Weekly report

Maire BEURTON-AIMAR and LE Van Linh

November 2016

Abstract

This document contains the summary about the morphometry and deep learning studied. Besides, it also contains the algorithms to apply for image processing such as segmentation or detection the dominant points.

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 method to analysis the morphometry based on the landmarks. 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 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

1.1.1 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:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}); 1 \leq i, j \leq (2k+1) \tag{1.1}$$

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:

$$G = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.2}$$

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.

1.1.2 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:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.3}$$

$$\phi = atan2(G_y, G_x) \tag{1.4}$$

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:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.5}$$

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.

1.1.3 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.

1.1.4 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.

1.1.5 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.

1.2 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.

1.3 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.

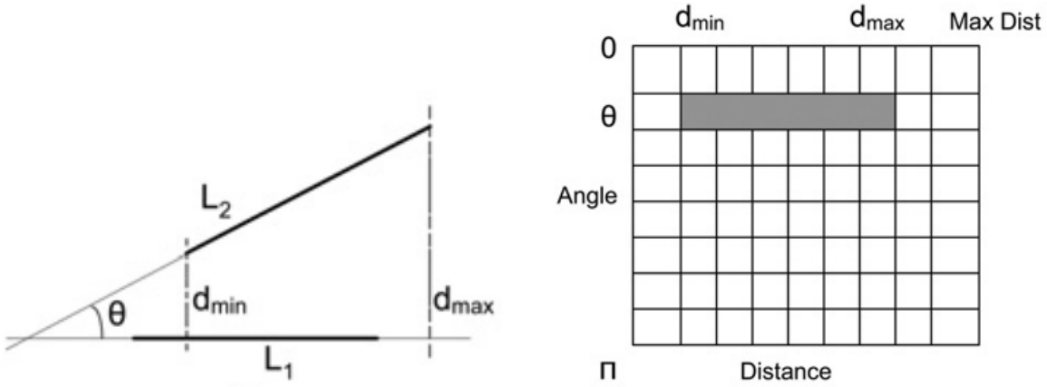
3. If the perpendicular distance from maximum point p_m is greater than the stop condition($d(p_m, l) > \lambda$), then the edge is split at this point and both parts are reprocessed. Otherwise, if we do not have any p_i that the perpendicular distance from they to l greater than λ then the edge can be represented by l.

1.4 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 1.1: 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:

$$d_{Bhattacharyya}(H_i H_j) = \sum_{\theta}^{\pi} \sum_{d}^{d_{max}} \sqrt{H_i(\theta, d) H_j(\theta, d)} \tag{1.6}$$

The significance of parameters in the formula 1.6, 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

Chapter 2

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.

2.1 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.

2.2 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:

$$R_{ccorr}(x,y) = \sum_{x',y'}[T(x'.y').I(x+x',y+y')] \tag{2.1}$$

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:

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x'.y')^2 . \sum_{x',y'} I(x+x',y+y')^2} \tag{2.2}$$

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

$$R_{ccorr\_norm}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)} = \frac{\sum\limits_{x',y'}[T(x'.y').I(x+x',y+y')]}{\sqrt{\sum\limits_{x',y'} T(x'.y')^2 . \sum\limits_{x',y'} I(x+x',y+y')^2}} \tag{2.3}$$

Chapter 3

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.

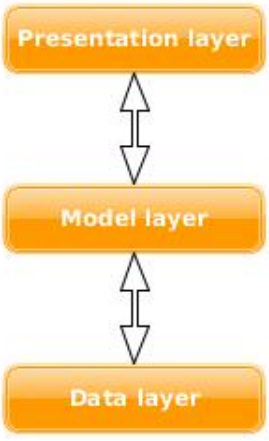


Figure 3.1: Three-tiers model

3.1 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 3.2. The functions of each modules is describing as followed:

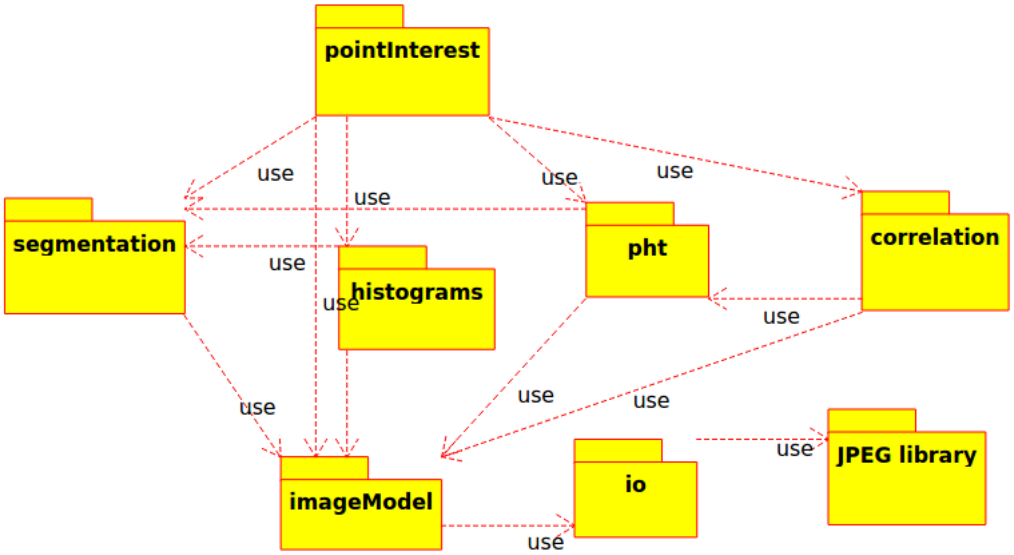


Figure 3.2: Three-tiers model

- **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.

3.2 The classes architecture

Figure 3.3 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.

3.3 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 3.1: 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 3.1, 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 3.1).

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 3.4 show the model image and its manual, and figure 3.5 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 3.6 and 3.7 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.

3.4 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.

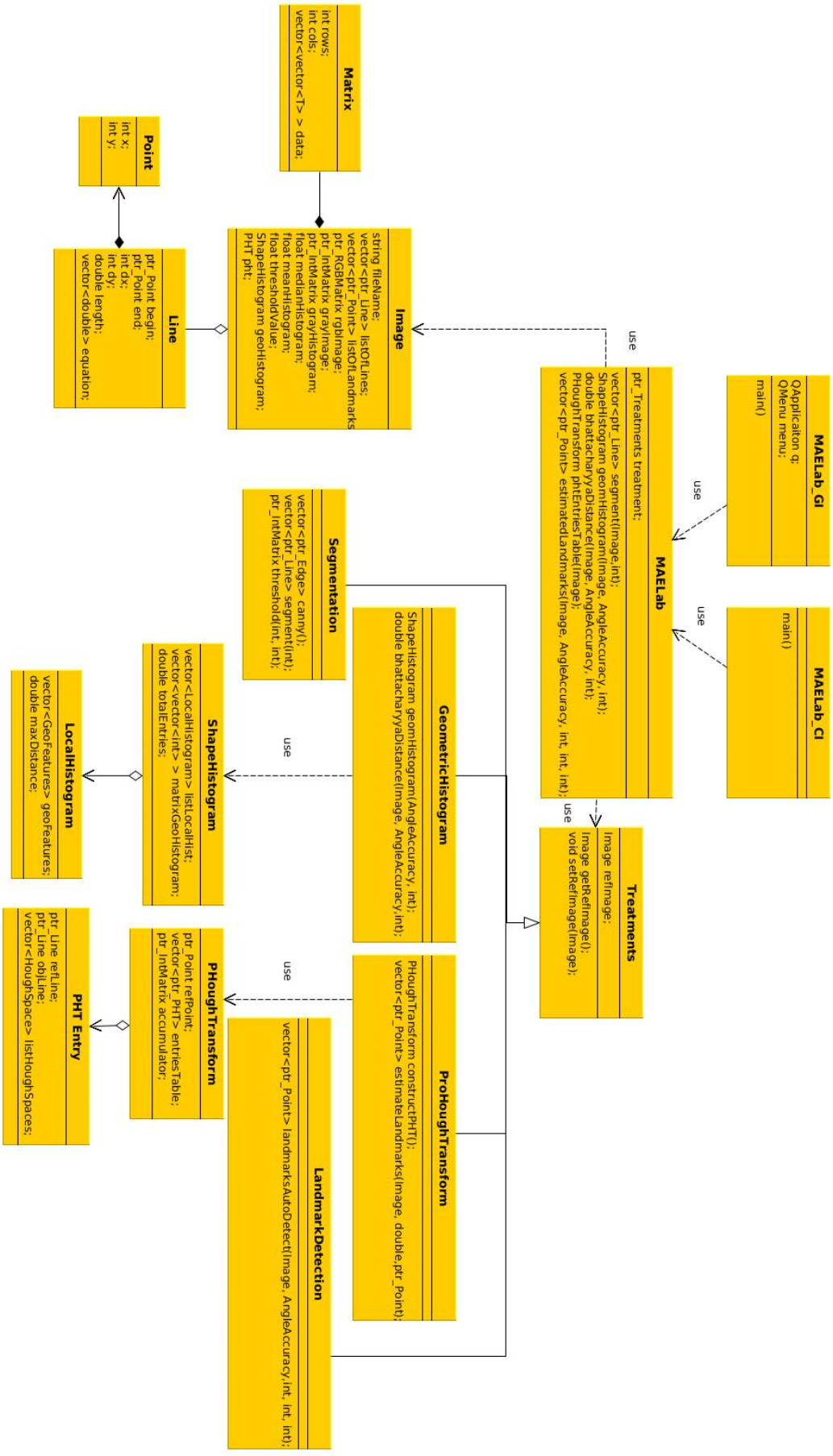


Figure 3.3: Classes diagram

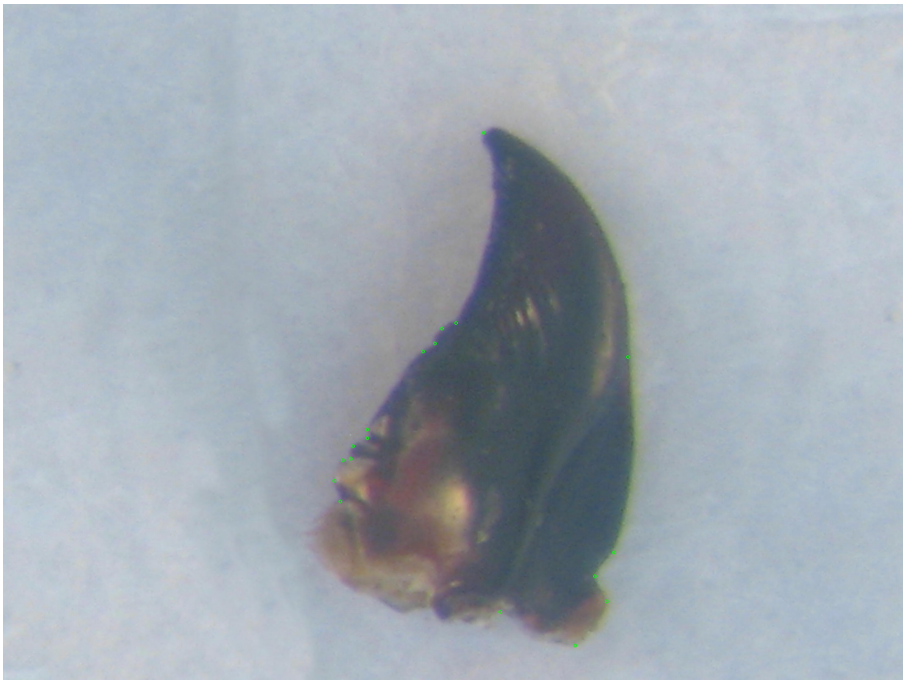


Figure 3.4: Model image with manual landmarks

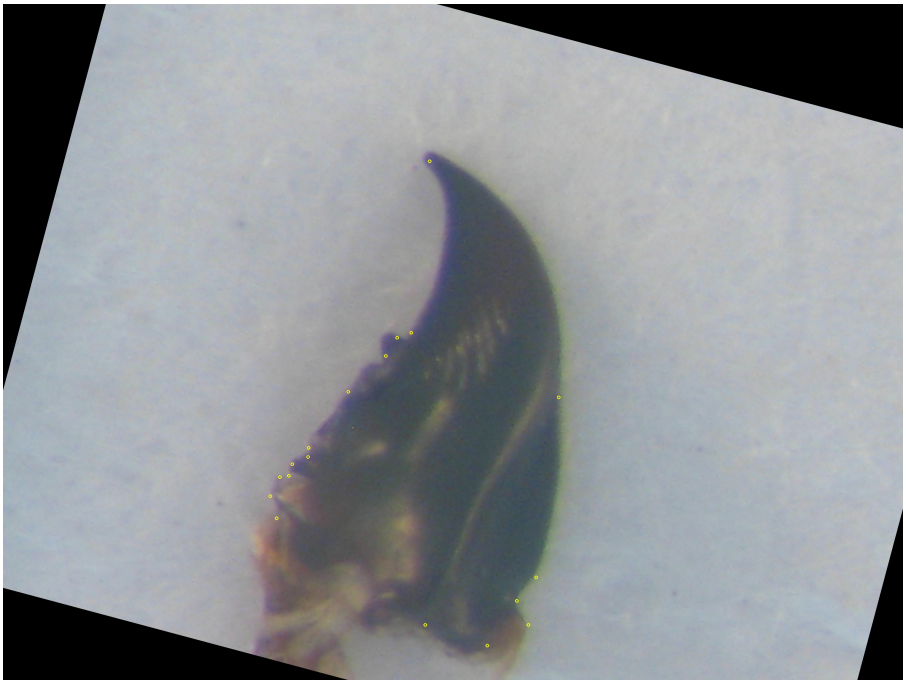


Figure 3.5: Automated landmarks indicated by our method

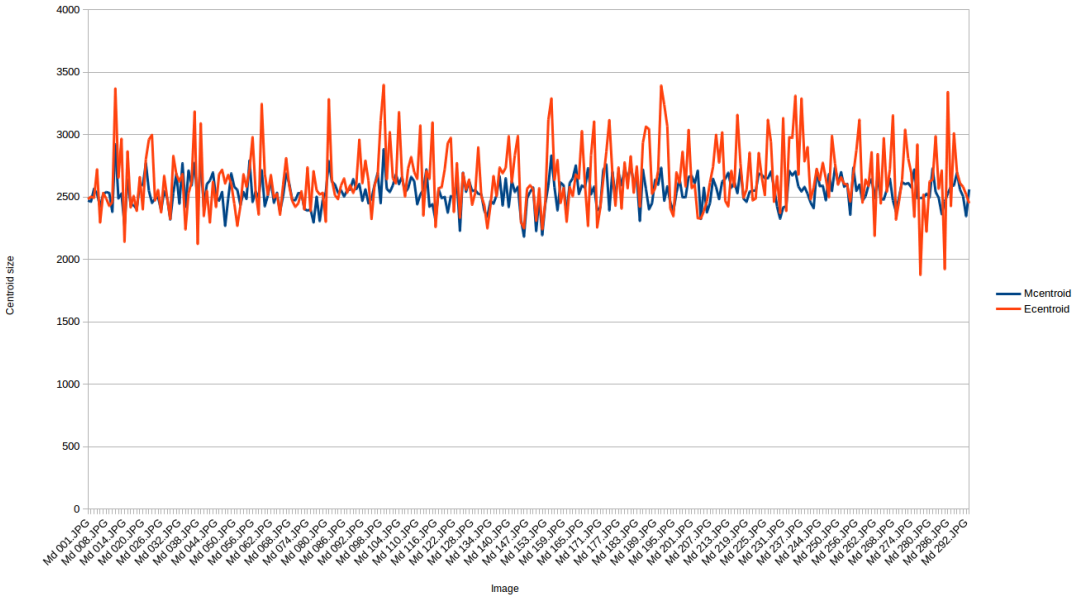


Figure 3.6: The chart presents the accuracy between manual and automated landmarks on right mandible

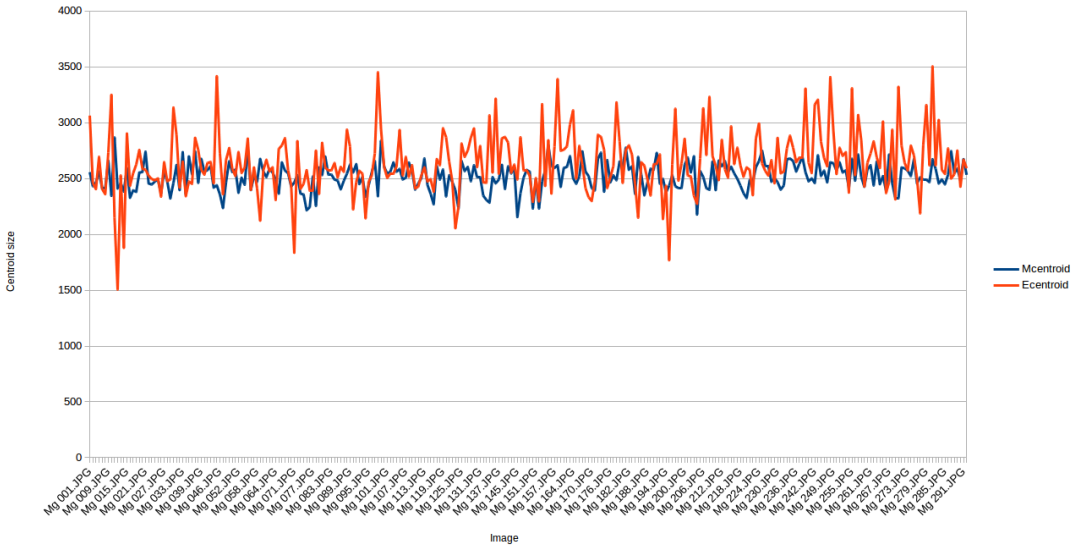


Figure 3.7: The chart presents the accuracy between manual and automated landmarks on left mandible

Part II

Deep learning

Chapter 4

Machine Learning

Machine learning is a norm refer to teach the computer the abilities which are only done by the humans. A machine learning algorithm is an algorithm that is able to learn from data. Most of machine learning algorithms can be divided into two categories: supervised learning and unsupervised learning algorithms.
A machine learning algorithm is built based on the tasked for a machine learning system. 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*

4.1 Supervised learning algorithms

4.2 Unsupervised learning algorithms

4.3 Stochastic Gradient Descent

Chapter 5

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.

5.1 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**):

$$d_1(I_1, I, 2) = \sum_p |{I_1}^p - {I_2}^p| \tag{5.1}$$

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:

$$d_2(I_1, I, 2) = \sqrt{\sum_p ({I_1}^p - {I_2}^p)^2} \tag{5.2}$$

For example, this is a way to compute distace between two images (fig. 5.1):

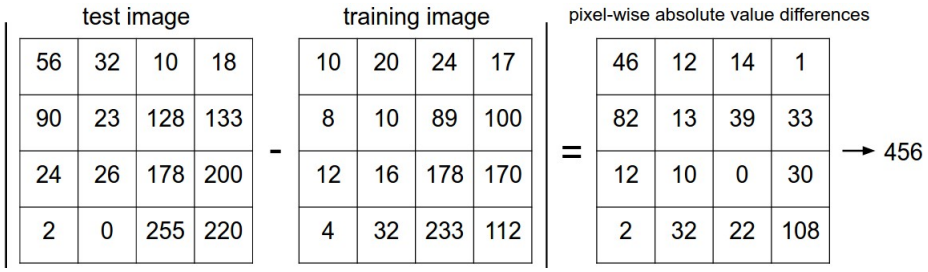


Figure 5.1: An example used **L1 distance** to compare two images

5.2 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.

5.3 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:

$$y = f(x_i, W, b) = Wx_i + b \tag{5.3}$$

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.

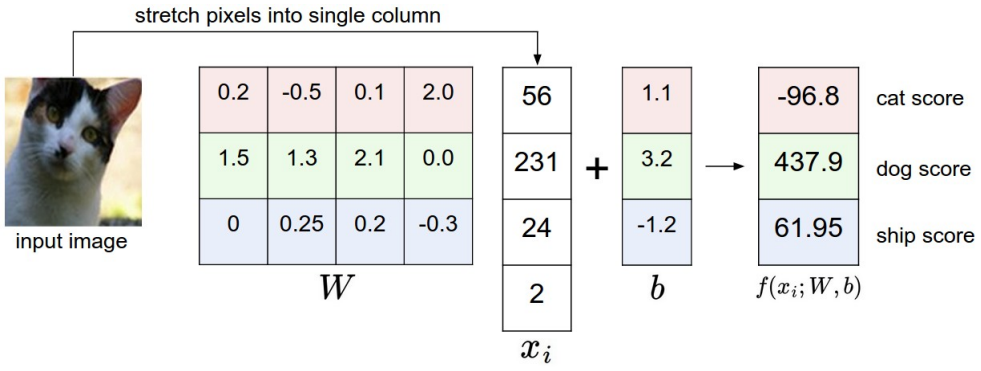


Figure 5.2: An example of mapping an image to a class scores

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:

$$y = f(x_i, W, b) = Wx_i \tag{5.4}$$

Visualation of new score function:

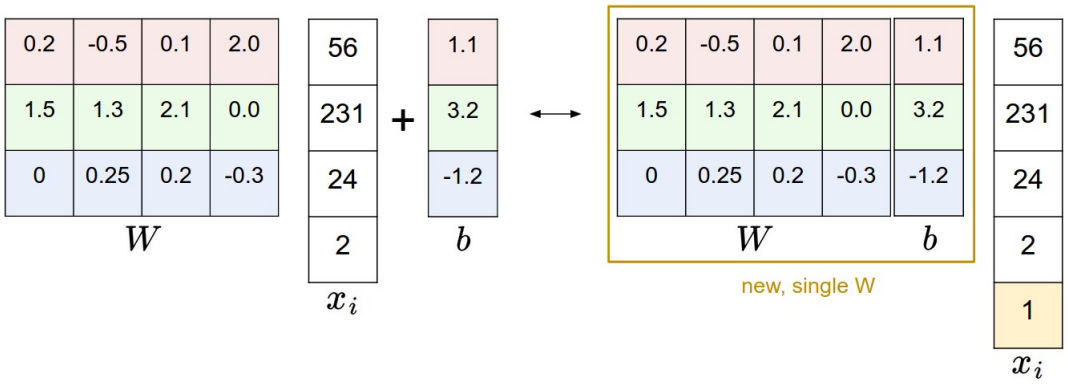


Figure 5.3: An example of bias trick

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.

5.3.1 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:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta) \tag{5.5}$$

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:

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

5.3.2 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.

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{5.6}$$

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

In formula 5.6, 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:

$$H(p, q) = -\sum_x p(x) log(q(x)) \tag{5.7}$$

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.

$$L = \frac{1}{N} \sum_i (H(p, q)) \tag{5.8}$$

The image 5.4 describes an example for a comparison between SVM and Softmax:

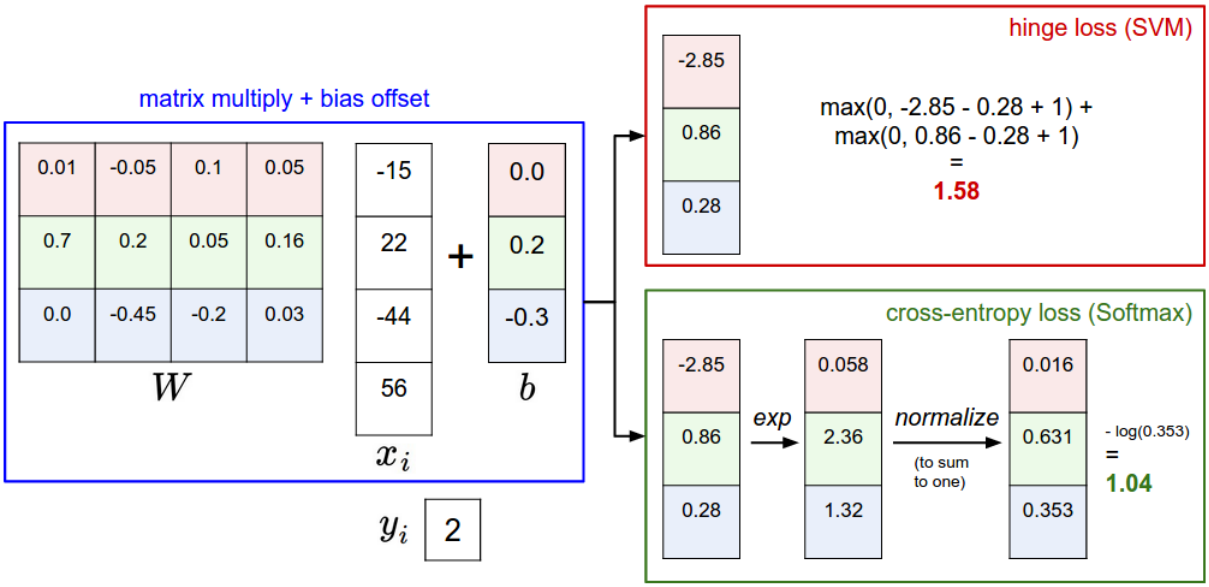


Figure 5.4: An example about SVM and Softmax classifiers

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.

5.4 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.

5.4.1 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).

5.4.2 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)**

5.5 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:

$$f(x, y) = xy \rightarrow \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x \tag{5.9}$$

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:

$$f(x, y) = x + y \rightarrow \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1 \tag{5.10}$$

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

$$f(x, y) = max(x, y) \rightarrow \frac{\partial f}{\partial x} = 1(x \geq y), \frac{\partial f}{\partial y} = 1(y \geq x) \tag{5.11}$$

The image 5.5 show an example about applying derivative to calculate the backward pass of the fucntion $f(x, y, x) = (x + y)z$:

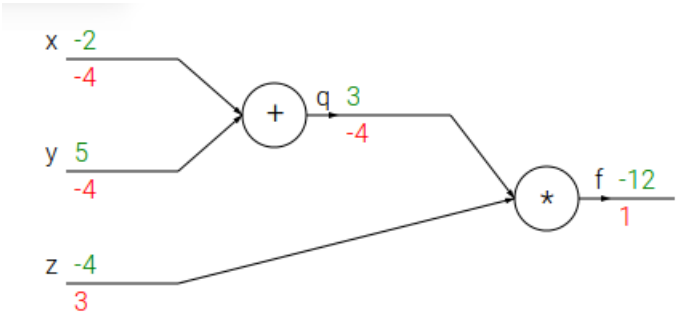


Figure 5.5: A backpropagation example

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:

$$\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow \frac{d\sigma(x)}{dx} = (1-\sigma(x))\sigma(x) \tag{5.12}$$

Chapter 6

Deep Network

6.1 Neural network

6.1.1 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.

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. If the sum is greater than threshold value, the neuron can fire, sending a spike along its axon. Actually, we have many firing rate (called activation function) at a neuron, and the common choice of activation function is the **sigmoid funciont** σ, because it take a real-valued input and squashes it to range between 0 and 1. The image () show the model of a neuron: Some activation functions which we can use:

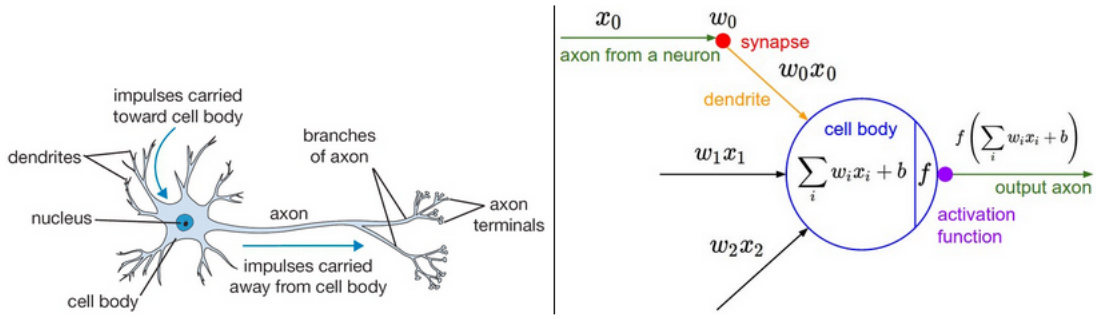


Figure 6.1: A drawing of a biological neuron and its mathematical model

- **Sigmoid function**:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6.1}$$

- **Tanh**

$$tanh(x) = 2\sigma(2x) - 1 \tag{6.2}$$

- **ReLU**

$$f(x) = max(0, x) \tag{6.3}$$

- **Maxout**:

$$f(w^T x + b) = max({w_1}^T x + b_1, {w_2}^T x + b_2) \tag{6.4}$$

6.2 The architecture of neural networks

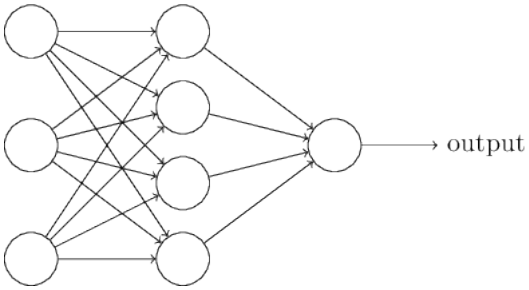


Figure 6.2: A model of neural networks

The image 6.2 show a simple model of neural networks. 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. The middle layer is called a hidden layer. The network in example 6.2 has just a single hidden layer, but many networks have multiple hidden layers. 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 layer. At each layer excepts the output layer, the output is indicated by a activation function (i.e loss, tanh,...). The size of a neural network can be to compute as the number of neurons, or the number of parameters.

6.3 Deep network

Chapter 7

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.

7.1 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.

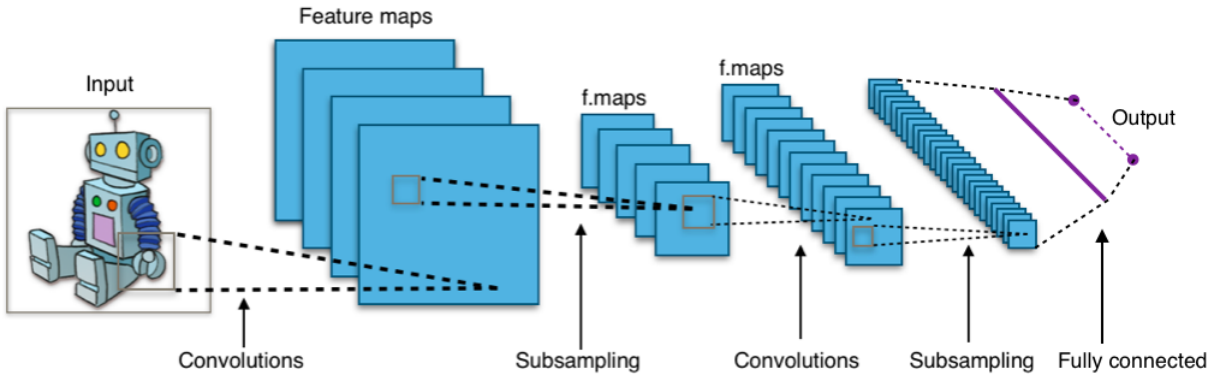


Figure 7.1: An architecture of convolutional neural network

7.1.1 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:

$$N = \frac{(W - F + 2P)}{S} + 1 \tag{7.1}$$

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 7.1 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 7.2, 7.4), 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 7.2 and 7.4 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.

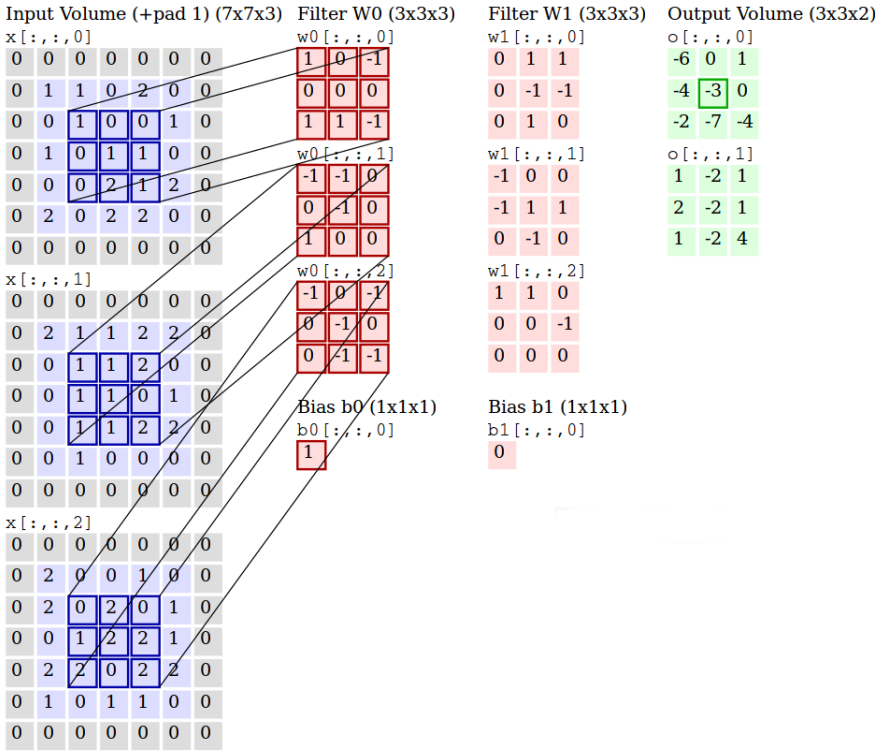


Figure 7.2: Convolutional input with W0 filter

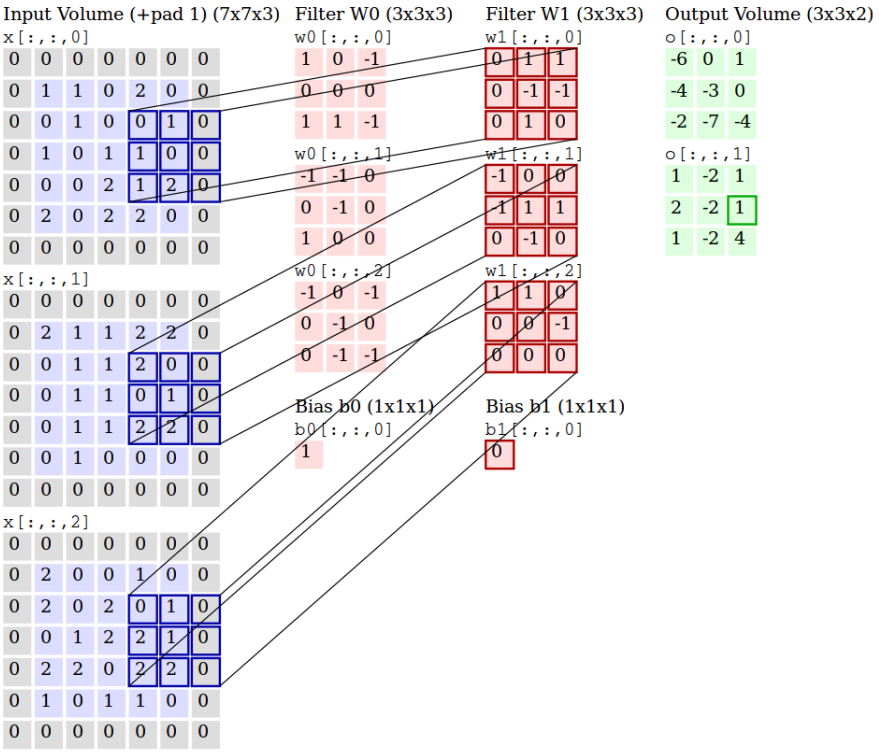


Figure 7.3: Convolutional input with W1 filter

7.1.2 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.

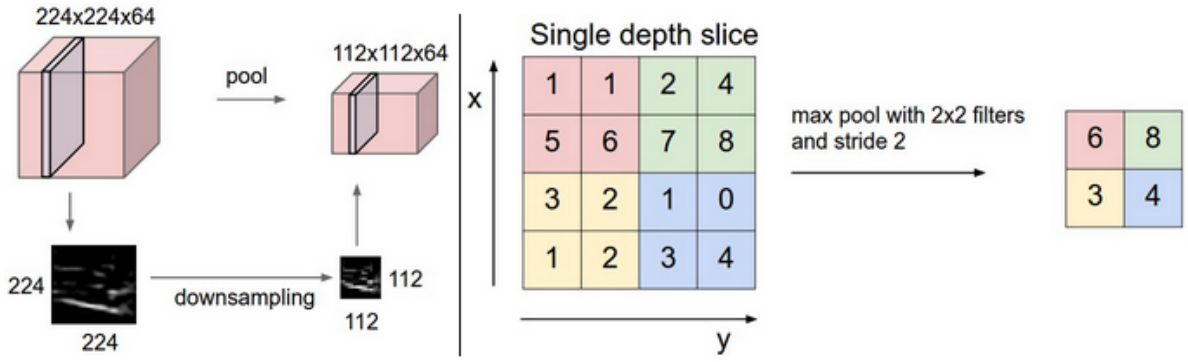


Figure 7.4: A POOL layer with MAX function

7.1.3 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.

7.2 Case studies

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

- **LeNet**: is developed by Yann Le Cun [5] in 1990's. It was used to read the zip code, digits.
- **AlexNet**: is developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [6]. 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 [7]. 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 [8]. 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[9]. They show that the depth of the network is a critical component for good performance.

- **ResNet**: is developed by Kaiming He and al[10]. 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[11].

7.3 Caffe framework

Caffe is a deep learning framework 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. This section will describe the components and the architecture of Caffe. We also using Caffe to create a small network for classification.

7.3.1 Definitions

Deep network is a model that represented as a collection of inter-connected layers. Keep this definition, Caffe defines a network layer by layer in its model. The data through in the network called `blobs`. Blobs is the standard array and unified memory interface for the framework. The layer is the foundation of the model and computation. The net is defined as the collection of connection between the layers. The structure of blob will describe the way that information is stored and communicated in the layers and networks.

Blobs

Caffe stores and communicates data using blobs. Blobs provide a uniform memory to hold data. The blob dimensions for batches of an image are number **N** x channel **K** x height **H** x width **W**. Where:

- Number N: is the batch size of the data (the number of data through the network in the same time).
- Channel K: is the feature dimension of image (i.e for RGB image K = 3)
- Width W and height H: are width and height of the image data.

In Caffe, the dimension of blob is dependent on the type and configuration of the layer.

Layers

The layer is the principle of the model and the fundamental unit of computation. Caffe has provided a lot of layer's type as convolution, pool, inner products,etc. But most of types have the same model as figure 7.5. A layer take input from the bottom connections and through the output to top connections. Each layer type defines three critical computations:

- **Setup**: initialize the layer and its connections.
- **Forward**: given the input from bottom connections, compute and send the output to the top connections.
- **Backward**: given the gradient from the top connection, compute the gradient to the input and send to the bottom connections.

An example about declaring a layer in Caffe and its model:

```
layer {
    name : "conv1"
    type : "Convolution"
    bottom : "data"
    top : "conv1"
    param{
        lr_mult : 1
    }
    param{
        lr_mult : 2
    }
    convolution_param{
        num_output : 20
        kernel_size : 5
        stride : 1
        weight_filler{
            type : "xavier"
        }
        bias_filler{
            type : "constant"
        }
    }
}
```

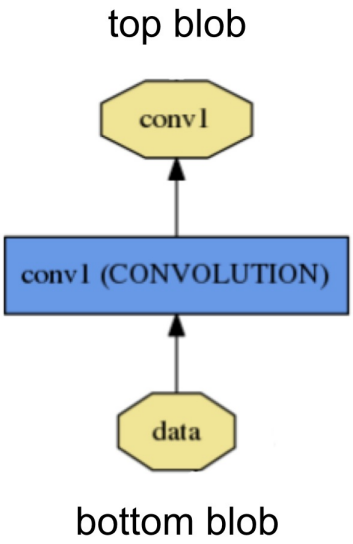


Figure 7.5: Model of a layer in Caffe framework

Nets

The net is a set of layers connected in a computation directed acyclic graph. A net begins with a data layer that loads the data from the disk and ends with a loss layer that computes the objective of the network (such as classification, recognition,...). As simple, a net is defined as figure 7.6.

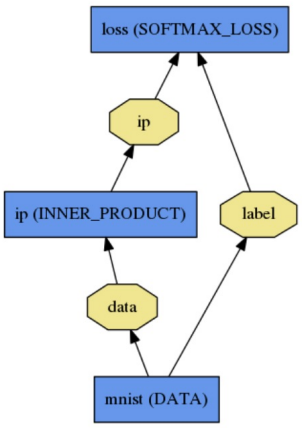


Figure 7.6: Model of a net in Caffe framework

Forward pass

Forward pass computes the output given the input by the functions. Each function stays at each layer of the model. The forward pass through from bottom to the top via each layer of model. The image 7.7 shows a forward example. The input data x through the **Inner Product** layer and **Softmax** layer to give the last result. Normally, the output of forward pass includes the result of score function and the error (loss) of the model.

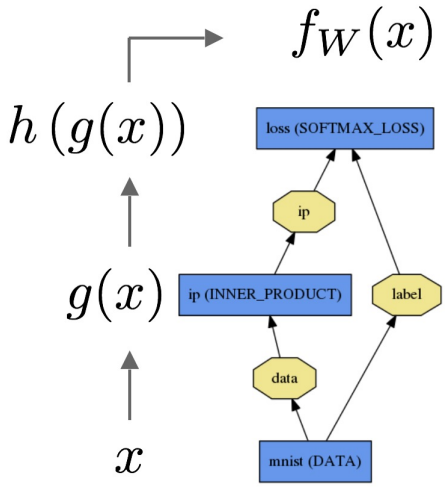


Figure 7.7: Forward pass in a Caffe model

Backward pass

Backward pass receives the loss as the input, it computes the gradient for learning of the model. The gradient of the model is computed by inverse-compose gradient of each layer in the model. Backward pass through from top to bottom. The gradient respect of the model is computed layer by layer via the chain rule through the parameters of each layer (figure 7.8).

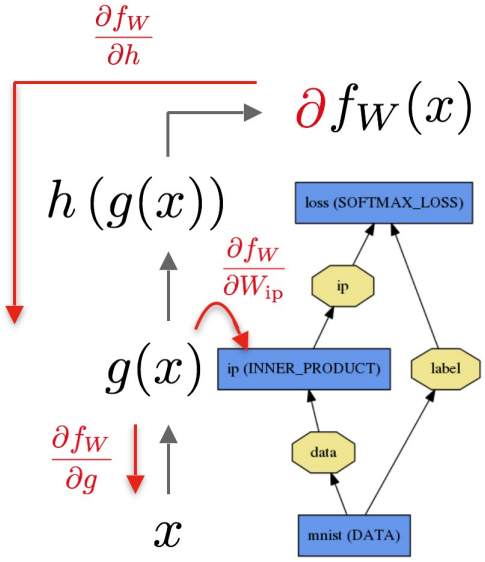


Figure 7.8: Backward pass in a Caffe model

Solver

In Caffe, Solver is designed for optimizing the model by coordinating the network's forward inference and backward gradients to form parameter update that attempt to improve the loss. The steps in optimizing process are followed:

- Firstly, calling the forward pass to obtain the output and loss,
- Secondly, calling the backward pass to generate the gradient of model,
- Finally, Incorporating the gradient into the weight update that attempts to minimize the loss.

The Caffe solvers include:

- Stochastic Gradient Descent
- AdaDelta
- Adaptive Gradient
- Adam
- Nesterov's Accelerated Gradient
- RMSprop

7.4 Layers Catalogue

The layers in Caffe framework are organized similarly to CSS language. They are defined following the structure with the specific attributes. The model architecture of a network using Caffe is defined in a protocol buffer definition file (prototxt). In general, the structure of a layer in Caffe looks like:

```
layer {     # layer is keyword
          name : "name"     # Name of layer
          type : "layer_type"     # Type of layer
          bottom : "bottom_blob"     # Input (bottom) of layer
          top : "top_blob"     # Output (top) of layer
          # And the parameters of layer(required, recommended or optional parameters)
     }
```

Depending to specific layer, Caffe has provided difference parameters. This section will discuss a few of the important layers.

Convolution layer

The convolution layer means applying a mathematical function to a range of image. The aim of this function is convolutional the image with a new depth. The parameters of this layer are as followed:

- Layer type: `Convolution`
- Required parameters:
  - `num_output(n)`: the number of filters

  - `kernel_size`: specifies width and height of each filter

- Recommended parameters: `weight_filler` has two sub-parameters: `type` and `value`.This is an initialization for filter)

- Optional parameters:

  - `stride`: specifies the moving-step when applying the filters to the input.
  - `bias_term`: specifies the addition biases to the outputs.
  - `pad`: specifies the addition pixels to each side of the input.
  - `group(g)`: Indicating the connectivity of each filter to a subset of the input.

An example of convolution layer (fig.7.9):

```
layer{
        name: "conv1"
        type: "Convolution"
        bottom: "data"
        top: "conv1"
        convolution_param{
                num_output: 100
                kernel_size: 5
                stride: 1
                weight_filler{
                        type: "gaussian"
                        std: 0.01
                }
                bias_filler{
                        type: "constant"
                        value: 0
                }
        }
}
```

Figure 7.9: A convolution layer in Caffe

Pooling layer

The pooling layer works similar to a convolution layer but the functions at pooling layer are different (maximum function (MAX) or average function (AVE)). The parameters of pooling layer are followed:

- Layer type: `Pooling`

- Required parameter: `kernel_size` specifies width and height of each filter

- Optional parameters:

  - `pool`: the pooling method, such as: **MAX, AVE, STOCHASTIC**.
  - `pad`: specifies the pixels to add to each side of the input.
  - `stride`: moving-step when applying the filters to the input

An example of pooling layer (fig. 7.10):

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
```

Figure 7.10: A pooling layer in Caffe

Loss layers

Loss of learning is indicated by comparing an output to a target and assigning cost to minimize. The loss computed includes the loss in forward pass and the gradient in backward pass. The loss functions in Caffe are described as followed:

- **Softmax** (layer type: `SoftmaxWithLoss`)
- **Euclidean** (layer type: `EuclideanLoss`)
- **Hinge/Margin** (layer type: `HingeLoss`)
- **Sigmoid Cross-Entropy** (layer type: `SigmoidCrossEntropyLoss`)
- **Infogain** (layer type: `Infogain`)

Example about a loss layer using **Softmax** (fig. 7.11):

```
layer{
        name: "layer4"
        type: "SoftmaxWithLoss"
        bottom: "fc"
        bottom: "label"
        top: "loss"
}
```

Figure 7.11: A loss layer in Caffe

Activation layers

Activation layers are taking one bottom blob and producing one top blob of the same size with an activation function. The activation function in Caffe includes:

- **ReLU** (layer type:`ReLU`)
- **Sigmoid** (layer type:`Sigmoid`)
- **TanH** (layer type:`TanH`)
- **Absolute** (layer type:`AbsVal`)
- **Power** (layer type:`Power`)
- **BNLL** (layer type:`BNLL`)

An activation layer with ReLU (fig. 7.12):

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
```

Figure 7.12: A ReLU layer in Caffe

Inner Product layer

This is a fully connected layer of Caffe. The parameters of this layer are followed:

- Layer type: `InnerProduct`
- Required parameter: `num_output(c_o)` the number of filters
- Recommended parameter: `weight_filler` initialize of filters.
- Optional parameters:
  - `bias_filler`: initialize of bias filters
  - `bias_term`: specifies whether to learn and apply a set of additive biases to the filter outputs.

An example of inner product layer (fig. 7.13):

```
layer{
        name: "layer3"
        type: "InnerProduct"
        bottom: "pool2"
        top: "fc"
        inner_product_param{
                num_output: 2
                weight_filler{
                        type: "gaussian"
                        std: 0.01
                }
                bias_filler{
                        type: "constant"
                        value: 0
                }
        }
}
```

Figure 7.13: An Inner Product layer in Caffe

Data layer

Data enters Caffe through data layers. They lie at the bottom of nets. Caffe strongly supports many kinds of data, directly from memory or from files on disk in HDF5 or common image formats. The types of data are include:

- **Database**: layer type `Data`
  - Required parameters:

    - * `source`: the name of the directory containing the database
    - * `batch_size`: the number of inputs to process at one time
  - Optional parameters:
    - * `rand_skip`: skip up to this number of inputs at the beginning
    - * `backend`: choose whether to use a **LEVELDB** or **LMDB**

- **In-Memory**: layer type `MemoryData`. Required parameters: `batch_size, channels, height, width` - specify the size of input chunks to read from memory
- **HDF5 Input**: layer type `HDF5Data`. Required parameters are followed:
  - `source`: the name of the file to read from
  - `batch_size`: number of images to batch together
- **HDF5 Output**: layer type `HDF5Output`. Required parameter are **file_name** which name of file to write the output.
- **Images**: layer type `ImageData`.
  - Required parameters:
    - * `source`: name of a text file. Each line gives an image filename and label
    - * `batch_size`: number of images to batch together
  - Optional parameters:
    - * **rand_skip**: skip up to this number of inputs at the beginning
    - * `shuffle`
    - * `new_height, new_width`: resize all images to this size

An example of data layer:

```
layer{
        name: "layer1"
        type: "ImageData"
        top: "data"
        top: "label"
        transform_param {
            mean_file: "./data/img_rating/mean.binaryproto"
          }
        image_data_param{
                source: "./data/img_rating/file_list_1.txt"
                batch_size: 10
                new_height: 64
                new_width: 64
        }
}
```

Figure 7.14: A data layer in Caffe

7.5 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.

Part III

Conclusion

Chapter 8

Discussion

Chapter 9

Conclusion

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] 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.

[6] 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.

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

[8] 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.

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

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

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