Chapter 3

Gabor Wavelet and AdaBoost

General comment. Hyphenation settings should be changed to stop this sort of hyphenation

This chapter presents the fundamentals of Gabor-Boosting algorithm: Gabor wavelets and AdaBoost. Section 3.1 discusses the Gabor wavelets and how face images are represented by Gabor wavelet features. In Section 3.2, the AdaBoost algorithm is presented, and the principle of constructing the "strong" AdaBoost classifier~~,~~ is discussed.

3.1 Gabor Wavelet

Spell out here first use of 2-D

vo-dimensional Gabor filter is a linear filter whose kernel is similar to the ~~two-dimensional (~~2-D~~)~~ receptive profiles of the mammalian cortical simple cells [74]. A 2-D Gabor kernel is a 2-D Gaussian function multiplied by a 2-D harmonic function. Generally, a harmonic function is a Fourier basis function. Especially, in a 2-D Gabor kernel it is a sinusoidally modulated function, in a form of complex exponential function. The Gaussian function varies in dilation and the harmonic function varies in rotation and frequency, so that a group of 2-D Gabor filters can be formed into 2-D Gabor wavelets. Gabor wavelet capture~~s~~ local structure corresponding to spatial frequency, *i.e.*, scale, spatial localisation, ~~*i.e.*,~~ coordinate, and orientation selectivity.

Gabor wavelets are used to extract facial information from face images. Section 3.1.1 gives the background study on Gabor wavelets. The definition of Gabor wavelet is given in Section 3.1.2. Section 3.1.3 illustrates the Gabor wavelet transform. Section 3.1.4 presents the representatives of face images - Gabor wavelet features.

3.1.1 Background of Gabor Wavelets

2-D Gabor wavelet ~~is~~ widely adopted as a feature extraction approach in texture segmentation [36, 35, 72, 151], iris recognition [30], face recognition [161, 162], face expression recognition [60, 102, 170], and image retrieval [104].

When Gabor filters are applied on computer vision or image processing task, ~~one~~ biggest problem is how to select the appropriate Gabor filters, *i.e.*, the parameters. The parametric characterisation has been studied extensively and different types of schemes have ~~been~~ emerged. In [36], 2-D Gabor filters transform different texture into detectable filter-output discontinuities at texture boundaries. Gabor filter output is modeled as Rician random variables [36]. A decision algorithm for selecting optimal filter parameters based on the Rician model is developed by Dunn and Higgins [35]. In [72], a bank of Gabor filters is characterised as uniformly covering the spatial-frequency domain, and a filter selection scheme is presented based on minimal "energy" loss in reconstructed images from the filtered images. Teuner *et al.* [151] choose parameters of Gabor filters based on the analysis of spectral feature contrasts obtained from iterations of pyramidal Gabor transforms. The work ~~is benefit from no need for a~~ prior knowledge of the texture image so that the segmentation processing is unsupervised. Since these parametrization solutions are all to choose optimal frequency or orientation of Gabor filter, an alternative wavelet scenario is proposed to bypass the optimisation.

A 2-D Gabor wavelet model with multi-scale and multi-orientation is originally proposed by Daugman [28, 30] into biometric research. In [30], by using a three-lay neural network, a nonorthogonal 2-D Gabor representation is generalised. Daugman also has applied his 2-D Gabor wavelet model on human iris recognition. In [29], visible texture of ~~humans'~~ iris is transformed as a sequence of multi-scale 2-D Gabor wavelet digits. Lades *et al.* [86] has applied Gabor wavelets for face recognition using the Dynamic Link Architecture (DLA) framework. The DLA start~~ed~~ by computing Gabor wavelets, and then it performs a flexible template comparison between resulting image decompositions using graph-matching. Wiskott *et al.* [161, 162] have extended on DLA by developing a Gabor wavelet based Elastic Bunch Graph Matching (EBGM) to label and recognise faces. Faces are represented by labelled graphs using Gabor wavelet transform. The labelled graphs consist of nodes and edges which are positioned by elastic bunch graph matching processing ~~with~~ comparing similarities. The testing is done on the FERET database [120] showing a high recognition rate for frontal face images. Liu and Wechsler [95] applied the Enhanced Fisher linear discriminant Model (EFM) to an augmented Gabor feature vector [97] derived from the Gabor wavelet representation of face images. Liu [94] also presents a Gabor-based kernel Principal Component Analysis (PCA) method by integrating Gabor wavelet representation and the kernel PCA for recognition. Fan *et al.* [38] combined Gabor wavelet and Null space-based Linear Discriminate Analysis (LDA) simultaneously on each orientation~~s~~ for generating feature vectors.

In analysis of facial expression, the recognition is to analyse the relationship between the movements made by facial features, such as eyebrows, eyes and mouth. These facial features can be defined as point-based visual properties of facial expressions. Hong *et al.* [60] use Gabor wavelets of five frequencies and eight orientations to define a "big" General Face Knowledge (GFK) with 50 nodes[1] on a face, and a "small" 16-node GFK with three frequencies and four orientations. The method which fits these nodes with face image is the elastic graph matching proposed by Wiskott *et al.* [161, 162] in face recognition. Zhang *et al.* [170] use 34 facial points for which a set of Gabor wavelet coefficients, the Gabor wavelets with three frequencies and six orientations have been utilised. Lyons *et al.* [102] use a fiducial grid of 34 nodes and apply wavelets of five frequencies and six orientations.

3.1.2 The Definition of Gabor Wavelet

Gabor wavelets were introduced to image analysis due to their biological relevance and computational properties [74]. A Gabor wavelet [30] (~~somewhere~~,

[1]The nodes are landmark points on human faces.

called Gabor Kernel or Gabor Elemental Function [36]) is defined as

$$\psi_{\mu,\nu}(z) = \frac{\|k_{\mu,\nu}\|^2}{\sigma^2} e^{-\frac{\|k_{\mu,\nu}\|^2\|z\|^2}{2\sigma^2}} [e^{ik_{\mu,\nu}z} - e^{-\frac{\sigma^2}{2}}] \tag{3.1}$$

where $z = (x, y)$ indicates a point with x, ~~i.e.,~~ the horizontal coordinate and y, ~~i.e.,~~ the vertical coordinate. the parameters μ and ν define the angular orientation and the spatial frequency of the Gabor kernel. In Equation 3.1, the spatial frequency modulates the size of 2-D discrete Gabor kernel, so that ν also determines the scale of kernel. The operator $\| \cdot \|$ denotes the norm operator. The parameter σ is the standard derivation of Gaussian window in the kernel. The wave vector $k_{\mu,\nu}$ is defined as

$$k_{\mu,\nu} = k_\nu e^{i\phi_\mu} \tag{3.2}$$

where $k_\nu = \frac{k_{max}}{f^\nu}$ and $\phi_\mu = \frac{\pi\mu}{8}$ if eight different orientations have been chosen. k_{max} is the maximum frequency, and f is the spatial factor between kernels in the frequency domain.

Wavelets

Gabor kernels in Equation (3.1) are all self-similar since they are generated from one kernel (a mother wavelet) by dilation and rotation via the wave vector $k_{\mu,\nu}$. Each kernel is a product of a Gaussian envelope formulated in Equation ~~3.3~~

First use of DC - spell it out

$$\frac{\|k_{\mu,\nu}\|^2}{\sigma^2} e^{-\frac{\|k_{\mu,\nu}\|^2\|z\|^2}{2\sigma^2}} \tag{3.3}$$

and a complex plane wave $e^{ik_{\mu,\nu}z}$. The complex wave determines the oscillatory part of the kernel. The term $-e^{-\frac{\sigma^2}{2}}$ compensates for the DC value Equation 3.3, which makes the kernel DC-free. DC-free [84] is a wavelet terminology that ensures wavelets do not lose any generality such that there is no minimal energy loss when images are reconstructed by the wavelets. The effect of the DC term becomes negligible when the parameter σ, which determines the ratio of the Gaussian window width to wavelength, is a sufficiently large value.

Format these properly as fractions

Parametrization

In this thesis, five different scales and eight orientations of Gabor wavelets are used, i.e., $\nu \in \{-1, \ldots, 3\}$, and $\mu \in \{0, \ldots, 7\}$. The images adopted are smaller than the images with 128×128 size used in [161, 162], so that the scale range is from -1 to 3 rather than from 0 to 4. The eight orientations in radian are 0, $\pi/8$, $\pi/4$, $3\pi/8$, $\pi/2$, $5\pi/8$, $3\pi/4$, and $7\pi/8$. Gabor wavelets are modulated by a Gaussian envelope function with relative width $\sigma = 2\pi$. The maximum frequency is $k_{max} = \frac{\pi}{2}$, and the factor $f = \sqrt{2}$. These parameters are chosen according to previous findings [162, 94]. The kernels exhibit desirable characteristics of spatial frequency, spatial locality, and orientation selectivity.

Complex Gabor

Gabor kernel is a product of a Gaussian and a complex plane wave with real ~~part~~ and imaginary part, also called even and odd. The Equation 3.1 is separated into real and imaginary unit, so that the real part is

$$\frac{k_\nu^2}{\sigma^2} e^{-\frac{\|z\|^2 k_\nu^2}{2\sigma^2}} \{\cos(k_\nu \cos(\phi_\mu)x + k_\nu \sin(\phi_\mu)y) - e^{-}\frac{\sigma^2}{2}\} \tag{3.4}$$

and the imaginary part becomes

$$\frac{k_\nu^2}{\sigma^2} e^{-}\frac{\|z\|^2 k_\nu^2}{2\sigma^2} \sin(k_\nu \cos(\phi_\mu)x + k_\nu \sin(\phi_\mu)y) \tag{3.5}$$

The real part and the imaginary part of 40 Gabor wavelets are shown in Figure 3.1 and Figure 3.2 respectively.

These Gabor wavelets share 5 scales and 8 orientations. The orientations from left to right are 0, $\frac{\pi}{8}$, $\frac{\pi}{4}$, $\frac{3\pi}{8}$, $\frac{\pi}{2}$, $\frac{5\pi}{8}$, $\frac{3\pi}{4}$, and $\frac{7\pi}{8}$. The scales from top to bottom are -1, 0, 1, 2, 3.

3.1.3 Gabor Wavelet Transform

In computer vision, a feature stands as a piece of “interesting” information which is relevant for solving a specific vision application. In the appearance

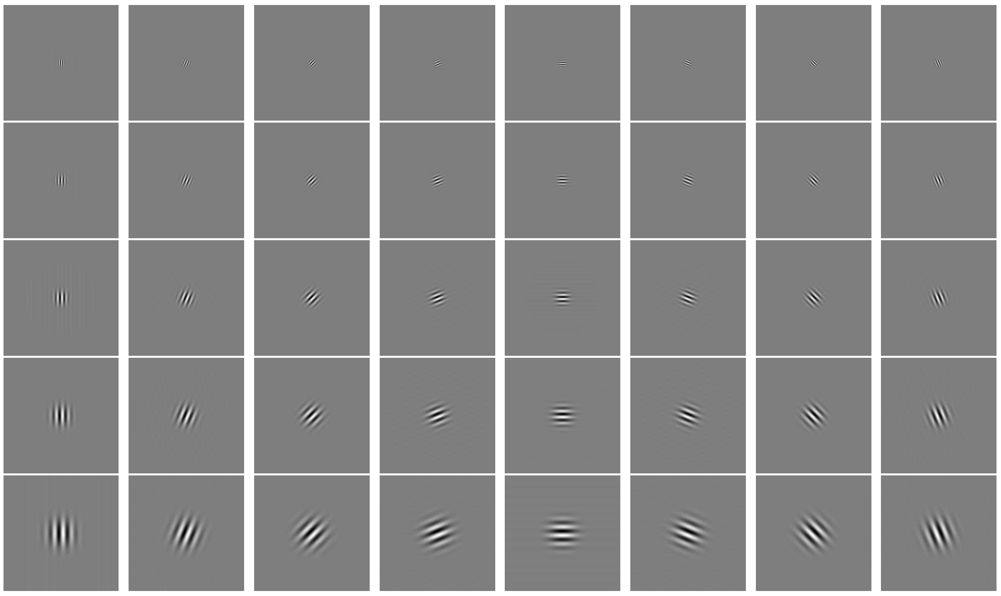

Figure 3.1: The real part of the 5 × 8 Gabor wavelets.

based approaches, features refer to the result after a general neighbourhood operation applied on an image. The result contains local or global information which contribute to resolv~~e~~ a specific vision problem. An image can be represented by a group of features. A computer vision system works on features rather than image pixels directly. Extraction of feature is defined in terms of local neighbourhood operations. In this thesis, Gabor wavelet transform is the process to extract features which are relevant to face recognition. Since the selective schemes are available for frequency range and orientation interval, Gabor wavelets are ideal for face feature representation.

This section illustrates the Gabor wavelet transform by convolution. First of all, the concept of convolution is given. Secondly, the 2D discrete convolution is presented. Thirdly, ~~T~~he size of mask for convolution is determined. Finally, the magnitude response is used as extracted features.

Convolution

The Gabor wavelet transform is the course of two-dimensional convolution of an image with a family of Gabor wavelet kernels defined by Equation 3.1.

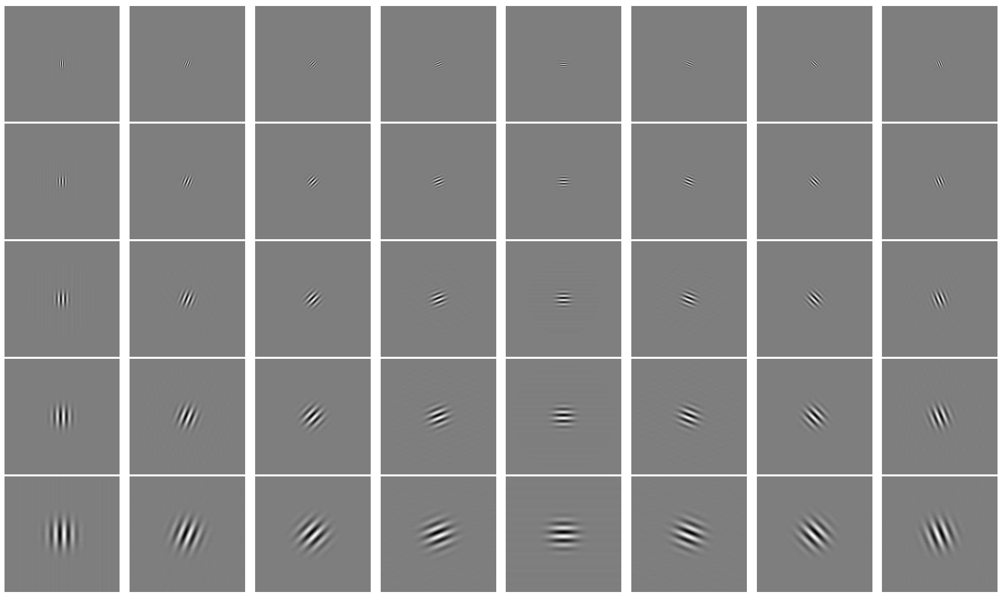
Figure 3.2: The imaginary part of the 5 × 8 Gabor wavelets.

Two-dimensional (2-D) convolution [144] is a specific type of local neighbourhood operation which belongs to the linear approach in image analysis. The 2-D convolution g of two-dimensional functions f and h is denoted by $f * h$. The function of 2-D convolution is expressed as

$$\begin{aligned} g(x,y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(a,b)h(x-a,y-b)\,\mathrm{d}a\,\mathrm{d}b \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-a,y-b)h(a,b)\,\mathrm{d}a\,\mathrm{d}b \\ &= (f*h)(x,y) = (h*f)(x,y) \end{aligned} \tag{3.6}$$

where the operator $*$ symbolise the convolution operator. The 2-D convolution is an integral of the functions f and h. The convolution means a linear filtering process using the filter h on the image f. The linear filtering is often used in local image pre-processing.

2D discrete convolution

In the 2D discrete image domain, the linear filtering calculates the image pixel $g(x, y)$ as a linear combination of the pixel value in a local neighbourhood of the pixel $f(a, b)$. The 2-D discrete convolution is described as

$$g(x, y) = \sum_a \sum_b f(a, b)h(x - a, y - b) \tag{3.7}$$

where $f(a, b)$ is the 2-D discrete input image and $h(x, y)$ is so called **convolution mask** or convolution kernel.The convolution mask is often used with an odd number of pixels in rows and columns, such as 3×3, 5×5 and so on. An example [31] of 3×3 convolution mask is ~~like~~

$$\begin{bmatrix} \mathrm{h}_4 & \mathrm{h}_3 & \mathrm{h}_2 \\ \mathrm{h}_5 & \mathrm{h}_0 & \mathrm{h}_1 \\ \mathrm{h}_6 & \mathrm{h}_7 & \mathrm{h}_8 \end{bmatrix}$$

where h_n is the coefficient of the convolution mask. The 2-D convolution result is the linear neighbourhood operation weighted by the corresponding coefficients in the mask h. In practice, fast convolution is used to increase the speed of the convolution. A fast convolution algorithm takes the fast Fourier transform (FFT) of the input image and the convolution mask, multiplies them together, and then performs the inverse fast Fourier transform (IFFT). In this thesis, the 2-D discrete convolution function is provided by OpenCV [87] in implementation.

The Size of Mask

In 2D discrete convolution, the size of a Gabor filtering convolution mask is an important issue. It should be large enough to show the nature of Gabor wavelets. However, it should not be too large, which increases computation cost. For instance, in Figure 3.1, the size of the Gabor convolution masks is 301×301, and the Gabor wavelets with the lowest frequency are shown on the top row. The span of these Gabor masks only possess a small part at the centre of the mask, and the rest area conveys no information. The size of a Gabor mask should be large enough to cover the shape of Gabor wavelet. The size of Gabor wavelet is determined by the spatial extent of Gaussian envelop, which is then determined by the spatial frequency ν and the deviation of Gaussian function σ in Equation 3.1. According to Dunn *et al.* [35], the Gabor filter is truncated to six times ~~of~~ the span of Gaussian function. The span of Gaussian function is $\frac{\sigma}{\|k_\nu\|}$ and $k_\nu = \frac{k_{max}}{f^\nu}$, so that the Gabor mask is truncated to a width W

$$W = \frac{6\sigma}{\|k_\nu\|} + 1 = 6f^\nu \frac{\sigma}{k_{max}} + 1 \tag{3.8}$$

Taking $k_{max} = \frac{\pi}{2}$, the factor $f = \sqrt{2}$ and the standard deviation of the Gaussian $\sigma = 2\pi$, the width is

$$W = 24 \cdot 2^{\frac{\nu}{2}} + 1 \tag{3.9}$$

For five different spatial frequencies $\nu \in \{-1, \ldots, 3\}$, the corresponding size of Gabor filtering masks are 19×19, 25×25, 35×35, 49×49 and 69×69. ~~Figure 3.3 with the $\frac{3\pi}{8}$ orientation, the corresponding real masks with the different spatial frequencies.~~

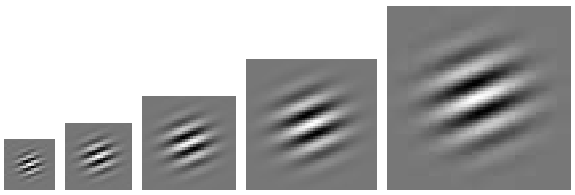

Figure 3.3: When orientation $\mu = 3$, *i.e.*, $\frac{3\pi}{8}$,the corresponding Gabor filtering convolution masks with the five different spatial frequencies $\nu \in \{-1, \ldots, 3\}$.

Magnitude Response

To get Gabor wavelet transform of an image, the 40 Gabor wavelets are convolved with the image. Let $I(z)$, where $z = (x, y)$ define the position in the image, the convolution of image I and a Gabor kernel $\psi_{\mu,\nu}$ is defined as

$$O_{\mu,\nu}(z) = I(z) * \psi_{\mu,\nu}(z) \tag{3.10}$$

where $*$ denotes the convolution operator, and $O_{\mu,\nu}$ is the convolution result corresponding to the Gabor kernel at the orientation μ and the spatial frequency ν[2]. Since Gabor wavelets ~~is~~ of the complex form, ~~so that~~ the convolution results contain the real response and imaginary response as follow

$$O_{\mu,\nu}(z) = \Re\{O_{\mu,\nu}(z)\} + i\,\Im\{O_{\mu,\nu}(z)\}$$

where $\Re$ represents the real response and $\Im$ represents the imaginary response. The real response of Gabor filtering is an image $I(z)$ convolved with the real unit of Gabor kernel described as (3.4). The real response of Gabor filtering is

$$\Re\{O_{\mu,\nu}(z)\} = I(z) * \Re\{\psi_{\mu,\nu}\} \tag{3.11}$$

The imaginary response is the image convolved with the imaginary part described as Equation 3.5. It is expressed as

$$\Im\{O_{\mu,\nu}(z)\} = I(z) * \Im\{\psi_{\mu,\nu}\} \tag{3.12}$$

Given a face image show~~ed~~ in Figure 3.4 in the **FERET** database [120], the

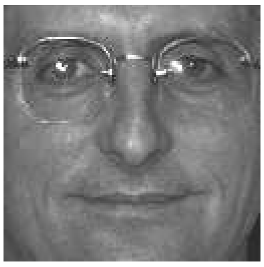

Figure 3.4: A face image selected from the **FERET** database.

40 Gabor real responses and imaginary responses displayed in Figure 3.5 and

[2]In some cases, the spatial frequency ν is also called scale.

Figure 3.6 respectively. From Figures 3.5 and 3.6, it is hard to see ~~any~~

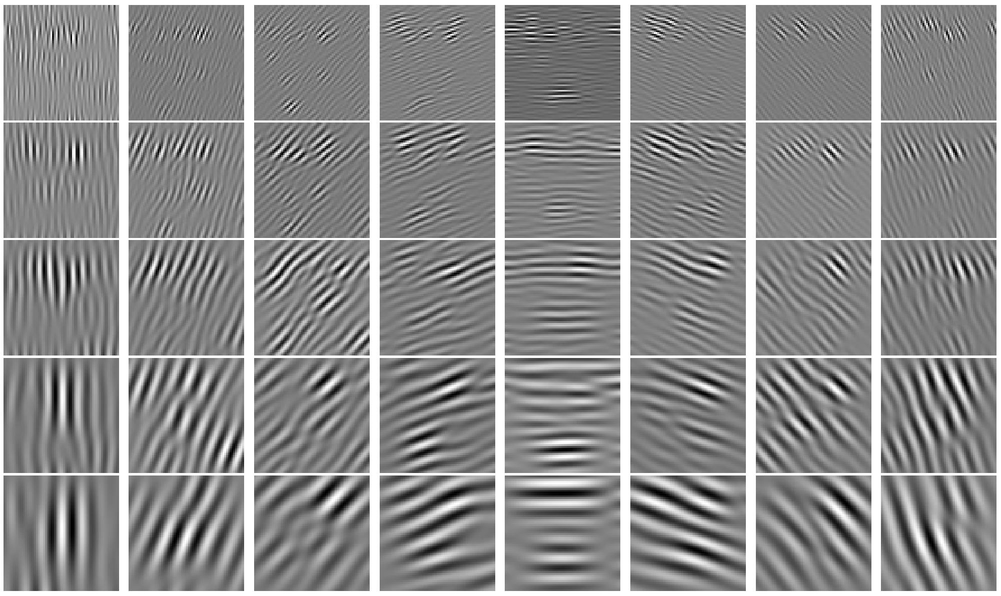

Figure 3.5: The 40 real response images.

human face~~s~~ rather than the stripes across the images.

Texture detection can be operated based on the magnitude of the output of the Gabor filtering [21]. The magnitude response of Gabor filtering is widely used in texture detection. Since human face contains various texture, the magnitude response of Gabor filtering will enhance the recognition ~~on~~ face. It is the square root of the sum of the squared real response and imaginary response, such as

The figure is at least two pages away

$$\|O_{\mu,\nu}(z)\| = \sqrt{\Re^2\{O_{\mu,\nu}(z)\} + \Im^2\{O_{\mu,\nu}(z)\}} \tag{3.13}$$

It can be seen that the magnitude response is modulus. These 40 Gabor magnitude responses are shown in Figure 3.7 ~~as~~ to the image shown in Figure 3.4. The magnitude responses demonstrate local variance within low spatial-frequency and global variance within high spatial-frequency. In the top rows, more precise dissimilarity across the face are shown, while in the bottom rows, more high-level scale dissimilarity across the face are shown. In this

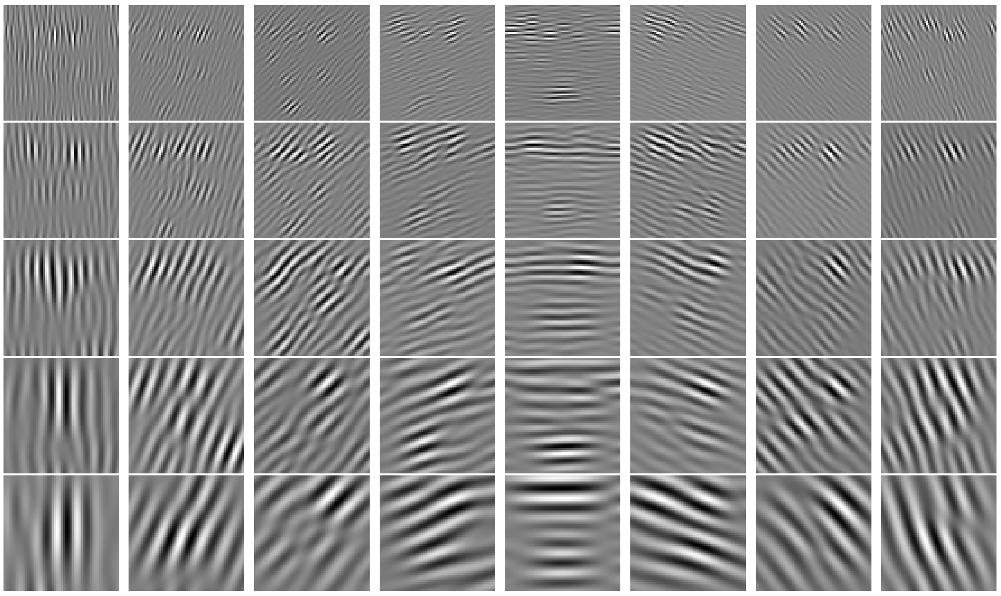
Figure 3.6: The 40 imaginary responses

thesis, the magnitude response of Gabor filtering is adopted to extract Gabor Wavelet Features.

3.1.4 Gabor Wavelet Feature

As stated before, Gabor wavelet feature j is configured by three key parameters: the position $z = (x, y)$, the orientation ν, and the spatial frequency μ. The value of a Gabor wavelet feature is the corresponding magnitude response of Gabor wavelet transform ~~as~~

$$j(z, \nu, \mu) = \|O_{\mu,\nu}(z)\| \tag{3.14}$$

Gabor wavelet features vary in orientation, frequency and position. There are eight different orientations from 0 to $\frac{7\pi}{8}$ with the interval $\frac{\pi}{8}$, and five different scales from -1 to 3. These 40 Gabor wavelets are applied on every position on a face image, so that the total number of Gabor wavelet features is determined by the number of orientations, the number of scales, and the resolution (size) of the image applied. For example, ~~there is~~ an image with 64×64 pixels,

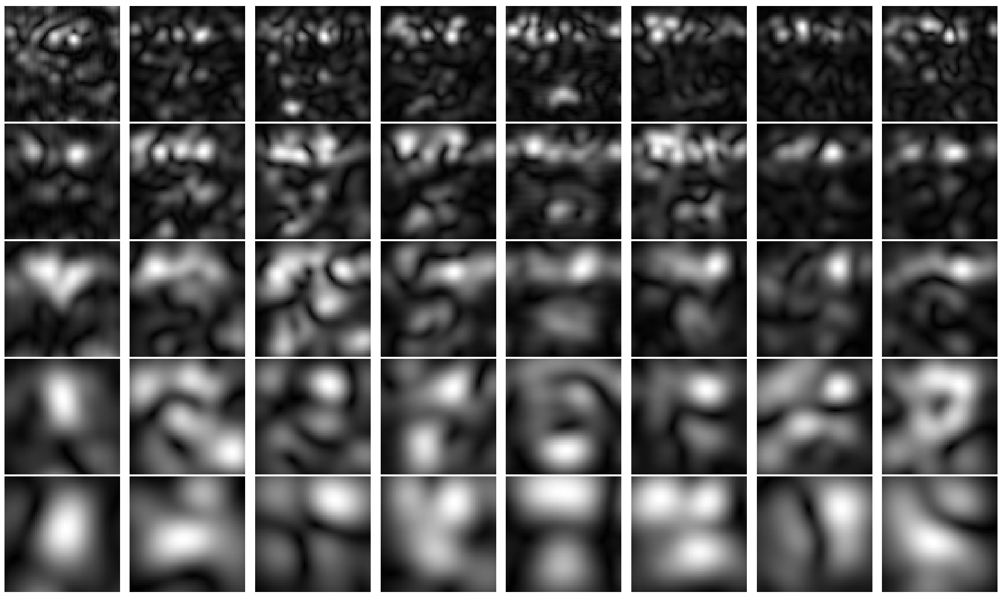

Figure 3.7: The 40 magnitude responses

~~and~~ the total number of Gabor wavelet features is $64 \times 64 \times 5 \times 8 = 163,840$. If the number of Gabor wavelets is fixed, the only factor which change the total number of features is the size of ~~images.~~

3.2 AdaBoost

AdaBoost, stand for Adaptive Boosting, is a machine learning algorithm, formulated by Freund and Schapire [43, 44, 136]. ~~The~~ AdaBoost can be used in conjunction with many other learning algorithms to improve their performance~~,~~ learning accuracy.

This section gives a full landscape of AdaBoost. In Section 3.2.1, ~~the~~ brief introduction of AdaBoost with a horse-racing analogy is given. Section 3.2.2 presents the AdaBoost algorithm, and Section 3.2.3 describes how a "strong" AdaBoost classifier is constructed by a number of "weak" learners. Finally, an experiment of AdaBoost training and classification is given in Section 3.2.4 in the Iris dataset.

Don't understand what "releasing" means in this context

3.2.1 ~~The~~ Introduction to AdaBoost

Boosting is a general method which is used to “boost” the accuracy of any given learning algorithm. Boosting is originated in a theoretical framework for machine learning called “PAC” (Probably Approximately Correct) learning model [154]. After the PAC releasing, there is a discussion whether a “weak” learning algorithm performing just slightly better than random guessing can be boosted into an arbitrarily accurate “strong” learning algorithm. Kearns and Valiant [77] are first to respond to the discussion with the positive answer. In 1989, Schapire proposed the first provable polynomial-time boosting algorithm [134]. Boosting is firstly applied to real-world application for an Optical Character Recognition (OCR) task, relying on neural networks as base learners in [34]. Although Freund and Schapire [43, 44, 136] claim~~s~~ that AdaBoost often tends not to overfit when running for a large number of iterations, other simulations [56] on data sets with higher noise content could clearly show overfitting side-effects. Due to the accuracy of Boosting, the algorithm has been extended for regression [45], multi-class classification [175], and unsupervised learning [131]. Recently, Boosting has been successfully implemented in various applications. OCR [34] is implemented, which used boosted classifier of neural network. In Human face detection [156], AdaBoost was used to train a “strong” classifier to detect human faces in an image [156, 157].

Don't follow the last part of this sentence

Horse-racing analogy An analogy [43, 44, 136] between the AdaBoost algorithm and horse-racing gambling is drawn for a better understanding on AdaBoost. A horse-racing gambler, whose purpose is to maximise his winning, wants to create a computer program helping ~~him~~ betting. The program would accurately predict the winner of a horse race. To create such a program, he consults a successful horse-racing gambling expert for the optimal rule to bet. However, the expert fail to answer, because the expert can not articulate a grand set of rules for choosing a winning horse. On the other hand, when the gambler provides the specific data of races to the expert, the expert can easily come up with a “rule of thumb” for the set of races. The possible “rule of thumb” is like “betting on the horse that has recently won the most races” or “betting on the horse with the most favoured odds”. The rule of thumb is very rough and moderately inaccurate, but the rule is quite reasonable. The rule can be easily concluded from the specific data rather than from a random guess. If the gambler keeps on asking the expert on different collections of races, the expert would draw many rules of thumb. To get the maximum advantage from these rules of thumb, two problem must be resolved.

- How could the gambler choose the collections of races which are given to the expert for extracting the most useful rules of thumb?
- When many rules of thumb are extracted, how these rules combined into one single accurate rule?

3.2.2 AdaBoost algorithm

AdaBoost is an efficient method of producing a highly accurate predication rule by combining a set of rough and moderately inaccurate rules of thumb. In general, the so-called “rule of thumb” is simply designed rules which give low accuracy over long-term observation. For computer vision and pattern recognition, the word “rule” is replaced by the word “classifier”, “learner”, “hypothesis”, etc. In this thesis, the term “learner” is used. For the term “rule of thumb”, it is called ~~“weak learner”~~. AdaBoost refers to a method of combining a set of “weak” learner into a “strong” classifier which gives a high accuracy for prediction. In order to design the “strong” classifier, ~~there are~~ two issues need to be resolved

- How is a set of “weak” learners organised into a “strong” classifier?
- How is a “weak” learner ~~is~~ selected so that it ~~is~~ contribute~~d~~ to a “strong” classifier?

AdaBoost resolves these two problems and gives the better performa~~cn~~e of classification.

Numbering has got mixed up with 3.19 here between 3.14 and 3.15

The AdaBoost algorithm is introduced to solve the difficulties of the earlier boosting algorithm. AdaBoost is an *adaptive* boosting algorithm, because AdaBoost adapts to the error rates of individual "weak" classifier. Pseudocode for AdaBoost is given in Table 3.1.

Training Data There are n examples in the training set $(x_1, y_1), \ldots, (x_n, y_n)$. x is the data of the example which can be integer or real, positive or negative. y is the label of the example, and it is assumed that $y \in \{0, 1\}$. In this chapter, only the two-class classification is discussed[3]. AdaBoost maintains a collection of weights on each example. All the weights ω_t are kept as a probability distribution D_t.

$$D_t(i) = \omega_{t,i} \tag{3.19}$$

At the beginning, the distribution D_t is uniform such that all weights are equal to $1/n$.

Weak Learner AdaBoost processes "weak" classifiers repeatedly in $t = 1, \ldots, T$ iteration. In each iteration, a "weak" learner h_t is trained by using the distribution D_t. To train a weak learner, there are many approaches, *e.g.*, Naive Bayes [33], Perceptron [130], Linear discriminant analysis (LDA) [105], Neural network [126], etc. When h_t is trained, the calculation of the error ε_t is shown in Equation 3.15. When an example (x_i, y_i) is misclassified by the weak learner h_t, *i.e.*, $h_t(x_i) \neq y_i$, the weight $\omega_{t,i}$ is added in error ε_t. When an example is classified, the weight will not be counted in ε_t. The smaller the ε_t is, the stronger is the weak learner h_t. Normally, the error ε_t is relative large, but not exceed 0.5. In practice, a weak learner is a learning algorithm that can use the weights $\omega_{t,i}$ on the training data.

Importance Once a weak learner has been trained, AdaBoost will determine a parameter α_t, which indicates how important the corresponding weak learner h_t is in the final classifier $H(x)$. When α_t is larger, the corresponding weak learner h_t contributes more in $H(x)$. Note that $\alpha_t \geq 0$ if $\varepsilon_t \leq \frac{1}{2}$, such that α_t gets larger as ε_t gets smaller.

[3]~~The c~~hapter 6 gives the description of multi-class classification.

Table 3.1: The AdaBoost algorithm

```
1: Given the training set $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i$ is the data of the
   $i$th example, and $y_i \in Y = \{0, 1\}$.
2: Initialise the weights $\omega_{1,i} = \frac{1}{n}$ for each example $(x_i, y_i)$.
3: for $t = 1, \ldots, T$ do
4:     Train a weak learner $h_t$ using the weights $\omega_{t,i}$.
5:     Get the weak learner with error
           $$\varepsilon_t = \sum_{i=0}^{n} \omega_{t,i} \| h_t(x_i) - y_i \|^2 \quad (3.15)$$
6:     Choose $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\varepsilon_t}{\varepsilon_t} \right)$
7:     Update the weights
           $$\omega_{t,i} = \omega_{t,i} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \quad (3.16)$$
8:     Normalise the weights
           $$\omega_{t+1,i} = \frac{\omega_{t,i}}{\sum_{i=1}^{n} \omega_{t,i}} \quad (3.17)$$
       where the normalisation can be expressed by a normalisation factor $Z_t$
       so that all $\omega_{t+1,i}$ will keep a probability distribution.
9: end for
10: The final "strong" classifier:
           $$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \quad (3.18)$$
```

Updating The weight $\omega_{t,i}$ is updated using the rule shown in Equation 3.16. The effect of the updating weight is to increase the weight of examples misclassified, and to decrease the weight of correctly classified examples. In this way, AdaBoost is modulated to focus on those examples which are "hard" to ~~be~~ classifi~~ed~~.

Normalisation The updated weights are normalised by a normalisation operator Z_t. After updating the weights, all the weights may not constitute a probability distribution, *i.e.*, the sum of all weights may not be equal to 1. To forge the weights into a probability distribution, all the weights for the next iteration will be normalised according to Equation 3.17. $\omega_{t+1,i}$ is increased to the example x_i over other examples, when x_i is misclassified and other examples are classified correctly in the tth iteration. In general, the weights only increase or decrease in a relative manner.

Final classifier The final "strong" classifier $H(x)$ is a weighted majority vote of the T weak learners where α_t is the weight assigned to h_t.

Training error AdaBoost is concerned to reduce the training error in the most basic theoretical properties. The error ε_t of h_t can be replaced by $(\frac{1}{2} - \gamma_t)$. γ_t measures how much h_t is better than random guessing. The training error of the final classifier H is at most

$$\prod_t [2\sqrt{\varepsilon_t(1-\varepsilon_t)}] = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2} \tag{3.20}$$

From Equation 3.20, if each weak learner is slightly better than random guessing, i.e. $\gamma_t > \gamma$ for some $0 < \gamma < \frac{1}{2}$, then the training error drops exponentially fast [43].

3.2.3 Construction of "Strong" AdaBoost Classifier

In [107], it gives a landscape how a strong classifier is constructed. Figure 3.8 displays this with an example. The original training data is shown in the

What's this?
Maybe "can be
formulated as"

top-left of Figure 3.8. There are many points which represent the training examples: blue examples are labelled as the positive data and red examples are labelled as the negative data. The positive examples form as a random Gaussian distribution $N(0,1)$ with the centre at the origin 0 and standard deviation 1. Outside of the positive examples, the negative examples encircle the positive examples, which foumaleted as $\frac{1}{r\sqrt{8\pi^3}}e^{-1/2(r-4)^2}$ with r indicating the radius of the negative data. The original training data is not linearly separated, but non-linearly separable.

In AdaBoost, when $t = 1$, the algorithm is to find a weak learner h_1. As there are many possible solution for the weak learner, an optimal weak learner is found by minimising the weighted training error ε_1. The nature of perceptron makes the weak learner h_1 exhibit as a line in the top-right of Figure 3.8. The line separates the training examples into two categories: plausible positive and plausible negative. As the left of the second row in Figure 3.8, the examples are classified by the first weak learner, the green area reflects the plausible negative, and the yellow area reflects the plausible positive examples. In the iteration $t = 2$, the weak learner h_2 is shown in the right of the second rows of Figure 3.8. Because the importance α_2 on the iteration is not significant, the classification result still ~~keeps~~ same as the first iteration, *i.e.*, the plausible positive and the plausible negative area ~~still keep~~ same. Also from Figure 3.9, it shows that the classification errors on the first and the second iterations are same. When the algorithm comes to the third iteration, the third weak learner h_3 is found and displayed in the left of the third row in Figure 3.8. The weak learner h_4 outlines the right-bottom boundary of the negative examples. The positive examples are constrained between the weak learner h_1 and h_2. The plausible positive area covers most positive examples, and the plausible negative area covers most negative examples, so that the classification error in Figure 3.9 drops very significantly in the third iteration. When AdaBoost comes into the iteration $t = 4$ as shown in the right of the third row in Figure 3.8, the fourth weak learner is found. The weak learner h_4 defines the left boundary of the negative examples, so that many negative examples are misclassified as ~~the~~ positive examples. In Figure 3.9, the classification error grows rapidly. When the

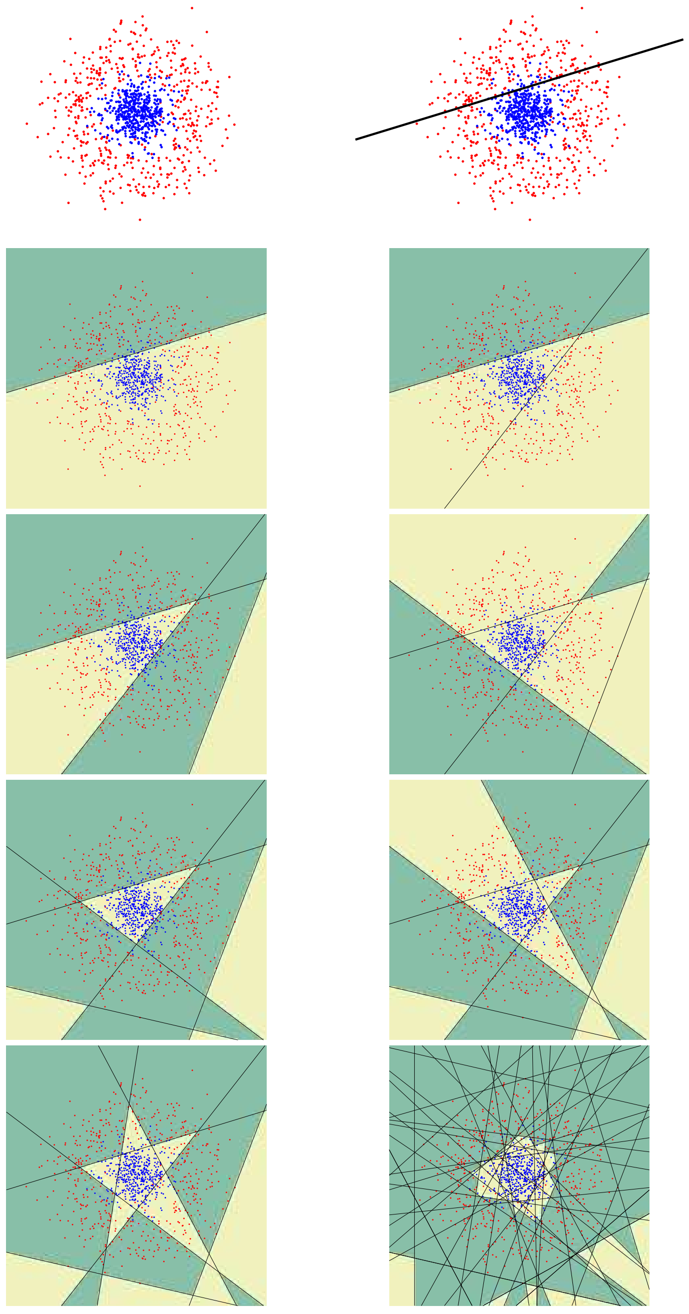
Figure 3.8: Construction of the strong AdaBoost classifier by weak learners

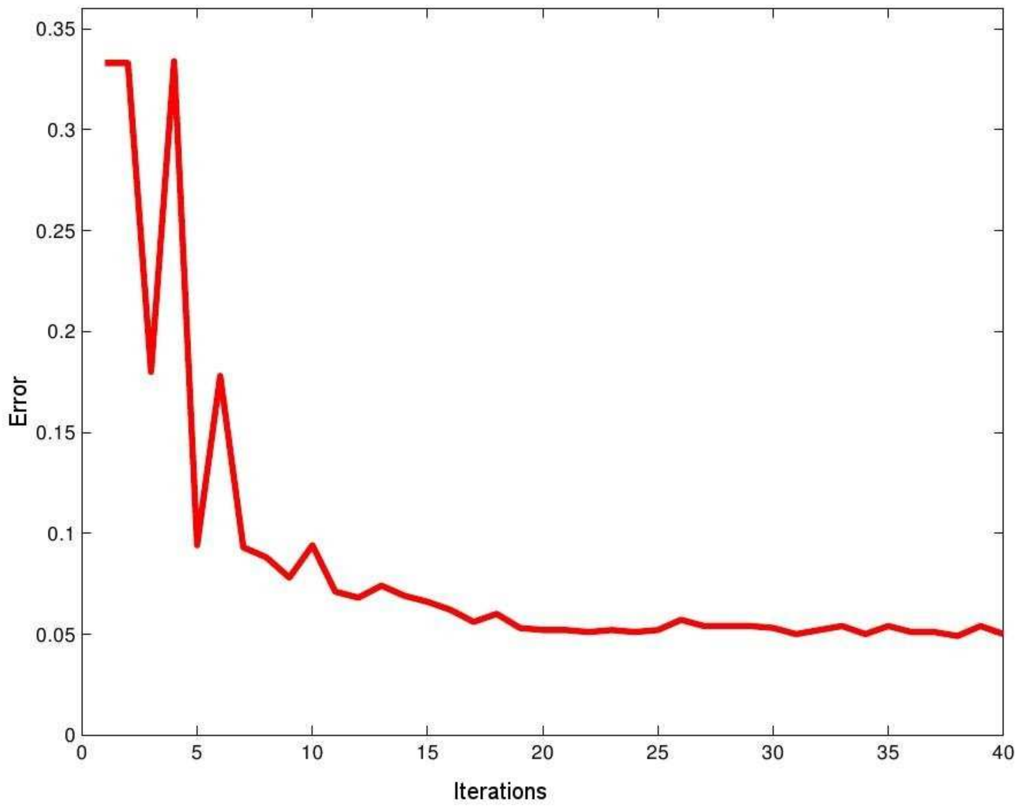

Figure 3.9: Classification error converge

algorithm comes into the iteration $t = 5$ as shown in the left of the fourth row in Figure 3.8, the positive examples are constrained in a triangle area, and the classification error drops again. Finally, AdaBoost goes into the iteration $t = 40$, the positive examples are outlined by many weak learners, and the classification error converge to 0.05.

From Figure 3.8, it is seen that AdaBoost uses many weak learners to fit the training data. The weak learners outline the boundary of the positive examples.

this hyphenation is horrible!

3.2.4 Iris dataset Experiment

To see how the performance can be boosted, Iris dataset is applied in Ad-aBoost training and classification. Two weak learners are used in the AdaBoost training to observe the boosted performance. These two weak learners are ~~AAN~~ weak learner and Naive Bayes weak learner. Also, the two types of weak learners are compared according to the accuracy and the speed of training.

This section is organised as follow: ~~Section 3.2.4~~ discusses two methods for weak learners using weights in the AdaBoost training. ~~Section 3.2.4~~ introduces the ANN weak learner, ~~while Section 3.2.4 introduces~~ the Naive Bayes weak learner. The Iris data set is described ~~in Section 3.2.4. F~~inally, the results are discussed ~~in Section 3.2.4.~~

Weak Learners in AdaBoost

Weak learners deal with training examples and the distribution D_t, rather than only training examples. In practice, there are two ways to process the examples and the weights.

- The weak learner may be an algorithm that uses the distribution D_t on the training examples.
- The training examples are re-sampled according to the distribution D_t, and these (unweighted) re-sampled examples are used to train the weak learner.

The most straightforward way to train a weak learner with training examples with respect to the weight is RPROP[126] Artifical Neural Network algorithm. In OpenCV, RPROP algorithm has been implemented as a class with an interface for inputting examples and corresponding weights. The alternative way is to adapt a Naive Bayes classifier as the weak learner, and resample the examples according to the weights.

Weak Learner: ANN:RPROP

An ~~artificial neural network (ANN)~~ is a mathematical model or computational model based on biological neural networks. It contains an interconnected group of artificial neurons. These are essentially simple mathematical models defining a function $f : X \rightarrow Y$. Each type of ANN model corresponds to a class of such functions.

In this chapter, OpenCV Machine Learning (ML) library was adapted in implementation. The ML library implements feedforward ANN. More particularly, the ANN ~~is~~ multi-layer perceptions (MLP). MLP consists of the input layer, output layer and one or more hidden layers. Each layer of MLP contains one or more neurons that are directionally linked with the neurons from the previous and the next layer.

All the neurons in MLP are similar. Each neuron has several input, *i.e.*, it takes the output values from several neurons in the previous layer ~~on input~~. Each neuron has several output, *i.e.*, it passes the response to several neurons in the next layer. The values computed from the previous layer are summed with certain weights, individual for each neuron, plus a bias term, and the sum is transformed using an activation function.

The larger the network size (the number of hidden layers and their sizes), the more is the potential network flexibility, and the error on the training set could be made arbitrarily small. But at the same time the learned network will also "learn" the noise present in the training set, so the error on the test set usually starts increasing after the network size reaches some limit. Besides, a larger network takes longer time for training than a smaller one. To build an ANN weak learner, it is not necessary to design a large network, but a simple network. For an ANN:RPROP weak learner, the input layer includes the number of neurons equal to the number of elements in the examples, *e.g.*, an example containing m digits (an m length vector), has m neurons in the input layer. There is one hidden layer including two neurons to keep the network structure as simple as possible. The output layer includes one neuron which gives positive or negative results. Figure 3.10 shows the ANN structure.

The RPROP (Resilient backpropagation) [126] algorithm has shown to perform well. The ML library in OpenCV implements a batch RPROP algorithm for training MLPs.

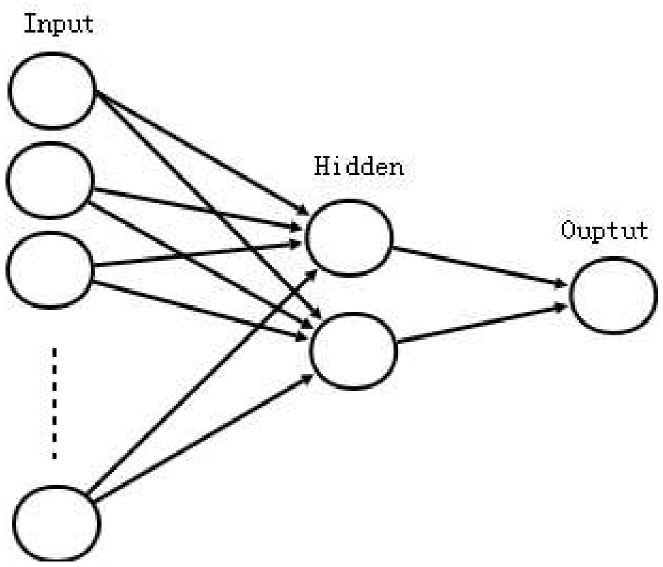

Figure 3.10: The ANN layers and nodes structure.

Weak Learner: Naive Bayes

A naive Bayes classifier is a simple probabilistic classifier in which Bayes' theorem is applied with strong independence assumptions. Depending on the precise nature of a probability model, naive Bayes classifiers can be trained efficiently in a supervised learning setting. Inspite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers often work much better in many complex real-world situations than ~~that~~ one might expect. An advantage of the Naive Bayes classifier is that it requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix. This simple classification model assumes that examples with ~~a~~ same label are normally distributed (though, not necessarily independently distributed). In two-class classification, the naive Bayes model supposes that positive examples form a normal distribution, ~~so as~~ negative examples.

In statistics, there are four definitions to explain the resampling process.

The first one is to estimate the precision of examples' statistical properties.

To put the weights into a naive Bayes weak learner, the training examples should be resampled, *i.e.*, resampling ~~is~~ to generate a new training set according to the given distribution. Some new training examples are generated in the process of resampling. The number of new examples generated depends on the probability. If the probability is high, more new examples are generated, and vice versa. The new examples share the same values and labels with the examples associated with the probability. The number of newly generated examples can be calculated by

$$n' = \lfloor \omega_{t,i} \times n \times fa \rfloor \tag{3.21}$$

where $\omega_{t,i}$ ~~is~~ the weight of corresponding example (x_i, y_i), also represents the probability of the example. n is the number of examples in the original training set. fa is an amplified scale factor which decides how the original set is expanded. $\lfloor x \rfloor$ denotes a function that returns the highest integer less than or equal to x. The amplified scale factor fa should not be small, otherwise extra new examples can not be generated. Also, the factor fa should not be too large, otherwise the total number of examples in the new set will be ~~resided in very~~ large ~~size~~, which leads high computational cost in training the weak learners. In this section, the factor is set as $fa = 50$, such that the new training set ~~not only is in a~~ moderate size which does not cause high computational cost, but also the examples reflect the probability distribution.

In the first iteration of AdaBoost training, there is no need to resample the training set. After the first iteration, the weight for each example is updated. The distribution D_t will not ~~keep as~~ uniform distribution. The weights for some examples might be greater than those for other examples. Hence, the resampling process is required after AdaBoost training starts at the second iteration, and so on.

Iris flower data set

The Iris flower data set (also called Fisher's Iris data set) [7, 40] is a multivariate data set as an example of discriminant analysis. It is sometimes called Anderson's Iris data set as Edgar Anderson collected the data to quantify the geographic variation of Iris flowers in the Gaspe peninsula. The dataset consists of 50 examples from each of three species of Iris flowers, which are setosa, virginica and versicolour. Four features were measured from each example, they are the length and the width of sepal and petal. Based on four features, the species of the flower can be determined. The plot of the Iris flower data set is shown in Figure 3.11

Experiment results

The Iris flower data set is tested by the AdaBoost algorithm with implementing both the ANN weak learner and the Naive Bayes weak learner. There are three classes in the iris flower data set, i.e., setosa, versicolor and virginica. Table 3.1 describes the AdaBoost algorithm in a two-class classification mechanism, so that to test the Iris flower data set according to Table 3.1 the data set is divided into three scenarios: setosa versus virginica, setosa versus versicolour, and virginica versus versicolour. In each example of the data set, all four features are used in AdaBoost training.

The classification error of the final strong classifier is expected to be small enough, *i.e.*, close to 0. In the setosa versus virginica scenario and setosa versus versicolour scenario, AdaBoost training with both ANN and Naive Bayes weak learners are finished at the first iteration. The first weak learner (both ANN and Naive Bayes) achieves zero error rate, so that the data set is well trained and no need for ~~the~~ further training. From Figure 3.11, it can be observed that the cluster of the setosa examples are distant from the cluster of the virginica and the versicolour. The setosa examples are perfectly separated from other examples. Therefore, AdaBoost can easily train a final strong classifier only using one weak learner. For the data which is well separated, AdaBoost is used as a classification solution, but due to the computational cost and design complexity, it may not be an appropriate

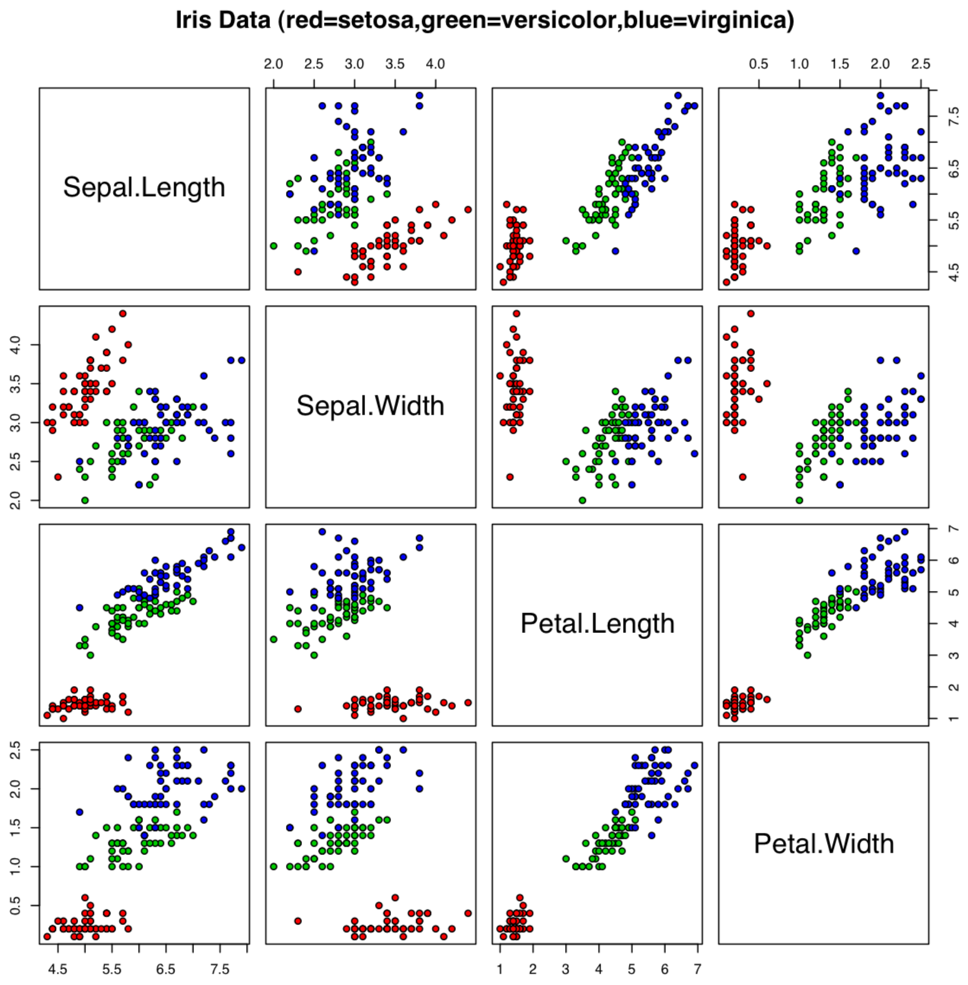

Figure 3.11: The plot of Iris flower data set

solution for the type of problem.

In Figure 3.11, the virginica and versicolour examples are adjacent. In the feature space projected by the length and the width of the sepal, the clusters of these two classes ~~are overlapped~~. In the iris data set, classification on virginica versus versicolour scenario is difficult. With the ANN weak learner, AdaBoost takes nine iterations to achieve zero error rate. The final strong classifier H is made of nine ANN weak learners $\{h_1, \ldots, h_9\}$ with the corresponding importance. ~~With the ANN weak learner, the overall error rate is converged to 0.0 after nine iterations.~~ The blue line in Figure 3.12 shows changes of the overall error rate of combined strong AdaBoost H at ~~different~~ iteration. Looking at the error change globally, the error rate is

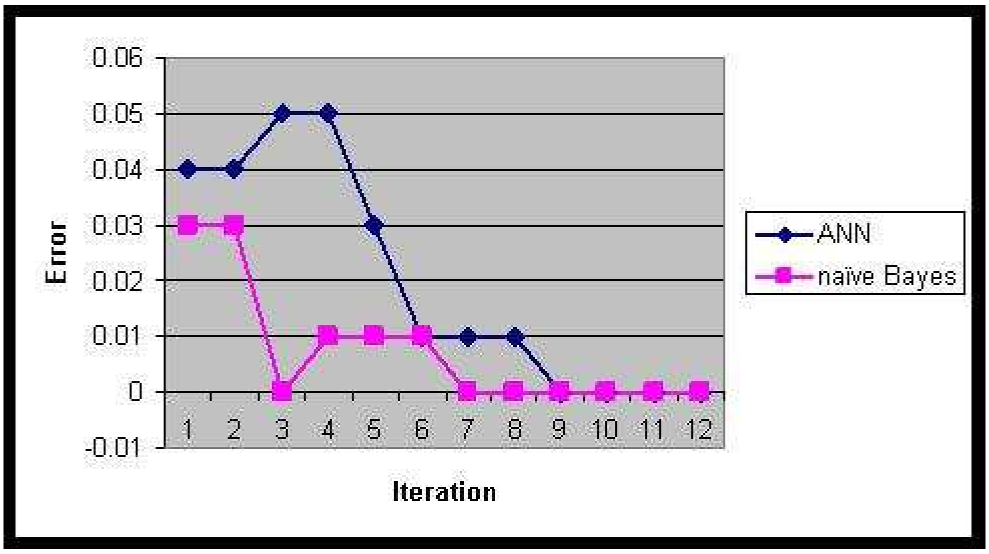

Figure 3.12: The error rates on different iterations with the ANN weak learner and the Naive Bayes weak learner.

eventually decreasing, although in some iteration, the error rate has ~~been raised.~~

With the Naive Bayes weak learner, AdaBoost takes seven iterations to form a final classifier H and its error rate reaches zero as the pink line shown in Figure 3.12. Although on the third iteration, the error has already reached 0.0, the training would not stop at ~~the~~ iteration. ~~It~~ is due to small example size (50 examples for each class) and inaccuracy ~~existence~~ in the training set. The error rate is eventually decreasing, although in some iterations, errors have ~~been raised~~. The examples in the new training set ~~is~~ added by applying the resampling technique in each iteration. The number of new examples is decided by the weight of original examples and the amplified scale factor. The number of new examples in the iterations 1 to 7 is 100, 2596, 3969, 3343, 4247, 2911 and 3787

From the result, AdaBoost is capable of performing classification on Iris flower data set. The results from two different weak learners the ANN and the Naive Bayes are compared. From the perspective of accuracy, the Naive Bayes weak learner is slightly better than the ANN weak learner with less iteration to construct a strong classifier. However, due to the time consum~~ing~~ on resampling and training of a larger training size, the Naive Bayes weak learner is more computational expensive than the ANN weak learner. Hence, both weak learners are appropriate for AdaBoost learning in ~~the~~ case.

The test tastes the functions of AdaBoost classification, which displays the performance of any classification rules can be boosted from it.

Odd choice of word - not sure what you are trying to say."samples" perhaps?

3.3 Summary

In this chapter, two state-of-the-art [illegible] computer vision and machine learning approaches, *i.e.*, Gabor wavelets and AdaBoost, are introduced. The Gabor wavelets are of similar shape as the receptive fields of simple cells in the primary visual cortex, and demonstrate significant potential in biometrics such as iris recognition and face recognition. AdaBoost is one of the most successful and popular learning algorithms. The boosting process is to construct a "strong" classifier from "weak" learning algorithms. By combining these two approaches in face recognition, the performance of Gabor-Boosting algorithm is boosted.