduce the spatial size of the representation to reduce the quantity of the parameters and control overfitting. Hence, it performs a downsampling operation along the spatial dimensions (width, height). This operation will not affect to the depth dimension of the input. More generally, the POOL layer:

- Accept a input of size W_1 **x** H_1 **x** D_1
- Requires two hyperparameters:
  - Size of filter (spatial extent) **F** (commonly F = 4)
  - The stride **S** (commonly S = 1)
- The output with size of W_2 **x** H_2 **x** D_s where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Note that it is not common to use zero padding for POOL layer since it computes a fixed function of the input.

The common function in POOL layer is **MAX** function and the size of filter is 2x2. The filter will slice over the input and using MAX operation over 4 number (in 2x2 region). Besides MAX function, the POOL layer can use the average function or L2-norm.

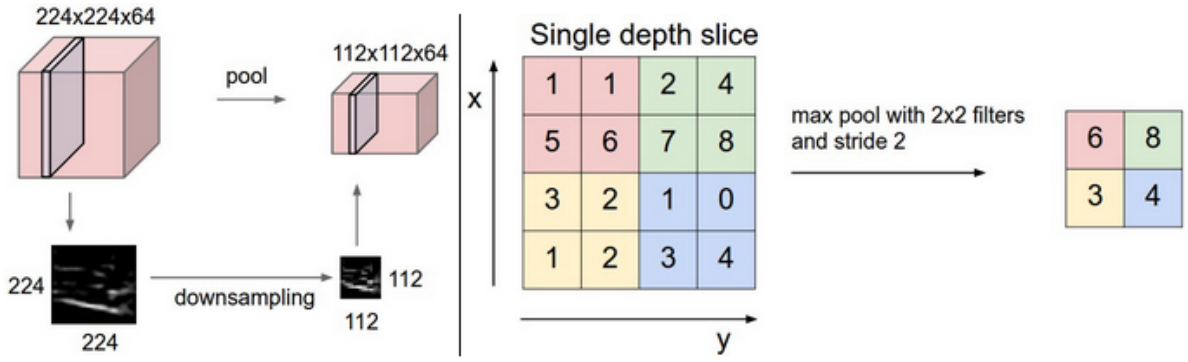


Figure 8.4: A POOL layer with MAX function

8.1.3 Full connected layer

Full connected (FC) layer computes the class scores of the input. The neurons in FC layer have full connections to all activations in the previous layer. Their activations can be computed with a matrix multiplication followed by a bias offset.

The difference between CONV and FC layer is that the neurons in the CONV layer are connected only to a local region in the input and the neurons in CONV layer are sharing parameters. However, the neurons in both layers still compute the dot products (it means the functional form is identical). Therefore, it turns out that it is possible to convert between FC and CONV layer.

- For any CONV layer can be become a FC layer if we implement the same forward function. The weight matrix would be a large matrix that is mostly zero except for at certain blocks where the weights in many of the blocks are equal.
- Conversely, a FC layer can be converted to a CONV layer by setting the filter size to be exactly the size of the input and the output will be give identical result as the initial FC layer because only a single depth column fits across the input.

8.2 Frameworks

8.2.1 Caffe

Caffe is a deep learning framework in C++ that is developed by the Berkeley Vision and Learning Center (BVLC) and community contributors. Besides the supporting help user defined the network without hard-coding, easily to change the device machine (CPU or GPU), speedly, Caffe already has a large community user. These are reasons that Caffe is used in many research projects and many application in vision, speech and multimedia.

Caffe uses a definition of *blobs* to store and to communicate data. It provides a uniform memory to hold data into the network. A network in Caffe is combined from the *layers*. A layer take input from the bottom connection and through the output to top connection. For each layer of the network, the setting parameters are different depending on the type of layer. However, each layer type has the same of three critial computiations: **setup** initilize the layer and its connections; **forward** give the input from bottom connections, compute and send the output to the top connections; **backward** given the gradient from the top connections, compute the gradient and send to the bottom connections. Besides the network components, the information for training, optimizing the network are defined in **Solver** file.

8.2.2 Theano

Theano is a Python library that allows to define, optimize and evaluate mathematical expression involving multi-dimensional arrays efficiently []. To realize this, the mathematical expressions which defined by the user are stored as a graph of variables and operations, that is pruned and optimized at compilation time. Like Caffe, Theano allows the users compile the program either on CPUs or GPUs.

8.2.3 TensorFlow

TensorFlow [**?**] is an open source software library for numerical computation using data flow graphs. Each node in the graph represents for a mathematical operation, while the edge represents the communication between the data arrays(tensors). Tensorflow supports Python and C++, along to allow computing distribution among CPU, GPU and even horizontal scaling.

8.2.4 Torch

Torch [**?**] is a scientific computing framework for machine learning. It is written by Lua language and an underlying C/CUDA implementation. At the heart of Torch is implemented the functions for neural network and optimization libraries. The user can build arbitrary graphs of neural networks and parallelize them over CPUs and GPUs in an efficient manner.

8.2.5 PyTorch

PyTorch [] is a python package that provides two high-level features:

- Tensor computation with strong GPU acceleration
- Deep Neural Networks built on a tape-based autodiff system.

Most frameworks have a static view of the network. One has to build a neural network and reuse the same structure again. With PyTorch is different, the technique Reverse-mode auto differentiation allows to change the way of the network works aribitrary with zero lag or overhead. Additional, it integrate acceleration libraries such as Intel MKL and NVIDIA to maximize speed. At the core, the backend on CPU, GPU Tensor and Neural Network are written as independent libraries.

8.2.6 Trends of Deep Learning libraries

Fig. 8.5 presents the trends of deep learing libraries in 2018. The color in the circle shows the age (in days) of the libraries from start date given on Github under Insights/Contributors (greener = younger, bluer = older). By all measures, TensorFlow is the leader. Keras, Caffe, Microsoft Cognitive Toolkit and PyTorch are in top five.

8.3 Case studies

This section gives several architectures in the field of CNNs that we have.

- **LeNet**: is developed by Yann Le Cun [29] in 1990's. It was used to read the zip code, digits.

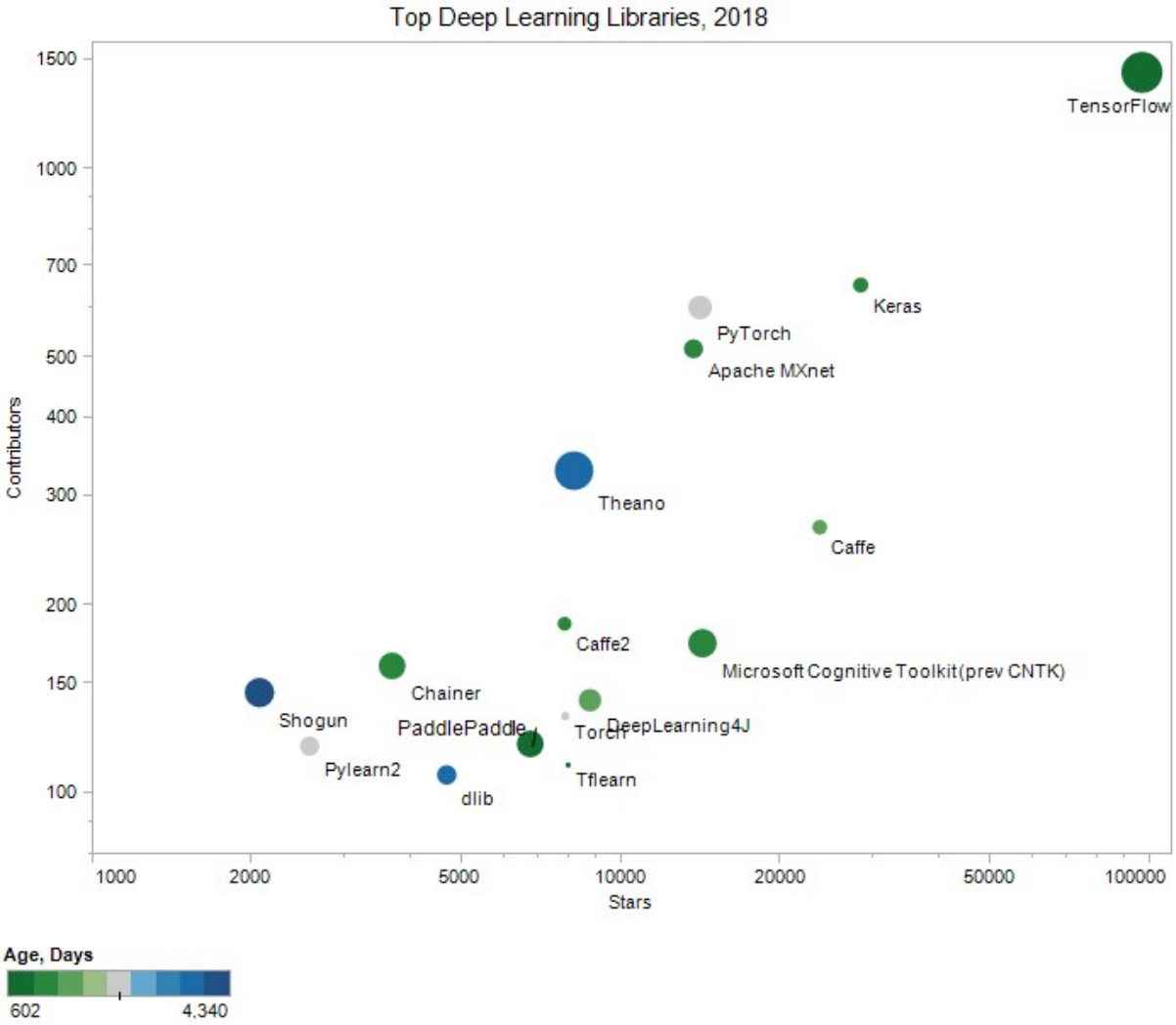


Figure 8.5: Top 16 open source deep learning libraries by Github stars and contributors

- **AlexNet**: is developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [30]. This network had very similar architecture to LeNet, but was deeper, bigger and featured CONV layers stacked on top of each other.
- **ZF Net**: from Matthew Zeiler and Rob Fergus [31]. It was improved the AlexNet by tweaking the architecture hyperparameters (extending the size of the middle CONV layers and making the stride and the filter size on the first layer smaller).
- **GoogLeNet**: from Szegedy and al from Google [32]. They develop an *Inception Module* to reduce the number of parameters in the network. Additionally, they have used Average pooling instead of FC layers at the top of the CNN.
- **VGGNet**: is developed from Karen Simonyan and Andrew Zisserman[33]. They show that the depth of the network is a critical component for good performance.
- **ResNet**: is developed by Kaiming He and al[34]. The architecture of this network is missing FC layers at the end of network. It features skip connections and a heavy use of batch normalization[35].

8.4 A small deep network with Caffe

Caffe uses prototxt format for specification of neural networks. Each layer is defined separately with the inputs, the outputs called blobs. The layers are stacked vertical with the input of layer at the bottom and the output of layer at the top. Besides, each layer expects a number of input and output blobs. The figure xx show an sample deep neural netwoks. The network begin the input layer that takes the data input, an output of network that computes the final prediction, a loss layer which can be intergrated with the final output layer. We will discuss how to create this network by Caffe.

The first layer in the network is the input layer. In this network, the ImageData has been used. It takes an input text file which contains multiple lines of input. Each line contains the path to each image and its label, separated as a space. Besides ImageData, we can use other input layers which are supported by Caffe such as HDF5, Data,...

The second layer is a convolutional layer. In this sample, we choose *kernel_size* as nine with a *stride* of three and the number of outputs will be set to 100. This layer take the output of the first layer as the input, then its output will be used at next layer. The parameters depend on how deep the convolutional layer is placed in the network and what level of detail of features is begin targeted.

The third layer is a fully connected layer which generates a prediction on the labels possible. The number of output is equal to the number of labels.

The fourth and fifth layer is the Sofmax and output layer. The layers use softmax distribution on the output of fully connected layer to compute the probabilities of the image belonging to each label. Besides, the layer also compute the loss for backpropagation for training.

Now, we have finished design network. Now, we will Caffe to create a real network. The stages to create network as follows:

1. Preparing the dataset: collect the images and place them in a directory. Then, create a text file with all the image names and its label.
2. Creating the network: The layers are stacked as description above in a file and saving in a **prototxt** format.
3. Create a solver file to describe the step learning procedure.
4. Star the training procedure with command
   `build/tools/caffe train --solver=./path/to/solver.protoxt --gpu=0`

After finishing the training procedure, a file with format caffemodel will be generated. We use this file as model of the network in specifies program.

Chapter 9

Using CNN to classify the patches

Deep learning powers computer vision with processing the data in high level such as image, sound, video in many tasks: classification, recognition or object detection. As a result of studying about the CNN, in this chapter, we proposed a model to classify the landmarks on the pronotum of beetle. For each landmark in pronotum image, a patch has been extracted with the size of 63×63. Then, the patches are used as the input of the model to train. At the end, a number of patches are tested to evaluate the prediction of the model.

9.1 Data

The pronotum dataset includes 293 images. For each image, a set of 8 landmarks have been set manually by the biologist. For each landmark, a patch is extracted with the size of 63×63. The images have been divided into 3 sets: training set included 200 images ($200 \times 8 = 1600$ patches), validation set had 60 images ($60 \times 8 = 480$ patches) and 33 images ($33 \times 8 = 264$ patches) belong to the test set.

To evaluate the efficient of the network, the data has been chosen followed 3 ways:

1. In the first way, 200 first images are used as training data; next, 60 images are used as the validation data; and the test is remaining images.

2. In the second way, 33 first images are chosen as test data; next, 200 and 60 images are chosen as training and validation data, respectively.

3. In the last way, the images are divided randomly into 3 sets with the corresponding quantity.

Besides, the patch's size also changed to find the suitable size for the patch for each landmark: 27×27, 45×45 and 63×63. The network trained with two types of the images: grayscale and RGB color.

9.2 The network

We propose a CNN-based method to classify the landmarks on the pronotum (see Fig. 9.1). The input of the network is the patches around the landmarks. The network includes three convolutional (CONV) layers followed by three pooling layers (maximum and average pooling) and two full connected layers.

The number of convolution filters are 32, 64 and 32, respectively. The window size of those filters are $5 \times 5, 5 \times 5$ and 3×3. For each CONV layer, a Rectifier Linear Unit is add to introduce the non-linearity for CNN.

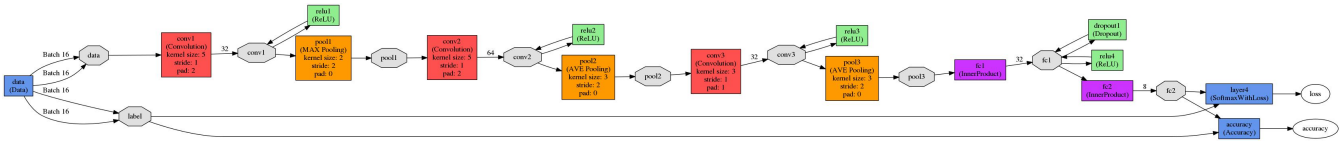


Figure 9.1: CNN model architecture

Followed each CONV layer is a pooling (POOL) layer. The POOL layers with stride of 2×2 is reducing the computation for the deeper layers. The first fully connected (FC) layer consists of 32 neurons. To reduce the risk of the over-fitting on the network, a dropout layer is followed the first FC. The second FC layer output 8 values corresponding to 8 classes of the landmarks. Finally, a softmax with loss layer and accuracy layer are placed at the last stage of the network.

9.3 Solver parameters

The network is trained in 100000 iterations. For each 10000 iterations, a test phase will be executed in 1200 iterations to validate the accuracy of the network. During training, the Stochastic Gradient Descent (SGD) is used to update the value for learnable parameters. At the beginning, the learning rate is set to 0.0001, after each 22000 iterations, the learning rate is dropped as 10^{-3}. Besides, 90% value of previous computation will be retained in the new calculation.

9.4 Experiment and results

The network is trained with the data that chosen in three ways. In each way, two kinds of datasets are used: color patches and grayscale patches. For each dataset, three patch's size is used to prepare the data: 27×27, 45×45 and 63×63. The accuracy of the network on each dataset is provided in the tables following:

Size of patches	27×27	45×45	63×63
Accuracy on color patches	73	83.14	88.97
Accuracy on grayscale patches	57.25	72.08	77.91

Table 9.1: The accuracy of the model on the data that chosen by the first way.

Size of patches	27×27	45×45	63×63
Accuracy on color patches	75.193	85.417	89.168
Accuracy on grayscale patches	64.385	72.11	78.1077

Table 9.2: The accuracy of the model on the data that chosen by the second way.

As the result shown in three tables (table. 9.1, 9.2, 9.3), it seems that the network did not have more sensitive with the way to choose the data. But we have the different result from color and grayscale patches and the result is also improved when we increase the size of the patches from 27×27 to 63×63.

From the accuracy of training on each dataset, the patches of size 63×63 which selected by randomly are used to predict. The prediction is run on 33 images ($33 \times 8 = 264$ patches). The label of a patch is label with highest prediction. Table 10.1 shows the statistic of prediction on

Size of patches	27×27	45×45	63×63
Accuracy on color patches	77.286	87.751	89.979
Accuracy on grayscale patches	64.163	76.455	82.709

Table 9.3: The accuracy of the model on the data that chosen by the third way.

	LM 1	LM 2	LM 3	LM 4	LM 5	LM 6	LM 7	LM 8
LM 1	87.88%	6.06%	0.00%	0.00%	0.00%	0.00%	0.00%	6.06%
LM 2	6.06%	90.91%	3.03%	0.00%	0.00%	0.00%	0.00%	0.00%
LM 3	0.00%	3.03%	87.88%	3.03%	0.00%	0.00%	6.06%	0.00%
LM 4	9.09%	0.00%	6.06%	81.82%	0.00%	0.00%	3.03%	0.00%
LM 5	51.52%	0.00%	0.00%	9.09%	39.39%	0.00%	0.00%	0.00%
LM 6	12.12%	3.03%	0.00%	0.00%	0.00%	81.82%	0.00%	3.03%
LM 7	3.03%	0.00%	9.09%	0.00%	0.00%	0.00%	84.85%	3.03%
LM 8	18.18%	0.00%	0.00%	0.00%	0.00%	0.00%	3.03%	78.79%

Table 9.4: The prediction accuracy on the color patches of size 63×63

the dataset. Followed that, most of the landmarks are predicted with high accuracy ($\geq 78\%$). The proportion on 5^{th} landmark is not hight and it has a confusion with the 1^{st} landmark.

9.5 Conclusion

In the content of this work, a model in CNN is proposed to predict the class of the landmarks on pronotum. The network consist 8 layers: 3 CONVs, 3 POOL, 2 FC. For each landmark, the patches with different size are extracted to use as the input data of the model. The experiment shown that the model is worked well on the color batches with larger size. However, a confusion also exists when the model try to predict 5^{th} landmark.

Chapter 10

Deep learning datasets for landmarking

In deep learning, the dataset is a most of important field that contribute to the success of deep learning. The collections the data is depend on the field of project and the size of the database has the effects to the result of training and testing processes. This chapter presents the deep learning datasets, specially, the biological datasets for landmarking detection in 2D images.

10.1 Facial keypoints problem

Face alignment is the first step in face recognition tasks. In which using geometric landmark localization has shown the effectiveness and improving the recognition results. Depending on the objective of the project, the number of collected samples and the number of landmarks on an image may be less or more. Due to the supporting for landmarking problems with the neural network, we present the datasets that have been built for this purpose. Most often the datasets are built on the human face along with some other on the animal.

10.1.1 The Annotated Facial Landmarks in the Wild dataset

Annotated Facial Landmarks in the Wild (AFLW) [36] is a dataset of annotated face images gathered from Flickr. The images recognize the human face with difference information such as, pose, expression, age, gener. The dataset is tagged as 59% female's faces, 41% male's faces; some images contains multiple faces. Totally, AFLW contains 25000 annotated face images with up to 21 landmarks per image. This database is public at AFLW site[1] under supporting of the FFG project MDL and SECRET (Austrian Security Research Programme KIRAS).

10.1.2 Multi-Task Facial Landmark (MTFL) dataset

Multi-Task Facial Landmark (MTFL) has been trained on the same database with [37]. It includes 5590 LFW images, 1521 BioID images, 781 LFPW training images, 249 LFPW test images and 7876 other images dowloaded from the web. The images have been divided into training (13466 images) and testing set (2552 images). During training, each face in the image is labeled with five key points.

10.1.3 Facial Keypoints Detection Kaggle Challenge

The dataset has been built for a challenge of facial keypoints detection. It includes two sets of images: training set contains 7049 images; testing set includes 1783 images. The size of images in the dataset was kept as 96×96 pixels. All the information of images has been saved into

[1] https://www.tugraz.at/institute/icg/research/team-bischof/lrs/downloads/aflw/

CSV files. In training set, each row of CSV file contains the coordinates (x, y) for 15 key points and the image data as row ordered list of pixels. In testing set, each csv row contains the image identification and image data as row ordered list of pixels. The dataset has been published at Kaggle website[2].

10.1.4 Cat dataset

CAT dataset [] includes more than 9000 cat images. For each images, a set of 9 points are stored. They are included two for eyes, one for mouth and six for ears. This dataset used to train a neural network for detecting the head of cat

10.2 Other problems

10.2.1 Syntheseyes dataset

Erroll Wood et al [38] proposed synthesizing perfectly labelled photo-realsitic training data in a fraction of the time. The eye-regions models have been built from head scan geometry by using computer graphics techniques. The models were randomly posed to synthesize close-up eye images for a wide range of head poses, gaze directions and illumination conditions. For each image, each 3D eye-region landmarks with 28 landmarks have been set manually, corresponding to the eyelids (12 landmarks), iris boundary (8 landmarks), and pupil boundary (8 landmarks). The database is exists at SynthesEyes[3].

10.2.2 Drosophila Wings dataset

Drosophila Wing (DW) datasets [39] was constructed from Drosophila melanogaster wings for the development, testing measurement, and classification tools for biological images. The images are individually identified and organized by sex, genotype, and replicate imaging system. DW database contains a large number of wing images representing multiple genotypes. Each wing was imaged at 20× and 40× magnification on an Olympus BX51 and Leica M125 microscope. The landmarks and semi-landmarks data have been extracted by WINGMACHINE. In total, this included 12 landmarks and 36 semi-landmarks. The images and landmarks can download at GigaScience website[4].

10.3 Summary

The summary of all datasets are shown in the following table:

[2]https://www.kaggle.com/c/facial-keypoints-detection/data
[3]https://www.cl.cam.ac.uk/research/rainbow/projects/syntheseyes/
[4]http://gigadb.org/dataset/100141

Name of databse	N^o images	Size of image	N^o landmarks	Project
AFLW	25000	–	21	No
MTFL	16018	250×250	5	No
FKDKC	8832	96×96	15	Challenge
SynthesEyes	11382	120×80	28(3D)	Yes
DW	8948	–	48	-

Table 10.1: The summary of deep learning datasets

Chapter 11

Automatic extraction the morphometry landmarks by Convolutional Neural Network

In this work, we study the two methods that used to detect the landmarks on 2D images: **Deep Convolutional Network Cascade for Facial Point Detection** proposed by Yi Sun et al[37] and **Automatic ear detection and feature extraction using Geometric Morphometrics and Convolutional neural networks** from Celia Cintas et al[40]. Both articles have shown the convolution neural network (CNN) to study the landmarks on 2D images: Yi Sun studied the human face while Celia worked on human ears. Then, we show the results when applying the methods on pronotum of beetle. Finally, we propose a specific network to work on pronotum and compare the results.

11.1 Model 1: Facial point detection by CNN

Yi Sun et al[37] focused on five facial keypoints: *left eye center*(LE), *right eye center*(RE), *nose tip*(N), *left mouth corner*(LM) and *right mouth corner*(RM) (called landmarks). A model with 3-levels of networks are proposed to study from high-level to low-level of the landmarks.

11.1.1 Data

The dataset with 13466 face images includes 5590 images from LFW [41] and remaining images are downloaded from the web. The dataset is randomly divided into the training set with 10000 images and validation set with 33466 images. Each face is labeled with five landmarks and the bounding box is created around the face.

11.1.2 Architecture

The proposed architecture includes 3-levels of CNN: three networks at the first level, and ten networks for each remaining level(Fig.11.1). The networks at level 1 is designed to detect multiple landmarks while two last levels are designed for working on each landmark.

At the first level, three CNNs are employed to study the location of the facial points: F1, EN1, NM1 whose input regions cover the whole face. F1 is studied all the position of five landmarks; EN1 is worked on the eyes and nose while NM1 worked on nose and the mouth corners. Each network predicts the landmarks corresponding with the region that it covers. At the end of level 1, the coordinate of each landmark is averaged of coordinates that predicted from three networks. Fig.11.2 illustrates the deep structure of the networks at level 1, which

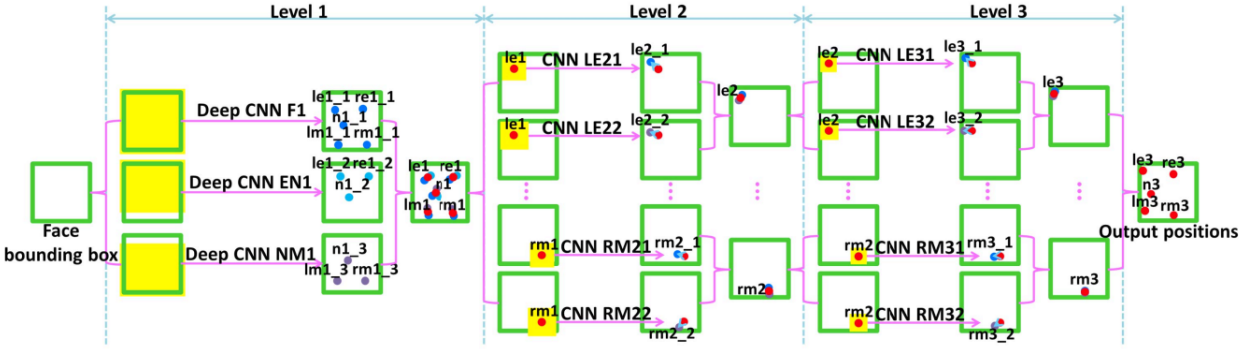


Figure 11.1: The 3-levels proposed architecture

contains four convolutional layers followed by max pooling and two dense layers. F1, EN1, and NM1 take the same structure but with different size of the input and different output at full-connected layers to suitable with the number of predicted landmarks.

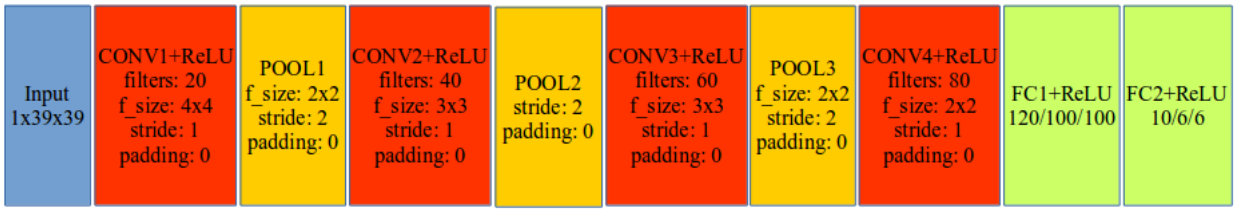


Figure 11.2: The structure of the networks in level 1

The networks at the second and third levels take local patches centered at the predicted position of previous levels as input. Besides, they allowed to make small changes to previous prediction. The size of patches are also reduced along with the cascade model. For each position, two networks are used to predict the new positions. The last predicted position is average of the new positions. Fig.11.3 illustrates the architecture of the networks in level 2 and level 3. Basically, the networks in last two levels are similar, the only difference is the way to choose the patch around the landmark. A padding is added to the coordinates of the landmark to make the change of the patches i.e $0.16, 0.18$ in level 2 and $0.11, 0.12$ in level 3. Then, the patch is resized to the size of 15×15 before giving the networks.

Figure 11.3: The structure of the networks in level 2, level 3

With 3-levels model, the purpose of the networks at the first level are estimated the landmark positions with large errors; the networks at last two levels are designed to achieve high accuracy.

11.1.3 Experiments

Training

At the first level, $F1$ takes the whole face as input (size of 39×39 of the network and outputs the position of all the five points. $EN1$ takes the top and middle part of face as input (size of 31×39) and outputs the positions of two eye centers and nose tip. $NM1$ takes the bottom and middle part of the face to predict the positions of nose tip and two corners of mouth.

All the networks at level 2 and level 3 take a small squares (15×15)centered at the predicted position by the previous level as the input and output the incremental prediction. The last predicted positions at each level are average of corresponding positions from all the networks in each level.

During training, the size of the patches is decreased for each level. The learnable parameters include weight w, the gain g and the bias b which are initialized by small random number and learned by stochastic gradient descent.

The detection error on each facial point is measured by Eq.11.1. If the error is greater than 5%, it is considered as failure.

$$err = \sqrt{(x - x')^2 + (y - y')^2}/l \tag{11.1}$$

Where:

- l is the width of the bounding box around the face.
- (x, y) is ground truth facial point
- (x', y') is predicted position

Testing

The model is tested with a dataset of 2557 face images. The image with the bounding box of the face is used as the input of the model. At the end, the predicted position is estimated from the model. By using the way in Eq.(11.1) to evaluate the model, the error statistic on each level is obtained (Figs. 11.4, 11.5, 11.6).

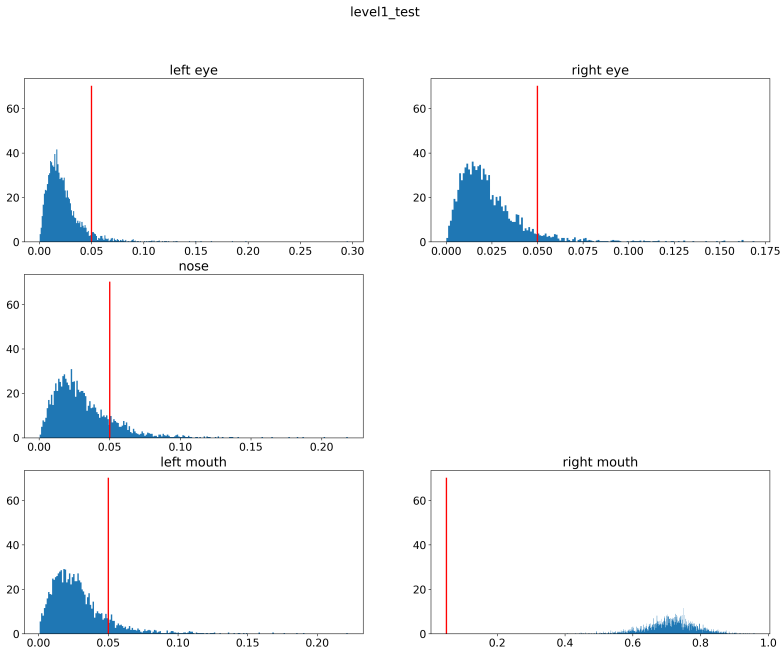


Figure 11.4: The error on each landmark in level 1

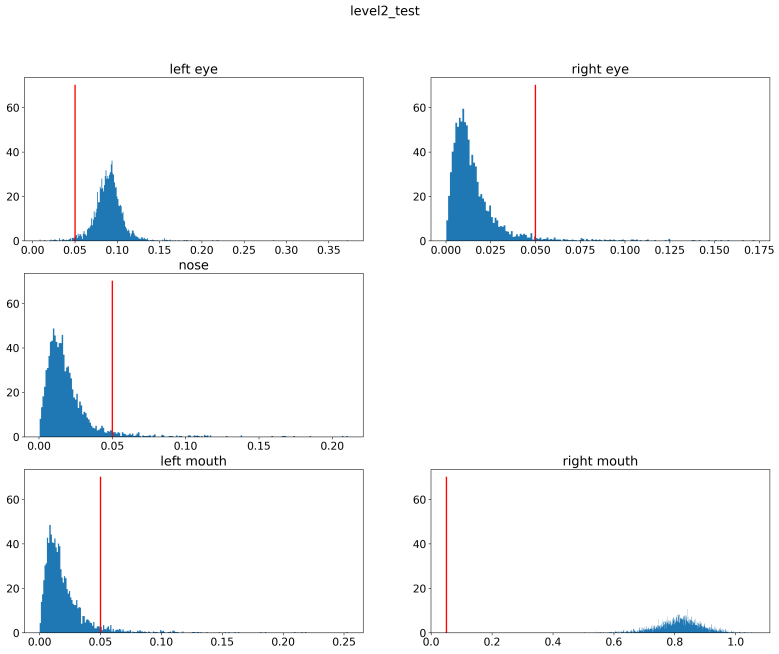


Figure 11.5: The error on each landmark in level 2

11.2 Model 2: Automatic ear landmarks detection by CNN

Celia Cintas et al[40] proposed a method based on geometric morphometric and deep learning for automatic ear detection and feature extraction in the form of landmarks. The convolutional neural network was trained with a set of manually landmarks examples. The network is able to provide the morphometric landmarks on ear image automatically.

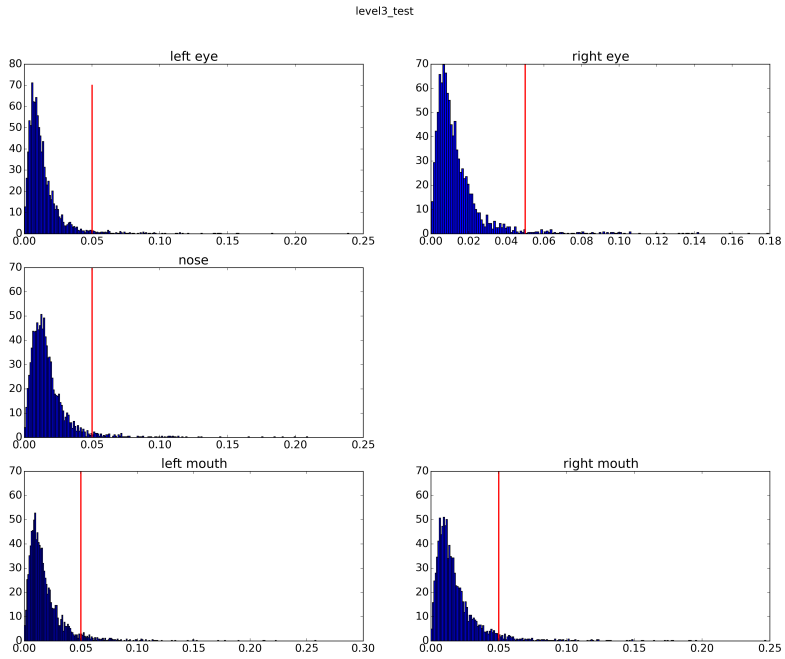


Figure 11.6: The error on each landmark in level 3

11.2.1 Dataset

The image and manual landmarks belong to the CANDELA initiative [1], a project includes geneticists, bioinformatics and social-anthropologists interested on Latin American. CANDELA contains 7500 images with the size of 2136×3216. The provided dataset contains 2753 images which extracted from the CANDELA dataset. For each image, a set of 45 landmarks and semi-landmarks provided by human operators. The dataset was split into a training set with 2051 images (75%) and a validation set of 684 images (25%).

11.2.2 Network

Three models were designed and trained for performing the automatic landmarks task. These architectures are different in the number of convolution layers, the filter sizes, and the learning rate. An image with a single channel of the size 96×96 with brightness scaled to $[0, 1]$, is taken as the input of the network. The target (landmarks coordinates) is scaled to $[-1, 1]$. **Fig.11.7** shows the best architecture. In this architecture, a structure of two convolutional layers with the filters, followed by maximum pooling and dropout layer. This structure is repeated three times to obtain features at different levels with different size of filters(i.e 4×4 and 3×3). After extraction the features, two fully connected linear layers with 1500 units each and a dropout layer is hired. The output layer contains 90 output units corresponding with 45 landmarks for the predicted position of the landmarks. The implementation used Python and the Lasagne library[42].

11.2.3 Experiments

Because the CANDELA dataset is not published. Another dataset was chosen to study the network. The new dataset was used for the Facial Keypoint Detection including 7049 gray-scale images (96×96). For each image, we are supported learn to find the position of 15 landmarks.

[1] https://www.ucl.ac.uk/candela

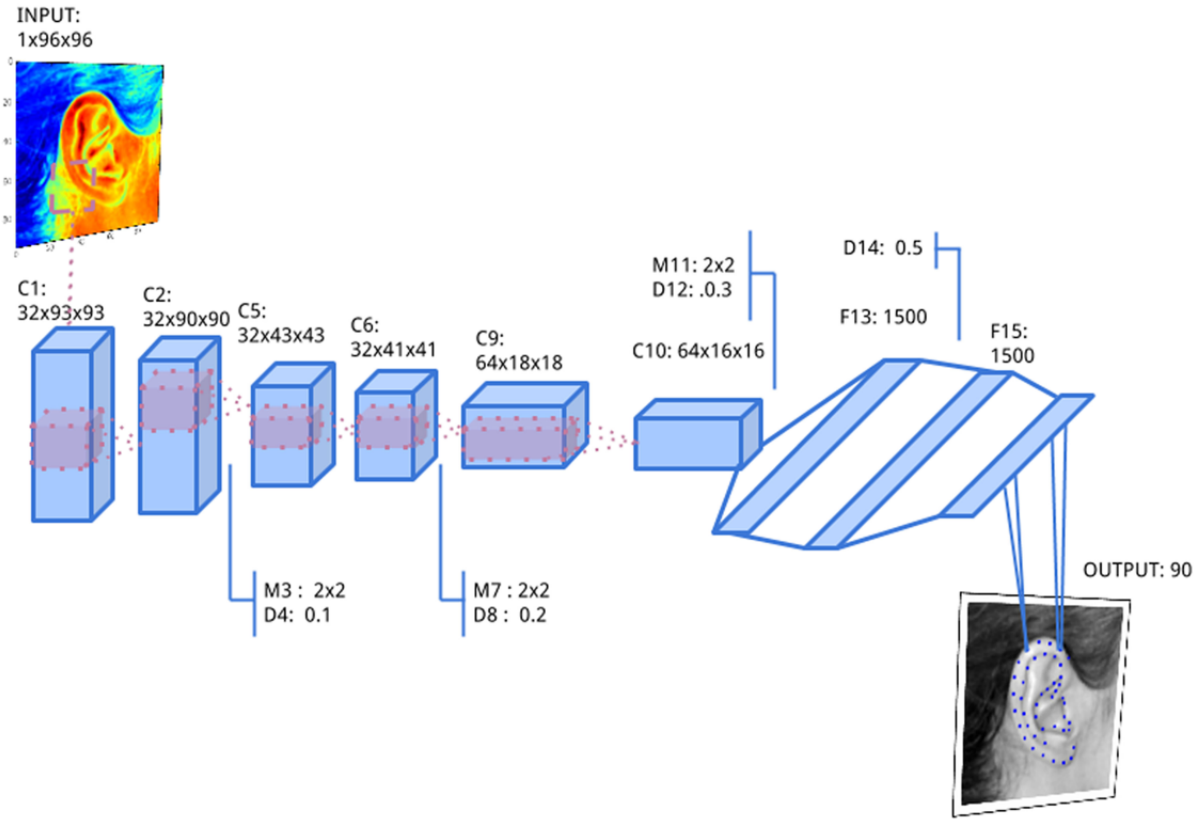


Figure 11.7: The best architecture for automatic ear's landmarks detection

After dropping some missing data, the dataset remains 2140 images. All the images with coordinates of manual landmarks is stored in csv file and fetched into the network. During the training and validation, the usual quality metrics for regression problems is used, in particular, root mean square error (RMSE). Fig.11.8 shows the learning curves of the model on training and validation set error. The training is finished after 3000 iterations with the loss arround 10^{-3}. Fig.11.9 shows some test on the real facial images.

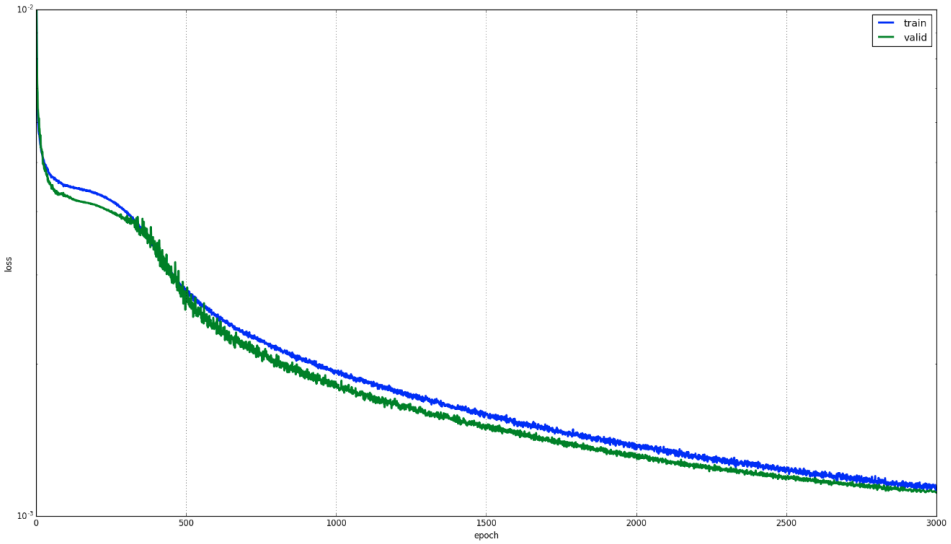


Figure 11.8: The loss of Ear-CNN network on facial point dataset

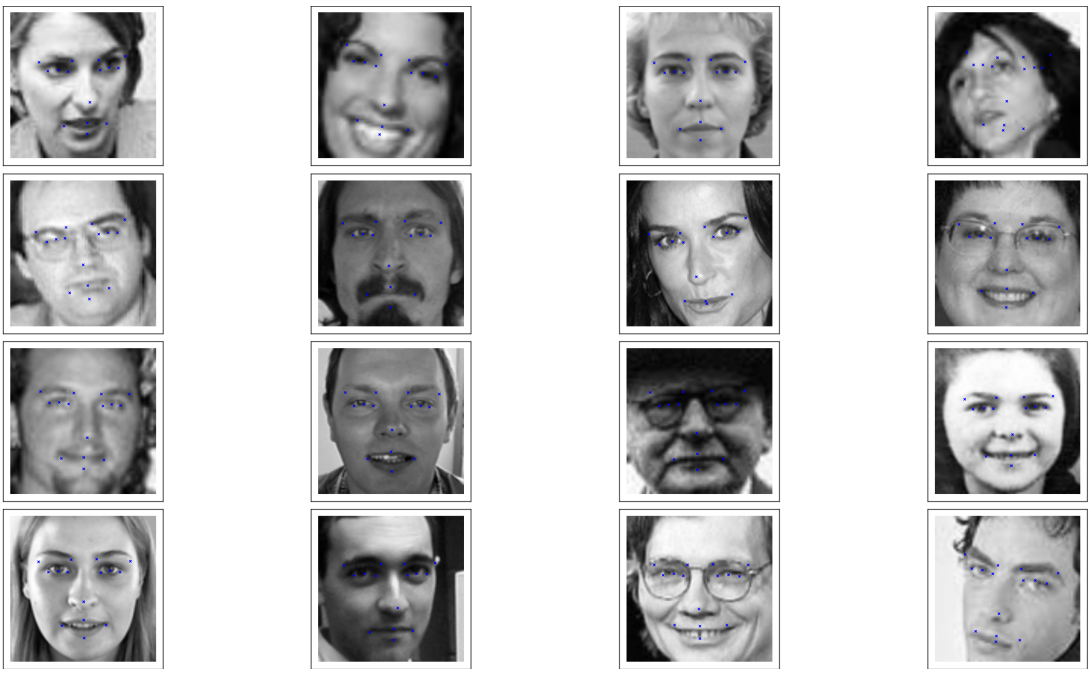


Figure 11.9: The prediction landmarks on human face

11.3 Model 1 and model 2 on pronotum

11.3.1 Dataset preparing

The dataset includes 293 pronotum images. The images are divided into three subsets: the training set (200 images), the validation set (60 images) and the testing set (33 images). Because the dataset is limited and the models are worked on gray-scale images, we applied some ways to en-large the dataset. Firstly, for each original image in RGB, each channel is modified by adding some values. Secondly, the channels of the original image are split. So, we have obtained 1400 images for the training set and 420 images for the validation set. At the end, the images are down-sampled with the size of 256×192 before giving to the networks.

11.3.2 Model 1 and pronotum landmarks

The networks in the first level are modified to suitable with the prediction of landmarks on the pronotum (8-landmarks). For each pronotum, eight manual landmarks have been set. The bounding box is created depending on the coordinate of the manual landmark and kept with the same size. The networks in the first level are used as followed:

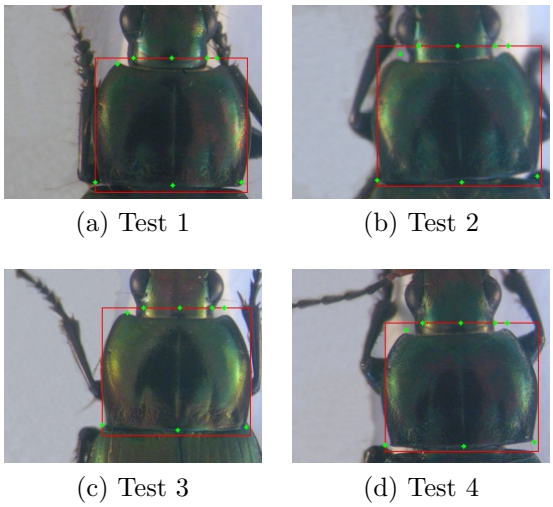
- F1 network recognizes whole pronotum bounding box with eight landmarks.
- EN1 network predicts the location of the first five-landmarks i.e [1..5].
- NM1 network is used to estimated the position of last four-landmarks and the first landmark i.e $[5..8, 1]$.
- At the end, the position of each landmark is average of the predicted position in the networks.

Testing

During training, the Euclidean distance (sum of squares) is used to compute the loss of the networks. The error rate of each network during training is shown in the Table.11.1:

Network	Loss
F1	0.013
EN1	0.47
NM1	0.5

Table 11.1: The loss of the networks in Model 1 on pronotum dataset



(a) Test 1 (b) Test 2

(c) Test 3 (d) Test 4

Figure 11.10: The pronotum with predicted landmarks at level 1

From Table 11.1, the errors of EN1 and NM1 are still high. That errors make the prediction result of level 1 do not enough good. Besides, the networks at level 2 and level 3 used the prediction at level 1 as the input data to predict the new position. So, we can not continue with the level 2, 3 until the result at level 1 is improved. Perhaps, the model in model 1 is not suitable to detect the landmarks on pronotum. Fig. 11.10 shows the prediction landmarks on four images. Followed that, the networks can detect the landmarks at positions 4, 5 and 7; the different positions still not good.

11.3.3 Model 2 and pronotum landmarks

The dataset is kept the same with model 1 (1400 images for training and 420 images for validation) but having some changes. Firstly, the ways to choose the data(to train and validate) is changed. All images are combined. Then, the network will automatically choose 75% data to train and 25% for validation. Secondly, the inputs that given to the network are just the image and landmarks(without the coordinates of the bounding box).

The network is run 3000 iterations with the learning rate begin from 0.08 to 0.01. During training, the learning rate is changed to fit with the remaining iterations[43]. Fig.11.11 shows the first 700 iterations during training. The loss did not have many changes after 100^{th} iteration.

Fig.11.12 shows the prediction landmarks on 16 images. Following, the prediction landmarks from the network of model 2 are closed with the pronotum but the location is still inaccurate.

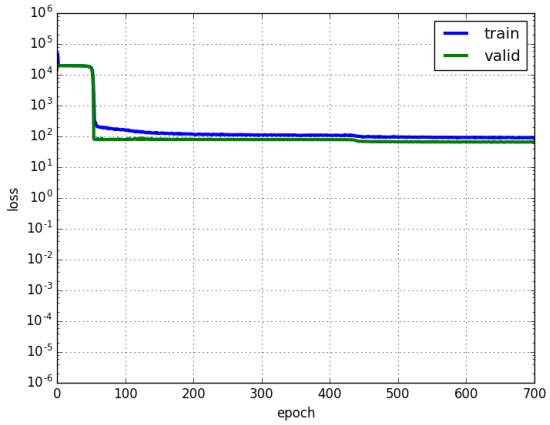


Figure 11.11: The losses of model 2 on pronotum dataset

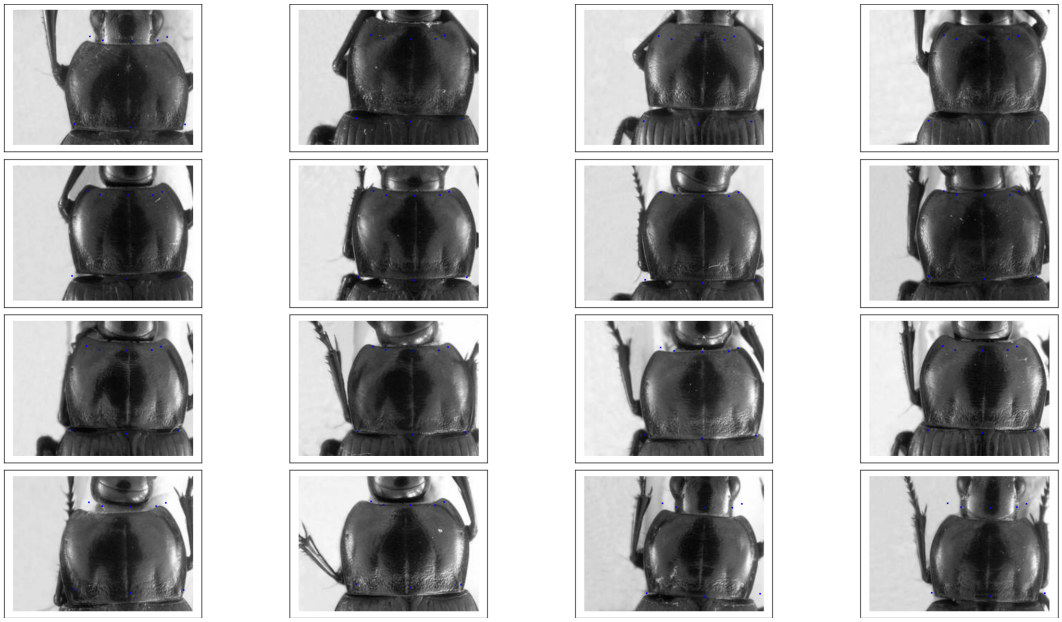


Figure 11.12: The prediction landmarks on pronotum of model 2

11.4 Proposed architecture (model 3)

11.4.1 Model and parameters

From the tutorial of Daniel Nouri[2] about using CNN to detect facial key points. We propose a CNN to detect the landmarks on pronotum. The proposed network includes three convolutional layers followed by three maximum pooling layers and three full connected layers(Fig.11.13). The network receives the gray-scale image (256×192) as the input. The deep of convolutional layers is increased from $32, 64$, to 128 with different size of filter. The size of filters in pooling layers are kept in the same size of 2×2. At the end of network, three full-connected layers with the size of $500, 500$, and 16 are set up to predict the positions of landmarks. Besides, the model is designed with a small sharing learning-rate and the momentum. The learning-rate and the momentum are changed overtime of training.

[2]http://danielnouri.org/

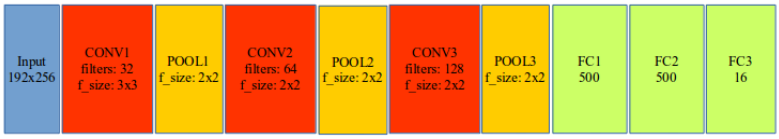


Figure 11.13: The architecture of proposed model

11.4.2 Training and experiments

The model is trained with 1820 images in 5000 iterations. The images are normalized before giving to the network by scaling the intensity value to [0,1], instead of 0 to 255. The target values (x and y coordinates) is kept as original. During the training, the root-mean-square error (MSE) is used to calculate the loss.

To evaluate the stability and confidence of the model, we proposed several rules to select the images for the network, as Table.11.2:

Rule	Test set	Training and validation set
rule_1	From 1^{st} image to 33^{rd} image	Remaining images
rule_2	From 260^{th} image to 293^{rd} image	Remaining images
rule_3	Random	Random
rule_4	From 90^{th} image to 122^{nd} image	Remaining images
rule_5	From 200^{th} image to 232^{nd} image	Remaining images

Table 11.2: The rules to choose the data for the network

Following the rules to choose the data, the loss of training and validation is shown in Table 11.3. From the results in the table, the training losses in the cases of *rule_4, rule_5* are smaller than other rules; but the validation losses are stability. It means the overfitting is appeared clearly in the case of *rule_4* and *rule_5*. In which, the smallest difference value between training and validation loss is belong to **random** case.

Rule	Training loss	Validation loss
rule_1	0.12739	0.63681
rule_2	0.15204	0.59480
rule_3	0.16694	0.55584
rule_4	0.08798	0.61934
rule_5	0.0918	0.52843

Table 11.3: The training loss and validation loss following each rule to choose the data

Fig.11.14 shows the training curve loss and validation curve loss of the model on each rule to choose the data. From the beginning, the loss is not changed. When the training is longer and the learning rate is improved, the loss is decreased and a large distance between training and validation is appeared (over-fitting) but it is not that bad.

The model is tested on the test datasets(five rules). Then, correlation between the manual landmarks and predicted landmarks is computed by applying the correlation methods (see Table.11.4, 11.5, 11.6)

Rule	x correlation	y correlation
rule_1	0.9953877	0.9941767
rule_2	0.9968787	0.9960827
rule_3	0.9966784	0.9957729
rule_4	0.9975662	0.9985097
rule_5	0.9972048	0.9976416

Table 11.4: The correlation between manual and predicted landmarks by Pearson[44] method

Rule	x correlation	y correlation
rule_1	0.9893943	0.9289319
rule_2	0.992556	0.9444423
rule_3	0.9913126	0.9565425
rule_4	0.9943106	0.9789221
rule_5	0.9920646	0.9864683

Table 11.5: The correlation between manual and predicted landmarks by Spearman[45] method

Rule	x correlation	y correlation
rule_1	0.913517	0.7498531
rule_2	0.9303295	0.8231899
rule_3	0.9273002	0.8273057
rule_4	0.9419902	0.8904413
rule_5	0.9299128	0.9051508

Table 11.6: The correlation between manual and predicted landmarks by Kendall[46] method

Fig.11.15 show the predicted positions on test dataset followed rule_3:

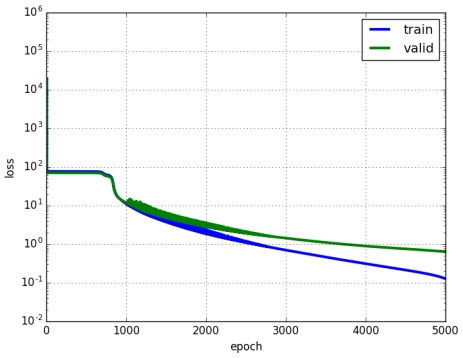
The network in model 3 is applied to predict the landmarks on tete and elytre with some modifications such as: the output at the last of full connected layer. The data that used to train and test are chosen by applying rule_3. Table.11.7,11.8 shown the statistic on the test set of tete and elytre by different correlation coefficient methods:

Method	x correlation	y correlation
Pearson	0.99082758	0.988833
Spearman	0.9868292	0.989447
Kendall	0.905933	0.9130359

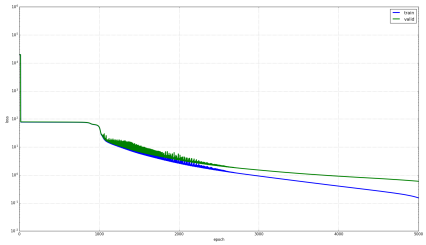
Table 11.7: The correlation between manual and predicted landmarks on tete images

From the result of Table 11.3, the dataset which was chosen by rule_3 is used to continue the experiments. To improve the results, we have modified the rule to pre-process the input data: the target (coordinates of manual landmarks) is normalized into the range of $[-1, 1]$, instead of $[0, 256]$ for x-coordinate and $[0, 192]$ for y-coordinate. Additional, the *learning rate* of the network has been changed to increase the speed of training process.

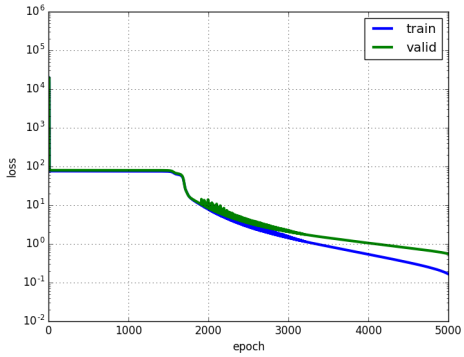
After changing, the result that we obtain as we expect: training loss and validation loss have



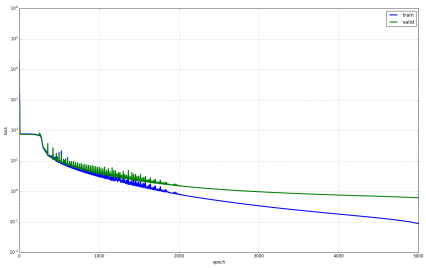
(a) rule_1



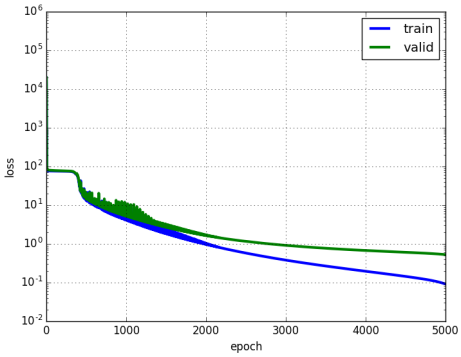
(b) rule_2



(c) rule_3



(d) rule_4



(e) rule_5

Figure 11.14: The loss during training and validation followed each rule to choose data

Method	x correlation	y correlation
Pearson	0.984969	0.9976646
Spearman	0.9840307	0.966091
Kendall	0.8996806	0.846849

Table 11.8: The correlation between manual and predicted landmarks on elytre images

decreased significantly(training loss: **0.00002**, validation loss: **0.00012**). Thus, the landmarks coordinates have been normalized into $[-1, 1]$, so, to calculate the MSE error, we will take the square error and multiply by scale ratio again: the error is arround 1.4(validation loss). Fig.11.16 shows the losses during training and validation. We can see that overfitting have appeared during the train and validation processes.

To prevent the overfitting on the network, we have modified both data and model: on data

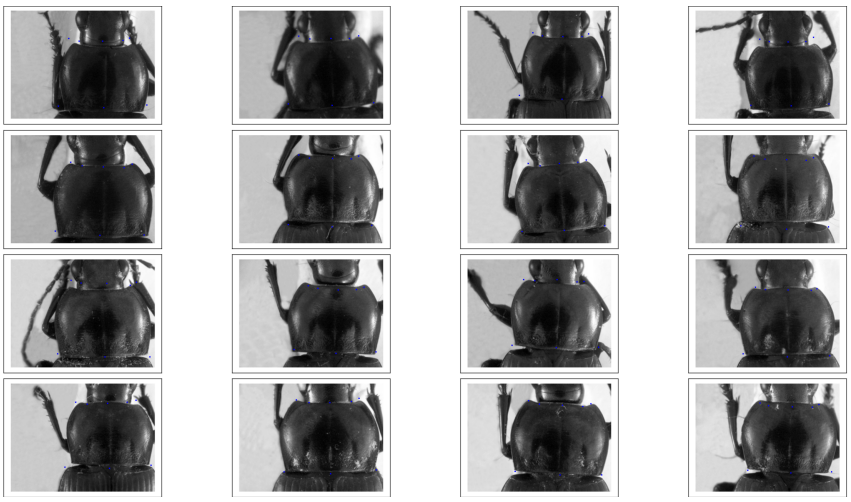


Figure 11.15: The prediction landmarks on 16-pronotum images

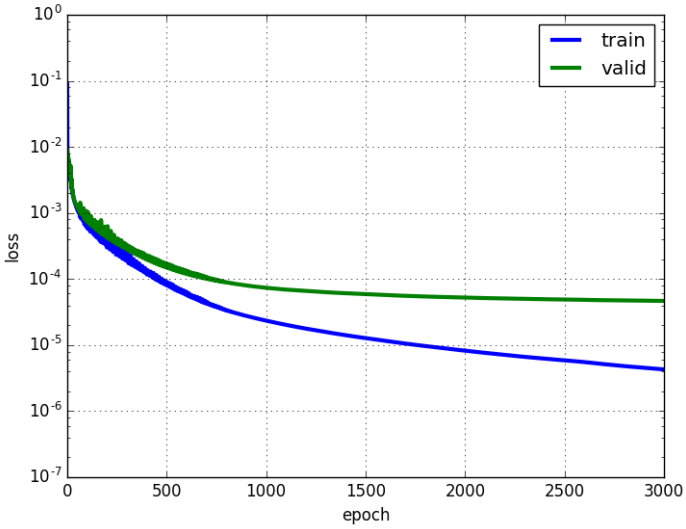


Figure 11.16: The training and validation loss when normalize the landmarks coordinates

side, we have changed the *split* ratio to get more samples for validation set(40% instead of 20% of number of samples); on model side, the number of units in the last two hidden layers (full-connected layer) are increased from 500 to 1000. Besides, four dropout layers have been added into the network. They have been located following the pooling layers and the first full-connected layer. The dropout ratios are 0.1, 0.2, 0.3 and 0.5 (Fig.11.17).

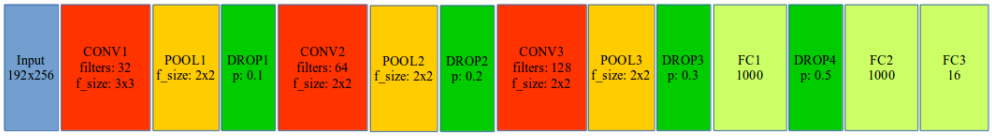


Figure 11.17: The proposed model with dropout layers

Fig.11.18 shows the losses after the model have been modified. The overfitting problem is solved but the losses are stability from 2000^{th} until the end(we need more data). Table.11.9 shows the correlation coefficient of new model. Clearly that, the result have been improved a little bit when we compare with the last result.

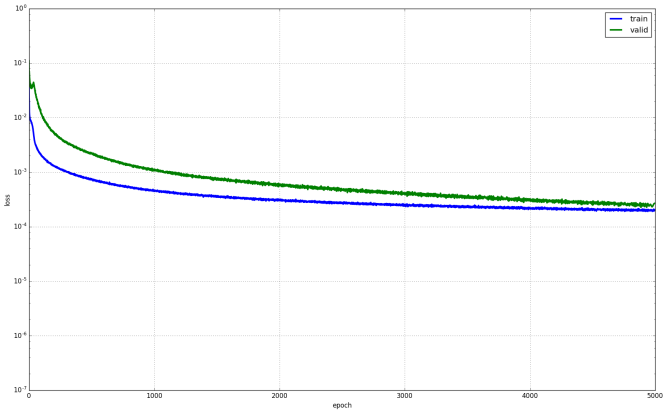


Figure 11.18: The training and validation loss with dropout layers

Method	x correlation	y correlation
Pearson	0.9970585	0.9978605
Spearman	0.9942475	0.9859642
Kendall	0.9430501	0.9067739

Table 11.9: The correlation between manual and predicted landmarks on pronotum images with new model

11.5 Conclusions

In this studied, two methods that used to predict the landmarks on 2D gray-scale images are studied. For each case, the model is suitable with different dataset but the results are still not good when we change the data (pronotum). Besides, we proposed a network to learn and detect the landmark positions on pronotum. The accuracy of the model is greater than 98%. The results are evaluated on 3 datasets: pronotum, tete and elytre. From the correlation coefficients of each dataset shows that if we consider on the statistic side, the coefficients are enough good to precise. But when we see the real position on each image, that is not good as we expect. It means the model is not really suitable for the problem and we need to improve the model.

Chapter 12

Convolutional Neural Network for predicting morphometry landmarks

In the previous chapter, we have studied two CNNs [37, 40] that applied to predict the keypoints on human face and human ear. Then, we have applied their methods to detect the landmarks (keypoints) on thorax of beetle but the obtained results are not really impressed. In this chapter, based on the knowledge of Deep Learning and CNN, we propose a new archtecture to predict the landmarks on some parts of beetle. We also describe the process that we have used to augment the dataset for training and evaluating the model.

12.1 Network architecture designing

In the process, we have tried three networks models before obtaining the final architecture for detecting the landmarks on beetle images. Like other CNN models, we have employed the classical layers to construct the models, i.e, convolutional layer, maximum pooling layers and full-connected layer.

The first architecture is very classical one, it receives an image with the size of $1 \times 192 \times 256$ as the input. Then, the network consists on three repeated strucutre of a convolutional layers followed by a maximum pooling layers. Most CNNs, the hyperparameters of convolutional layers have been set to increase the depth of the images from the first layer to the last layer. That is reflected in the setting of the number of filters at each convolutional layer. So, the depths of convolutional layers increase from $32, 64$, and 128 with different size of the kernels: 3×3, 2×2 and 2×2, respectively. Inserting pooling layers after a convolutional layers is a common periodcally. The pooling layer effects to progressively reduce the spatial size of the representation to reduce the number of parameters, computation in the network, and it also controls over-fitting. The operations of pooling layers independent on every depth slice of the input. The most common form is a pooling layer with filters of size (2×2) and a stride of 2. It downsamples every depth by 2 along width and height of the input. Thefore, all the kernels of maximum pooling layers have the same size of 2×2 with a stride of 2 as usual. At the end of the model, three full-connected layers have been added to extract the global relationship between the features and to procedure the outputs. The first of two full-connected layers are set to non-linearity to make sure these nodes interact well and take into account all possible dependencies at the feature level. The outputs of the full-connected layers are $500, 500$ and 16. The output of the last full-connected layer corresponds to the coordinates (x and y) of 8 landmarks which we would like to predict. Fig. 12.1 shows details of the first model: The orange rectangles represent for convolutional layers while the yellow rectangles represent for maximum pooling layers and three full-connected layers with their parameters are presented at the end of the model.

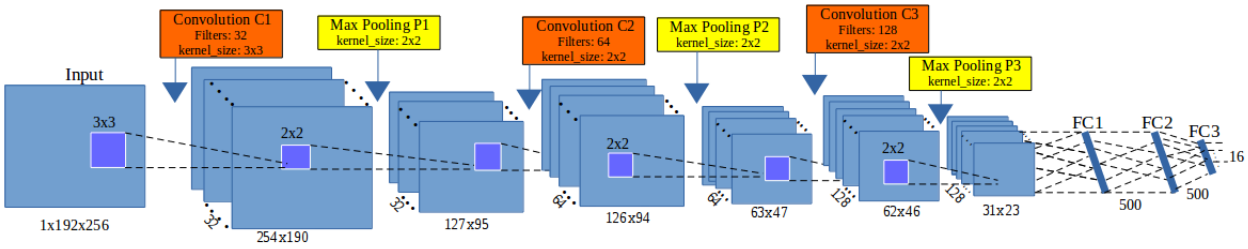


Figure 12.1: The architecture of the first model

The second architecture is modified from the first model. The layers are kept the same as the first one but the outputs of the first of two full-connected layers are changed from 500 (in the first model) to 1000 (Fig. 11.14e). Increasing the value at full-connected layers is hoping to obtain more features from convolutional layer and to prevent the over-fitting.

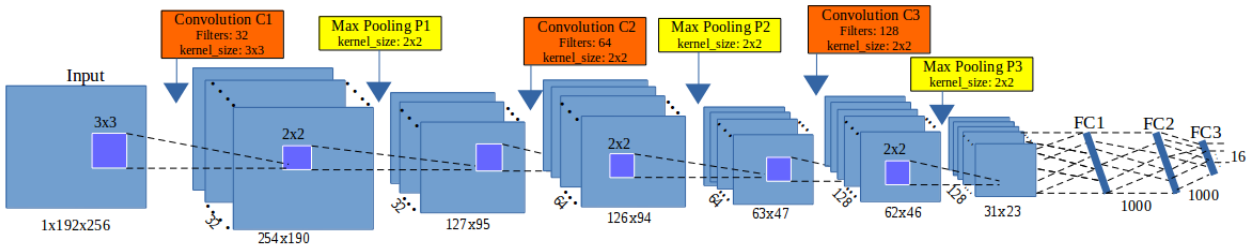


Figure 12.2: The architecture of the second model

To build the third architecture, we have used the definition of *elementary block*. An elementary block is defined as a sequence of convolution (C_i), maximum pooling (P_i) and dropout (D_i) layers. This significantly reduces overfitting and gives major improvements over other regularization methods []. The idea of dropout is to include some variations between different runs. During training phase, dropout samples are done from an exponential number of different "thinned network. At test phase, it is easy to approximate the effect of averaging the prediction of all thinned networks by simply using a single unthinned network with smaller weights. So, we have modified the architecture by combining some *elementary blocks*. Fig. 12.3 illustrates the layers in the third architecture. For our purpose, we have assembled **3 elementary blocks**. The parameters for each layer in each elementary block are as below, the list of values follows the order of elementary blocks ($i = [1..3]$):

- CONV layers:
  - Number of filters: $32, 64$, and 128
  - Kernel filter sizes: $(3 \times 3), (2 \times 2)$, and (2×2)
  - Stride values: $1, 1$, and 1
  - No padding is used for CONV layers
- POOL layers:
  - Kernel filter sizes: $(2 \times 2), (2 \times 2)$, and (2×2)
  - Stride values: $2, 2$, and 2
  - No padding is used for POOL layers
- DROP layers:

– Probabilites: 0.1, 0.2, and 0.3

Three full-connected layers (FC) are kept the same as the second architecture: FC1 and FC2 have 1000 outputs, the last full-connected layer (FC3) has 16 outputs. As usual, a dropout layer is inserted between FC1 and FC2 with a probability equal to 0.5.

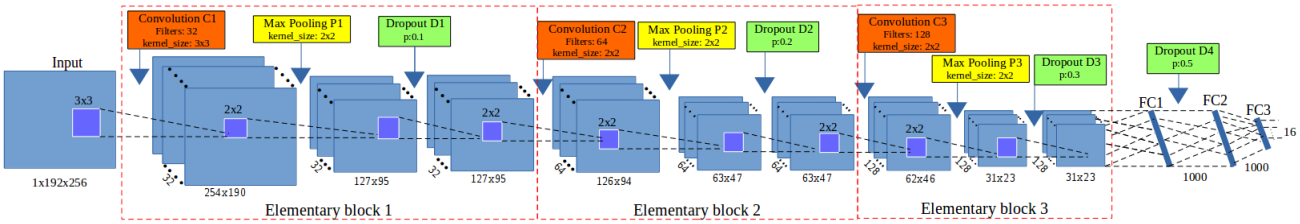


Figure 12.3: The architecture of the third model

The core of neural network is training over iteration. There are many ways to optimize the learning algorithm, but gradient descent [] is currently a good choice to establish the way of optimizing the loss in neural network. The core idea is following the gradient until we statify with the results will remain the same. So, we have chosen gradient descent in the backward phase to update the values of learnable parameters and to increase the accuracy of the network. The networks are designed with a small sharing learning rate and a momentum. The learning rate is initialized at 0.03 and stopped at 0.00001, while the momentum is updated from 0.9 to 0.9999. Their values are updated over training time to fit with the number of epochs [1]. The implementation of the architectures have been done on Lasagne framework [] by Python.

12.2 Data augmentation

A characteristic of machine learning and deep learning is using a volume dataset to train the model. Of course, in practice, we are not always have enough data for training. One way to solve this problem is to create the fake data from real data and to add it to the training set. Dataset augmentation has been a particularly effective technique for a specific problem. For example, in images classification problem, the operations like translating, rotating or scaling the images have also effective. The fake images may be generated by translating (rotating or scaling) in each direction. Besides, injecting noise in the input can also see as a form of data augmentation.

Our dataset includes 293 images of beetles (for each anatomical part). All the images are taken with the same camera in the same condition with a resolution of 3264×2448. Each image has a set of manual landmarks provided by biologists, i.e, each thorax has 8 landmarks, each head has 10 landmarks. Applying CNNs to train each part with a small number of images to reach good results is impossible. So, we need to augment the dataset before training the networks. Firstly, we have found that the original solution of the images (3264×2448) are heavy for the neural network. For performance considerations, in most of CNNs [], the size of the input is limited to 256×256 pixels, so we have decided to down-sampling the images to a new resolution (256×192) (to respect the ratio between x and y). Of course, the coordinates of manual landmarks have been also scaled to fit with the new resolution of the images. In usual way, the transformations have been used to augment the dataset (i.e rotation, translation,...) but the analysis of image by CNN is most often translation and rotation invariant. Therefore, two other procedures have been imaged to increase the number of images in the dataset (256×192).

[1] An epoch is a single pass through the full training set

The first procedure is to change the value of a color channel in the original image to generate a new image. According to that, a constant is added to one of the RGB channels each time it is used for training. Each constant is sampled in a uniform distribution $\in [1, N]$ to obtain a new value caped at 255. For example, Fig. 12.4 shows an example when we added a constant $c = 10$ to each channel of an original image. Following this way, we can generate three version from an image.

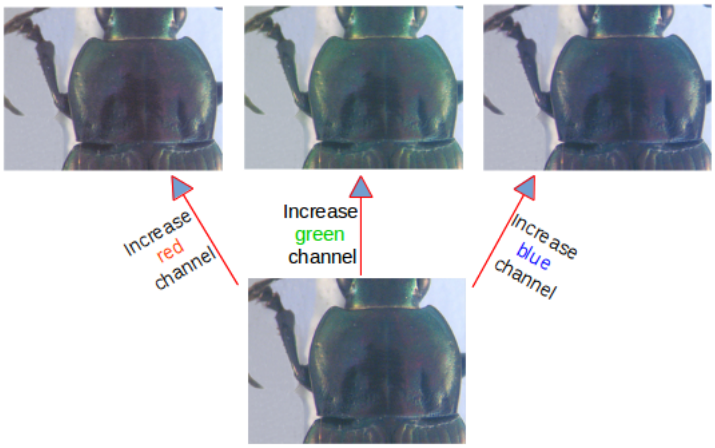


Figure 12.4: A constant $c = 10$ has been added to each channel of an original image

In the second procedure, we have applied the opposite procedure to the first one. Instead of adding the value, we separate the channels of RGN into three gray-scale images as the network works on single channel images. At the end of the processes, we are able generate six versions from an original image. In total, we have $293 \times 7 = 2051$ images for each anatomical part of beetle (an original image and six generated images). However, we have not used all images for training and validation. So, we have chosen 260 original images and their generations (1820 images) of each dataset for training and validation processes, the remaining images (33 original images) are used for test process.

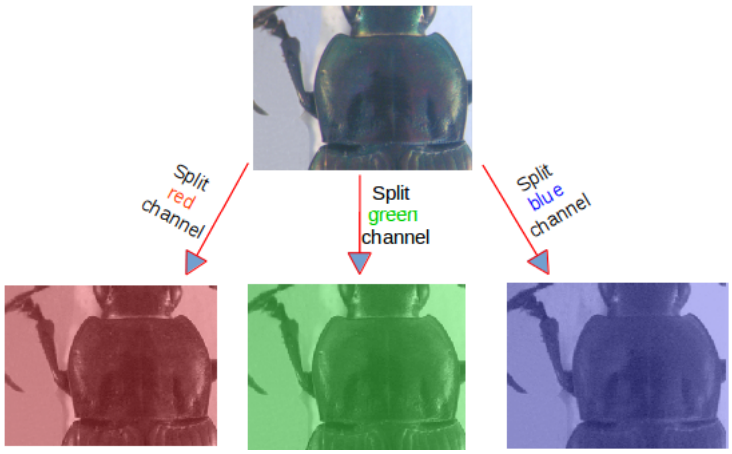


Figure 12.5: Three channels (red, green, blue) are separated from original image

In practical, to obtain a fast convergence during the computing, it is useful to normalize the brightness of the images to $[0, 1]$ instead of $[0, 255]$ and the coordinates of the landmarks have been also normalized.

12.3 Experiments and results

Before widely applying to all anatomical parts, we have firstly tried with thorax part to evaluate the performance. The networks have been trained in $5,000$ epochs on Ubuntu machine by using NVIDIA TITAN X cards. The set of images that used for training and validation are merged together.

During the training, the images are chosen randomly from the dataset with a ratio of 60% for training and 40% for validation. The training step takes into account a pair of information (*images, manual landmarks coordinates*) as training data. In the context of deep learning, landmark prediction can be seen as a regression problem. So, we have used Root Mean Square Error (RMSE) to compute the loss of implemented architectures.

At the test phase, images without landmarks are given to the trained network to produce output coordinates of the predicted landmarks. The results then evaluated by comparing with the manual landmarks coordinates provided by biologists which have been seen as ground truth. Fig. 12.6 shows the training errors and the validation errors during traning phase of the first architecture. The blue curve presents the RMSE errors of training process, the green curve presents the validation errors. Clearly, over-fitting has appeared in the first model. The training losses are able to decrease but the validation losses are stable. In the second model (section 11.14e), we have modified the parameters of full-connected layers to prevent the over-fitting but it seems that this solution is still not suitable. The over-fitting is also appeared as the first model.

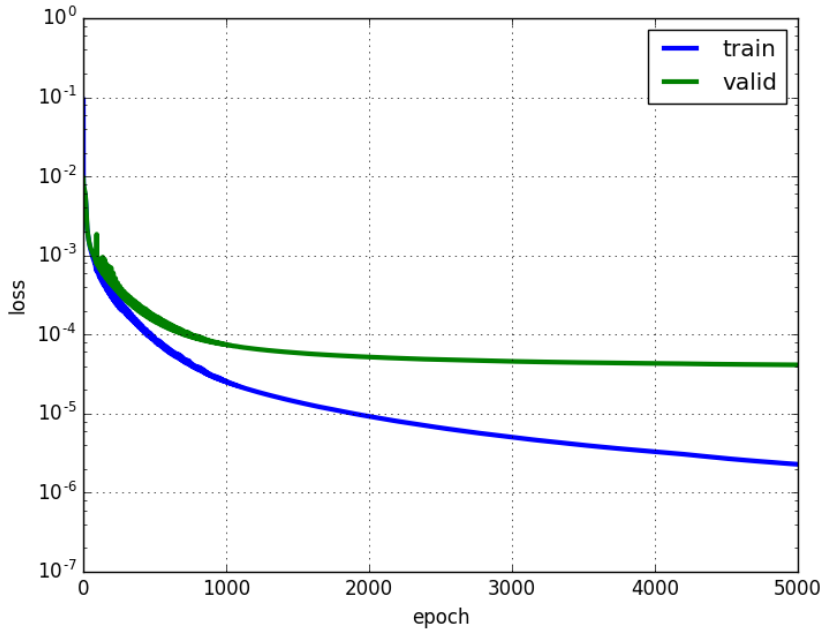


Figure 12.6: The training and validation losses of the first model

Then, we have continued to train the third model on the same dataset of thorax images. Fig. 12.7 illustrates the losses during the training of the third model. Like the previous figure (Fig. 12.6), the blue line is training losses, the green line is validation losses. In the opposite with two previous models, the losses are different (far) from the beginning but after several epochs, the values become more proximate and the over-fitting problem has been solved. This proves that adding dropout layers to build the elementary blocks have been effects to prevent over-fitting and contributory improve the accuracy of the model. *So, we have decided to keep the architecture of the third model for our landmarking problem.*

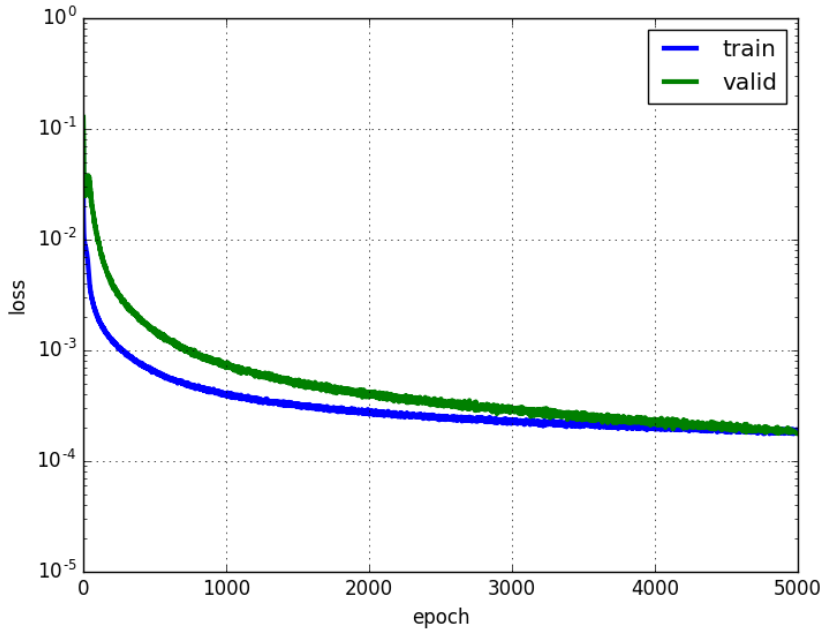


Figure 12.7: The training and validation losses of the third model

In order to have the predicted landmarks for all thorax images (instead of only 33 images), we have applied *cross-validation* to choose the test images, called *round*. For each time, we have chosen a different fold of 33 images as testing images, the remaining images are used as training and validation images ($293/33 \approx 9$ rounds). Following that, the network will be trained with many different datasets, then the trained model will be used to predicted the lanmarks on the images in the corresponding test set. Table. 12.1 resumes the losses of 9 rounds when we trained the third model on thorax images.

Round	Training loss	Validation loss
1	0.00018	0.00019
2	0.00019	0.00021
3	0.00019	0.00026
4	0.00021	0.00029
5	0.00021	0.00029
6	0.00019	0.00018
7	0.00018	0.00018
8	0.00018	0.00021
9	0.00020	0.00027

Table 12.1: The losses during training the third model on thorax images

To evaluate the coordinates of predicted landmarks, the correlation metrics have been computed the correlation between the manual landmarks and their corresponding predicted one. Table. 12.2 shows the correlation scores of 3 metrics (using *scikit-learn* []), i.e, coefficient of determination (r^2), explained variance (EV), and Pearson correlation. All of three metrics have the same possibility. The best score is 1.0 if the correlation data is good, lower values are worse. It means that our predicted coordinates are very close with the ground truth. However, the measure is not enough good to provide a useful result to biologists. Moreover standing on the side of image processing, we are looking forward to seeing the predicted coordinates than the statistical results.

Metric	$\mathbf{r^2}$	**EV**	**Pearson**
Score	**0.9952**	**0.9951**	**0.9974**

Table 12.2: Correlation scores between manual landmarks and predicted landmarks

The main goal of computing is to predict the coordinates of landmarks, so the distances (in pixels) between the coordinates of manual landmarks and corresponding predicted landmarks have been taken into account on all images. Then, the average of distances are computed by landmarks. Table. 11.14e shows the average distances by landmarks on all images of thorax dataset. With images of resolution 256×192, we can consider that an error of 1% corresponds to 2 pixels that could be an acceptable error. Unhappily, our results exhibit average distance of 4 pixels in the best case, landmark 1 and more than 5 pixels, landmark 6. Other error distances are more than 2% pixels.

Landmark	**Distance** (in pixels)
1	4.002
2	4.4831
3	4.2959
4	4.3865
5	4.2925
6	5.3631
7	4.636
8	4.9363

Table 12.3: The average distances on all images per landmark.

Fig. 12.8 shows the distribution of the distances on the first landmark of all images. The accuracy based on the distance in each image can be separated into three spaces: the images have the distance less than average value (4 pixels): 56.66%; the images have the distance from average value to 10 pixels (average distance plus standard deviation): 40.27%; and the images have the distance greater than 10 pixels: 3.07%.

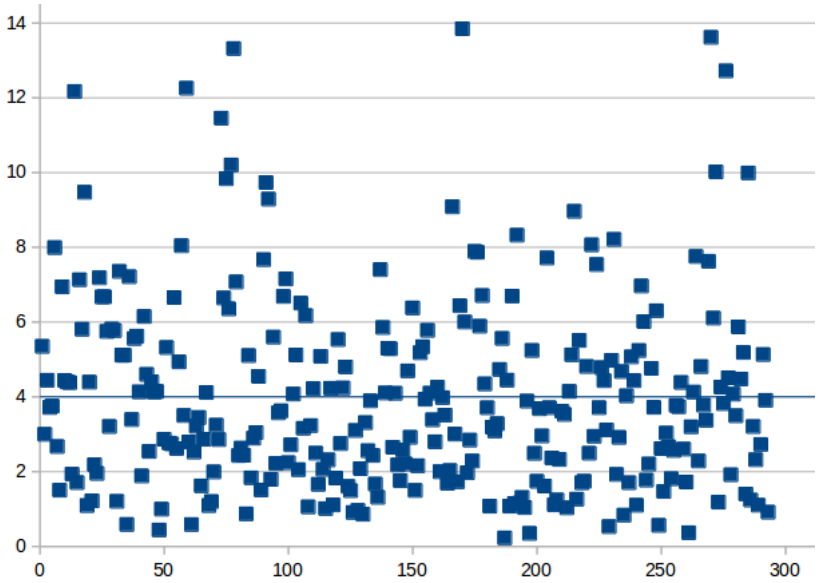
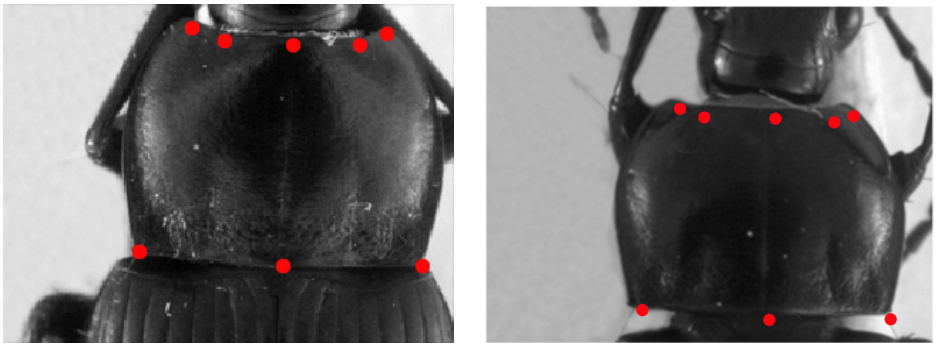


Figure 12.8: The distribution of the distances on the first landmark. The blue line is the average value of all distances.

To illustrate this purpose, Fig. 12.9 shows the predicted landmarks on two test images. One can note that even some predicted landmarks (Fig. 12.11a) are closed to the manual ones, in some case (Fig. 12.11c) the predicted ones are far from the expect results. The next step has been dedicated to the improvement of these results.



(a) Image with well-predicted landmarks

(b) Image with inaccuracy landmarks

Figure 12.9: The predicted landmarks, in red, on the images in test set.

From the success of the third architecture on thorax dataset, we apply the same procedures (data augmentation, training, ...) on other parts of beetle: *elytra and head*. However, we have modified the number of the last full-connected layer to adapt with each dataset before training. Arcoding, the values at the last full-connected layer are set to 22 and 20 outputs corresponds to 11 and 10 landmarks on elytra and head, respectively. Of course, we have also applied *cross-validation* to select testing data to get all predicted landmarks for all images in each dataset. Then, the quality of predicted landmarks are evaluated by comparing with the corresponding manual landmarks (distance computation). Table. 12.4 and 12.5 show the average distances on each landmark of elytra and head anatomical, respectively. The losses during training of both two parts are presented in Appendix 11.14e. Comparing with the average distances on the thorax part, it seems that the proposed architecture provides more accurate predictions on the elytra dataset, but the results are opposite on head dataset.

Landmark	**Distance** (in pixels)
1	3.8669
2	3.9730
3	3.9166
4	3.8673
5	4.0151
6	4.8426
7	5.2125
8	5.4685
9	5.2692
10	4.0709
11	3.9896

Table 12.4: The average distance on all images per landmark on **elytra** images

Landmark	**Distance** (in pixels)
1	5.5280
2	5.1609
3	5.3827
4	5.0345
5	4.8393
6	4.4516
7	4.7937
8	4.5322
9	5.1412
10	5.0564

Table 12.5: The average distance on all images per landmark on **head** images

12.4 Resulting improvement by fine-tuning

The proposed network (third architecture) presented in section 11.14e have been trained from scratch on three datasets (thorax, elytra, and head). At the first step, the network was able to predict the landmarks on the images. But as we have discussed, even if the strength of the correlation seems to validate the results, when we display the predicted landmarks on the images, the quality of the predicted coordinates are also not enough precise, and the average error are also still high (of course, we have the distances are higher than the average distances).

In order to reach more acceptable results for biologists, we have broadened model with another step of deep learning: **transfer learning**. That is a method enables to re-uses the model developed for a specific task/dataset to lead another task (called *target task*) with another dataset. This process allows rapid process and improves the performance of the model on the target task []. The most popular example has been given with the project ImageNet of Google [] which has labeled several millions of images. The obtained parameter values which can be used in another context to classify another dataset, eventually very different dataset []. The name of this procedure to re-use parameters to pretrain a model is currently called **fine-tuning**.

Fine-tuning does not only replace and retrain the model on the new dataset but also fine-tunes the weights of a trained model by continuing the backpropagation. Unfortunately, *some rapid tests have shown that re-using ImageNet features has not been relevant for our application.* We have designed a way to reproduce the method with our own data. It is worth noting that of course the size of data to pre-train has drastically decreased. For our pre-training step, the network has been trained on the whole dataset including the images of three parts of beetle *i.e thorax, elytra and head*. Then, the trained model has been used to fine-tune and test on each dataset.

12.4.1 Data preparation and training

The images training dataset is combined from the images of three sets: *thorax, elytra, and head* (after augmentation). When applying the training from the scratch, we have used cross-validation to select the data (9 folds). It means that for each dataset, we have some different training data and corresponding testing data. So, the images that use to train the model are just select from one of the folds in each dataset. Specifically, we have taken $1,820$ images of each part. In total, it includes $5,460$ images ($260 \times 7 \times 3$).

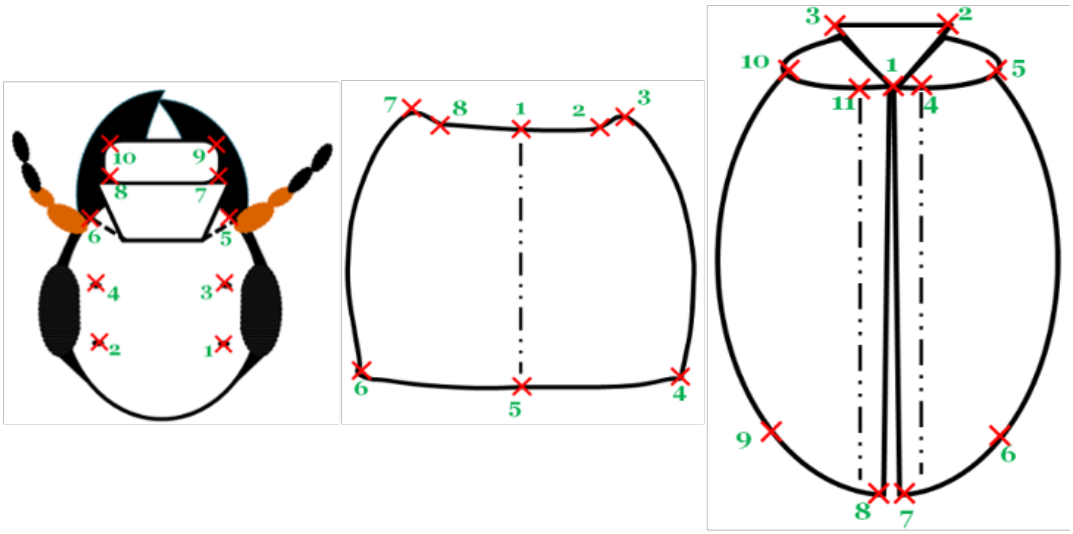


Figure 12.10: A presentation of head, thorax and elytra part with corresponding manual landmarks

However, another problem has been appeared when we combined the images from different dataset. That is the different number of landmarks on each part: 8 *landmarks on thorax part,* 10 *landmarks on head part, and* 11 *landmarks on elytra part*. Fig. 12.10 shows the possition of the landmarks on each part. Because of the meaning of landmarks on each anatomical part for biologists, we cannot insert the landmarks arbitrary. So, we have decided to keep the landmarks on thorax as reference and to remove the landmarks on elytra and head parts instead of adding. We kept 8 (landmarks) as a reference number, then we have removed the supernumerary when it is unnecessary. Specifically, we have removed three landmarks on the elytra part ($1^{st}, 6^{th}, 9^{th}$), and two landmarks on the head part ($5^{th}, 6^{th}$).

During training the proposed architecture on the combined dataset, the parameters of the network (learning rate, momentum, ...) are kept the same as training from scratch but the number of epochs are increased to $10,000$ instead of $5,000$ to achieve better learning on the parameters (weights). Additional, we have shuffled the training set. Because the neural network learns the faster from the most unexpected sample. It is advisable to choose a sample at each iteration that is the most unfamiliar to the system. Shuffling the examples will be helped the model works with different anatomical parts rather than the same anatomical samples in each training time.

12.4.2 Fine-tuning on each dataset

The combined dataset then used to train the third architecture (with 8 outputs). Then, the trained model is used to fine-tuning on each dataset. To compare the result with the previous one, we have also fine-tuned the trained model with different dataset (applying cross-validation). Firstly, we consider on the losses during fine-tuning. *For example*, Table. 12.6, 12.8, 12.10 show the losses during fine-tuning on thorax, elytra, and head dataset, respectively. Comparing with the losses when we trained the model from scratch, *i.e.* on thorax, the validation losses of this scenario have been significantly improved (around 40%).

On each part, the landmarks are predicted on the test images. Then, the average error based on the distances between predicted and corresponding manual landmarks have been also computed. Table. 12.7, 12.9, and 12.11 show the average distances per landmark on thorax, elytra, and head dataset, respectively. **From scratch** columns remind the previously average distances. **Fine-tune** columns present the new average distances after applying fine-tuning on each part. It is clearly shown that the result of predicted landmarks with the help of fine-tuning is more precise than training from scratch. For example, when we compare the average distances between two processes, the worse case of fine-tuning process is still better than the best case of training from scratch.

Round	Training loss	Validation loss
1	0.00019	0.00009
2	0.00018	0.00010
3	0.00018	0.00010
4	0.00019	0.00008
5	0.00019	0.00009
6	0.00018	0.00008
7	0.00019	0.00008
8	0.00018	0.00006
9	0.00018	0.00009

Table 12.6: The losses during fine-tuning model on thorax dataset

#Landmark	From scratch	Fine-tune
1	**4.00**	**2.49**
2	4.48	2.72
3	4.30	2.65
4	4.39	2.77
5	4.29	2.49
6	**5.36**	**3.05**
7	4.64	2.68
8	4.94	2.87

Table 12.7: The average error distance per landmark of two processes on thorax images

Round	Training loss	Validation loss
1	0.00020	0.00006
2	0.00020	0.00006
3	0.00021	0.00006
4	0.00021	0.00006
5	0.00019	0.00006
6	0.00019	0.00006
7	0.00018	0.00005
8	0.00020	0.00006
9	0.00019	0.00006

Table 12.8: The losses during fine-tuning model on elytra dataset

#Landmark	From scratch	Fine-tune
1	**3.87**	2.34
2	3.97	2.27
3	3.92	2.27
4	**3.87**	**2.25**
5	4.02	2.27
6	4.84	3.14
7	5.21	3.14
8	**5.47**	3.29
9	5.27	**3.42**
10	4.07	2.49
11	3.99	2.30

Table 12.9: The average error distance per landmark of two processes on elytra images

Round	Training loss	Validation loss
1	0.00022	0.00007
2	0.00022	0.00007
3	0.00023	0.00008
4	0.00023	0.00008
5	0.00022	0.00008
6	0.00023	0.00007
7	0.00022	0.00008
8	0.00023	0.00007
9	0.00024	0.00008

Table 12.10: The losses during fine-tuning model on head dataset

#Landmark	From scratch	Fine-tune
1	5.53	**3.03**
2	5.16	2.94
3	**5.38**	2.96
4	5.03	2.88
5	4.84	2.76
6	**4.45**	2.67
7	4.79	2.29
8	4.53	**2.20**
9	5.14	2.57
10	5.06	2.44

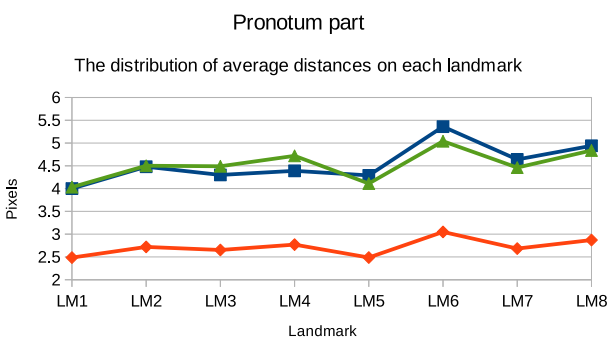
Table 12.11: The average error distance per landmark of two processes on head images

In other view, Fig. 12.12 shows the comparation of the average distance distribution on each dataset in two procedures (from scratch and fine-tuning). In which:

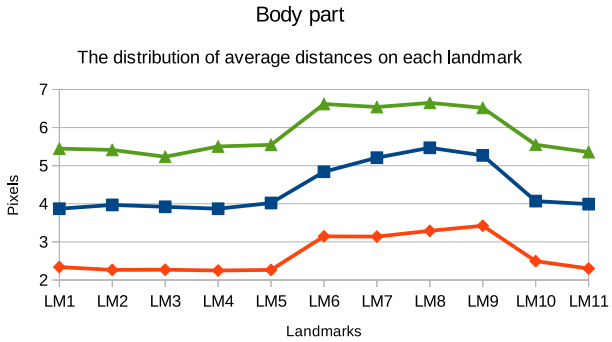
- **Blue** curves: present for the average distances on each landmarks when we train the model from scratch.
- **Orange** curves: describe for the average distance on each landmark when we fine-tune the trained model.
- **Green** curves: illuslate for the average distances **plus its standard deviation** on each landmark in fine-tuning case.

The fine-tuning process has improved the results of the proposed architecture on both 3 datasets: thorax, elytra and head. All the average distances are significantly decreased. Specially, the results have been improved $\approx 40.3\%$ on thorax, $\approx 39.8\%$ on elytra, and $\approx 46.4\%$ on head part based on considering the average distances per landmark.

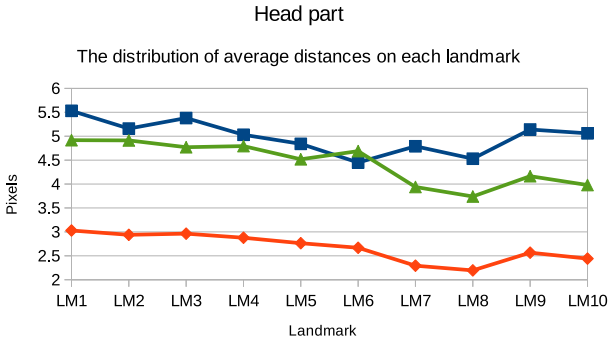
To illustrate the final results, we display the distribution of the distances of both the best and the worst results (resp. landmark 1 and 6) on thorax dataset. The Fig. 12.12 shows in (12.12a) and (12.12b) diagrams how much the average distances (blue lines) and standard errors (red lines) have been improved for the landmark 1, the (12.12c) and (12.12d) diagrams for the landmark 6.



(a) Thorax

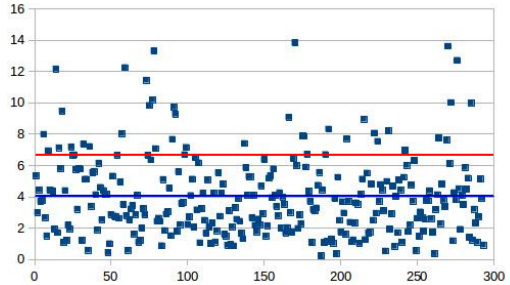


(b) Elytra

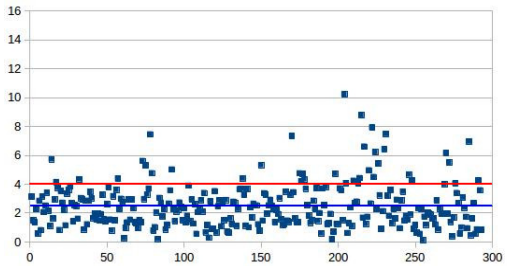


(c) Head

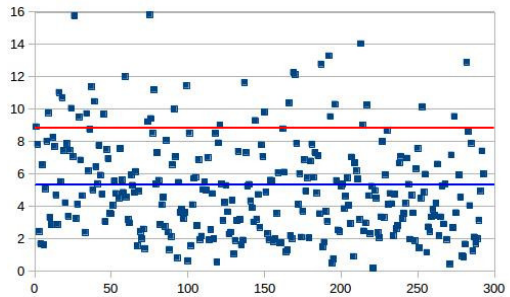
Figure 12.11: The distribution of average distances on each landmark of each part.



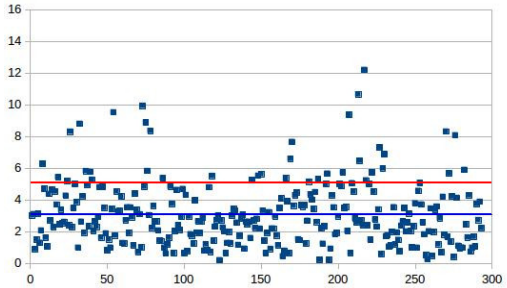
(a) Landmark 1 - CNN



(b) Landmark 1 - fine-tuning



(c) Landmark 6 - CNN



(d) Landmark 6 - fine-tuning

Figure 12.12: The distribution of distances on 1^{st} and 6^{th} landmarks of all images in two testing steps (CNN and fine-tuning) (on thorax dataset). The blue and red lines present the average distances and standard deviation values, respectively.

12.5 Conclusion

In this chapter, we have presented how to apply convolutional neural network to predict the landmark on 2D anatomical images of beetles. After testing several models, we have presented a convolutional neural network for automatic detection landmarks on anatomical images of beetles. The training and testing processes have been finished by using two strategies: *train from scratch* and *fine-tuning*.

In our case, the size of dataset is limited. Therefore, we have applied some techniques to augment dataset. The predicted landmarks have been evaluated by calculating the distance between manual landmarks and corresponding predicted landmarks. Then, the average of distance errors on each landmarks has been considered.

The results have been shown that using the convolutional network to predict the landmarks on biological images leads to satisfying results without need for segmentation step on the object of interest. The best set of estimated landmarks has been obtained after a step of fine-tuning using the whole set of images that we have for the project, i.e. about all beetle parts. The quality of prediction allows using automatic landmarking to replace the manual ones.

The results obtained in this chapter was:

- published and presented in the articles at the conference on Multimedia Analysis and Pattern Recognition 2018 (MAPR 2018) [] and International Conference on Pattern Recognition Systems 2018 (ICPRS-18) [].
- presented at Symposium de Morphomtrie et volution des Formes (SMEF 2018) [].

Part III

Conclusion

Chapter 13

Conclusion

Chapter 14

Discussion

Bibliography

[1] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[2] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[3] Neil A Thacker, PA Riocreux, and RB Yates. Assessing the completeness properties of pairwise geometric histograms. *Image and Vision Computing*, 13(5):423–429, 1995.

[4] David G Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987.

[5] Mostafa S Ibrahim, Amr A Badr, Mostafa R Abdallah, and Ibrahim F Eissa. Bounding box object localization based on image superpixelization. *Procedia Computer Science*, 13:108–119, 2012.

[6] Raman Maini and Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, 3(1):1–11, 2009.

[7] Stephen M Smith and J Michael Brady. Susana new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997.

[8] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[9] Li Wang and Dong-Chen He. Texture classification using texture spectrum. *Pattern Recognition*, 23(8):905–910, 1990.

[10] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

[11] Gustav Theodor Fechner. Elements of psychophysics, 1860. 1948.

[12] Albert Abraham Michelson. *Studies in optics*. Courier Corporation, 1995.

[13] Eli Peli. Contrast in complex images. *JOSA A*, 7(10):2032–2040, 1990.

[14] Timo Ojala and Matti Pietikäinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477–486, 1999.

[15] Priyanka Mukhopadhyay and Bidyut B Chaudhuri. A survey of hough transform. *Pattern Recognition*, 48(3):993–1010, 2015.

[16] Hough Paul VC. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[17] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[18] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[19] Robert F Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(1):579–589, 1991.

[20] Sourabh Niyogi and William T Freeman. Example-based head tracking. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 374–378. IEEE, 1996.

[21] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *European conference on computer vision*, pages 364–378. Springer, 2008.

[22] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In *European Conference on Computer Vision*, pages 29–43. Springer, 2010.

[23] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[24] Chunxia Xiao, Meng Liu, Nie Yongwei, and Zhao Dong. Fast exact nearest patch matching for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1122–1134, 2011.

[25] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[26] Connelly Barnes and Fang-Lue Zhang. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, pages 1–18.

[27] Christine Guillemot and Olivier Le Meur. Image inpainting: Overview and recent advances. *IEEE signal processing magazine*, 31(1):127–144, 2014.

[28] Taeg Sang Cho, Moshe Butman, Shai Avidan, and William T Freeman. The patch transform and its applications to image editing. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[36] Peter M. Roth Martin Koestinger, Paul Wohlhart and Horst Bischof. Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization. In *Proc. First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

[37] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3476–3483, 2013.

[38] Erroll Wood, Tadas Baltrusaitis, Xucong Zhang, Yusuke Sugano, Peter Robinson, and Andreas Bulling. Rendering of eyes for eye-shape registration and gaze estimation. In *Proc. of the IEEE International Conference on Computer Vision (ICCV 2015)*, 2015.

[39] Anne Sonnenschein, David VanderZee, William R Pitchers, Sudarshan Chari, and Ian Dworkin. An image database of drosophila melanogaster wings for phenomic and biometric analysis. *GigaScience*, 4(1):25, 2015.

[40] Celia Cintas, Mirsha Quinto-Sánchez, Victor Acuña, Carolina Paschetta, Soledad de Azevedo, Caio Cesar Silva de Cerqueira, Virginia Ramallo, Carla Gallo, Giovanni Poletti, Maria Catira Bortolini, et al. Automatic ear detection and feature extraction using geometric morphometrics and convolutional neural networks. *IET Biometrics*, 6(3):211–223, 2016.

[41] Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[42] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, et al. Lasagne: First release., August 2015.

[43] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[44] Julie Pallant. *SPSS survival manual*. McGraw-Hill Education (UK), 2013.

[45] Jerome L Myers, Arnold Well, and Robert Frederick Lorch. *Research design and statistical analysis*. Routledge, 2010.

[46] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.