THREE PROBLEMS IN IMAGING SYSTEMS: TEXTURE RE-RENDERING IN ONLINE DECORATION DESIGN, A NOVEL MONOCHROME HALFTONING ALGORITHM, AND FACE SET RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Tongyang Liu

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

First of all, I want to thank my advisors Dr. Qian Lin and Professor Jan P. Allebach whose wonderful advise and tutoring in the projects that I participated helped me get into the field of deep learning, computer vision, image processing and halftoning fast and think sharply about new ideas. My advisors' attitudes toward work and their hard-working spirit will always inspire me to work harder and harder and think harder and harder.

I want to thank my family. I have not spent much time with my parents ever since I came to US to pursue my PhD degree. I wish they can share the joy of my academic achievements.

I want to thank all friends and lab mates. They have greatly expanded my vision of knowledge and they have had lots of encouragement and inspirations for my research.

I want to specially thank HP Labs Palo Alto, HP Inc Boise and DzineSteps for their support for my research projects. They not only provide me with valuable funding support but more importantly also provide me with opportunities to participate in the exciting industry engineering projects and giving me chances of integrating my research topics into their industrial products design. This is awesome.

I want to thank my committee professors for reviewing my work and giving me valuable feedback. I really appreciate that.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Liu, Tongyang PhD, Purdue University, May 2019. Three Problems in Imaging Systems: Texture Re-rendering in Online Decoration Design, a Novel Monochrome Halftoning Algorithm, and Face Set Recognition with Convolutional Neural Networks. Major Professor: Qian Lin, Jan P. Allebach.

In this thesis, studies on three problems in imaging systems will be discussed.

The first problem deals with re-rendering segments of online indoor room images with preferred textures through websites to try new decoration ideas. Previous methods need too much manual positioning and alignment. In the thesis, a novel approach is presented to automatically achieve a natural outcome with respect to indoor room geometry layout.

For the second problem, the laser electrophotographic system is eagerly looking for a digital halftoning algorithm that can deal with unequal printing resolution, since most halftoning algorithms are focused on equal resolution. In the thesis, a novel monochrome halftoning algorithm is presented to render continuous tone images with limited numbers of tone levels for laser printers with unequal printing resolution.

For the third problem, a novel face set recognition method is presented. Face set recognition is important for face video analysis and face clustering in multiple imaging systems. And it is very challenging considering the variation of image sharpness, face directions and illuminations for different frames, as well as the number and the order of images in the face set. To tackle the problem, a novel convolutional neural network system is presented to generate a fixed-dimensional compact feature representation for the face set. The system collects information from all the images in the set while having emphasis on more frontal and sharper face images, and it is regardless of the number and the order of images. The generated feature representations allow direct, immediate similarity computation for face sets, thus can be directly used for

recognition. The experiment result shows that our method outperforms other state-of-the-art methods on the public test dataset.

1. PREFACE

In this thesis, three problems in imaging systems will be discussed. The imaging systems are generally designed for observation or capturing images in a variety of applications. Common imaging systems include printers, cameras, flatbed scanners, mobile phones and flat panel displays [1]. Depending on color types, imaging systems can be either color imaging systems, such as the daily mobile phone and PC monitors that we use, or monochrome imaging systems, like the black and white printers. The color imaging systems nowadays are highly integrated and applied to huge amount of the modern devices. They can be either used as image capturing devices, such as cameras and scanners, or used as interconnection space and processing devices, such as smart phones, tablets, personal computers or cloud, or used as image output devices, such as displays or printers. Considering the high-level integration of imaging systems and the very close cooperation between the systems and the whole device, I have done research on three important problems regarding the imaging systems, to further enhance the imaging systems' integration with modern digital devices and to further increase the imaging systems' capability of handling more complicated application scenarios.

The first problem is about re-rendering textures for online decoration design, which is also known as web images customization. The specific case that we are especially interested in is about customizing images on websites. Specifically, users are able to replace some part of the images which are already on websites with new textures that they prefer, and then create new images by mixing up the replaced texture with the original image. This has become more and more popular as it is very convenient for users to try new decoration ideas without spending huge budget on that in the first place. Prior to the proposed method of this research, users need to put great effort in positioning and aligning the newly rendered texture in order to get a natural feeling

with respect to the correspondence to the room layout from indoor scene images. And in this research I proposed a novel tool that is able to autonomously align the re-rendered textures to room spatial layout. Details will be discussed in Chapter 2.

Monochrome digital halftoning is the process of generating a pattern of binary pixels with a limited number if tone levels that creates the impression of a continuous tone image. In order to achieve high quality illustration for the continuous tone image, the halftone image must contain high spatial frequency patterns of coarsely quantized pixels that are perceived as intermediate gray levels through low pass filters [2]. The human visual system (HVS) through which the halftone pattern is observed has the low-pass filtering characteristic [2]. Therefore, the goal of our research on monochrome digital halftoning is to generate the binary image that correctly reproduce tone and the details for the original image without introducing noticeable artifacts. Most halftoning algorithms are one of the three categories [2]: the first is point processes, typical cases for which is screening or dithering [3] [4] [5] [6]; the second is neighborhood process, typical cases for which is error diffusion [7]; the third is iterative algorithms (least square and direct binary search (DBS)) [8]. Although the quality of the halftone increases with the order, the computational complexity required to generate decent halftone images also increases in this order. In fact, screening and error diffusion can be operated at real time, thus they are widely-used in practical electrophotographic systems. The iterative algorithm, however, cannot be directly integrated into normal printers due to the limit of computational resources. But on the other hand, iterative approach provides us a great capability for offline halftone patterns design. This brings us the idea of using the iterative search method for designing screening algorithms. Such technology is called hybrid screens. There has been plenty of research on hybrid screens [2] [5] [9]. And many of them are based on the property of printer that the printing pixels are square-shaped. In this research, the hybrid screen is developed for the laser printers that have unequal resolution printing pixels. For this kind of case, the original hybrid screen that is based on the equal resolution pixels needs to be revised. Details will be discussed in Chapter 3.

In third part of the thesis a novel Cluster Aggregation Network (CAN) for face set recognition is presented. This network takes a set of face images, which could be either face videos or clusters with a different number of face images as its input, and then it is able to produce a compact and fixed-dimensional feature representation for the face set for the purpose of recognition. The whole network is made up of two modules, among which the first one is a face feature embedding module and the second one is the face feature aggregation module. The first module is a deep Convolutional Neural Network (CNN) which maps each of the face images to a fixed-dimensional vector. The second module is also a CNN which is trained to be able to automatically assess the quality of input face images and thus assign various weights to the images' corresponding feature vectors. Then the one aggregated feature vector representing the input set is formed inside the convex hull formed by the input single face image features. Due to the mechanism that quality assessment is invariant to the order of one image in a set and the number of images in the set, the aggregation is invariant to these factors. Our CAN is trained with standard classification loss without any other supervision information and we found that our network is automatically appealed to high quality face images while repelling low quality images such as blurred, blocked and non-frontal face images. We trained our networks with CASIA and YouTube Face datasets and the experiments on IJB-C video face recognition benchmark show that our method outperforms the current state-of-the-art feature aggregation methods and our challenging baseline aggregation method.

2. WEB IMAGE PERSONALIZED RENDERING

2.1 INTRODUCTION

In this research I propose a novel tool for re-rendering objects in indoor scene images with new textures. It aims to address the problem of too much manual work of positioning and alignment when applying new texture onto an object surface in an indoor scene image. The algorithm of the tool is based on establishing 2D projective transformation between texture images and planar object surfaces in scene images. In order to find the transformation, I use a sampled rectangular texture pattern from a large synthesized planar texture and a planar quadrangle corresponding to object surface orientation estimation, which is generated by a geometric orientation hypothesis framework. The tool also puts effort in adjusting the scaling and reducing artifacts for re-rendered textures. I present the re-rendering results for ceilings, walls, floors, etc. that naturally correspond to room geometry layout.

Digital imaging and rendering technology has brought us tremendous amount of benefits, one of which is the privilege of creating variable media contents through user customizations. One case that is receiving increasing interest these days, is in the application of customizing images on websites. Specifically, users can replace some part of images that is already on websites with new textures that they prefer, and then create new images by mixing up the replaced texture with original images. This kind of customization is known as image personalization [10]. The specific instance of image personalization to which our tool is targeted is the virtual customization of images taken from indoor scenes of residential structures, such as kitchen, living room or bedroom. This kind of customization lets users replace the original textures from object surfaces in the scene with preferred new textures, which is called texture re-rendering and image re-mixing. Figure 3.1 shows examples of texture re-rendering

and image re-mixing. In the figure, the original floor textures and/or side wall textures from the indoor bathroom image are replaced with new types of textures, and then a re-mixed scene image is created. Figure 3.1b and Figure 3.1d separately show us that original appearance of floor and/or side wall is replaced by wooden and brick like textures. The highlighted green lines in Figure 1a illustrate the orientation of parallel lines within original floor texture. And the highlighted yellow lines in Figure 1b illustrate the orientation of parallel lines within replaced wooden-like texture. The brick-like texture in Figure 1d is clearly showing the orientation of parallel lines. By comparing the parallel line orientations in the re-rendered texture in Figure 1b and Figure 1d, we are able to see that the re-rendering result in Figure 1d looks more natural than that in Figure 1b, with respect to the room spatial layout. The position and orientation of the re-rendered texture in Figure 1b apparently needs to be manually adjusted before it appears to natrually corresponds to room spatial layout. Figure 1c shows a different type of texture replacement, where the surface of side walls is replaced with pure color textures. This type of texture re-render, however, can only be suitable to the situation where original textures of the object surface have no complicated texture patterns. Currently, several commercial websites has been devoted to developing web-based interfaces for indoor scene image re-texturing, among which [11]- [12] are quite noticeable. [13] and [14] provide a functionality that is similar to the one shown in Figure 1c. [13] and [12] let users apply preferred textures in a virtual design environment. Although the visualization tool provides quite reasonable rendering result, it is hard to see the effect of newly applied texture in original scene images. As for [11], which provides a functionality as shown in Figure 3.1b, even though their tools are able to let users select various types of textures and render them in the original scene images, the result does not look as if it is naturally corresponding to the geometry orientation of the scene, thus an amount of manual adjustment afterwards is needed. In addition, the zoomed-in view of their rendered texture has poor quality. As a result, a tool is desired that has the following properties: firstly, it supports various types of textures; secondly, it is able to allow

(a) (b)

(c) (d)

Fig. 2.1. Examples of texture re-rendering for the indoor bathroom image. Highlighted lines show the orientation of parallel lines in floor texture. (a) Original scene image taken from bathroom (b) Re-rendered floor and side wall with wooden texture. (c) Re-rendered side-wall with pure color texture. (d) Re-rendered floor with brick texture that naturally corresponds to geometry layout of the room.

the re-rendered texture be directly mixed with the original scene image; thirdly, the re-rendering result natrually corresponds to room layout; finally, it is able to allow high-resolution rendering. Our proposed tool aims at these targets. In [10], the authors proposed a method for inserting text in images based on pinhole camera model with camera parameter estimation. However, text-insertion usually happens within a limited spatial range, thus the error for parameter estimation is not going to greatly affect the alignment and orientation of inserted text. In addition, their approach is based on straight line detection where the lines are either perpendicular

or parallel with each other. This is not applicable to indoor scene images since straight lines in these images do not have to be aligned either horizontally or vertically to each other.

In this paper, we propose a texture re-rendering tool that is based on 2D projective geometry [15], and we are specially focused on establishing the 2D projective transformation from texture images to scene images. To begin with, we adopt a room layout estimation framework proposed in [16], which is based on line sweeping algorithm and convex edge detection to generate layout hypothesis. And then we apply Direct Linear Transform (DLT) algorithm to calculate the transformation matrix for texture mapping. The details are discussed in the rest of the paper.

2.2 PROJECTIVE GEOMETRY

In this section, we briefly introduce projective geometry, which is the study of 2D planar geometry, as the starting point for our texture re-rendering method. And in the following discussions, we denote 2D plane as $\mathbb{P}^2$.

2.2.1 HOMOGENEOUS REPRESENTATION FOR 2D POINTS

The coordinate pair (x, y) can represent a point in 2D plane. Therefore we identify a plane as $\mathbb{R}^2$, and the coordinate pair (x, y) is then identified as a 2D vector, thus a plane can be considered as a vector space. A point $(x, y) \in \mathbb{R}^2$ lies on line (a, b, c) if and only if $ax + by + 1 = 0$, which, in matrix form, can be written as $(x, y, 1) \cdot (a, b, c)^\top = 1$. Here we can see that $(x, y) \in \mathbb{R}^2$ is represented as a 3D vector $(x, y, 1)$. And since it is also true that $k(x, y, z) \cdot (a, b, c)^\top = 0$, we are able to draw the conclusion that (kx, ky, k) and $(x, y, 1)$ represents the same point (x, y) in $\mathbb{P}^2$, thus the 3-vector representative of the form (x_1, x_2, x_3), where $x_3 \neq 0$, is the homogeneous representation for points $(x_1/x_3, x_2/x_3)$ in $\mathbb{R}^2$.

Now we discuss the case when $x_3 = 0$. The homogeneous representation for points in $\mathbb{R}^2$ with the form of $(x_1, x_2, 0)$ is known as ideal points, representing the points

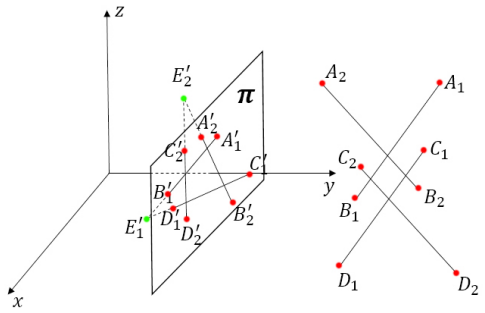


Fig. 2.2. An illustration for projective geometry. Lines A_1B_1, A_2B_2, C_1D_1 and C_2D_2 are lines in 3D world space. A_1B_1 is parallel to C_1D_1 while A_2B_2 is parallel to C_2D_2, also A_1B_1 is perpendicular to A_2B_2. $A_1'B_1'$, $C_1'D_1'$, $A_2'B_2'$ and $C_2'D_2'$ are respective projections on plane π. Note that $A_1'B_1'$ and $C_1'D_1'$ intersect at point E_1', while $A_2'B_2'$ and $C_2'D_2'$ intersect at point E_2'. E_1' and E_2' are vanishing points in plane π. And these vanishing points correspond to orientations of lines in 3D world space.

at infinity. If there are two parallel lines (a, b, c) and (a, b, c') in $\mathbb{R}^3$, the ideal point $(-b, a, 0)$ then lies on both of the two lines. Therefore, we can actually consider ideal point $(x_1, x_2, 0)$ as the intersection of two parallel lines in 3D world, which apparently extends to infinity. Based on the above discussions, we are now able to introduce the concept of vanishing points, as illustrated in Figure 3.2. Note that vanishing points are indeed the projection of ideal points on $\mathbb{P}^2$.

2.2.2 PROJECTIVE TRANSFORMATION BETWEEN TEXTURE IMAGE AND SCENE IMAGE

The core idea for our texture re-rendering tool is to estimate the transformation between pixels in the texture image and pixels in the scene image, which is called texture mapping. And it is actually a projective planar transformation that maps one set of points in $\mathbb{P}^2$ to another set of points in $\mathbb{P}^2$. Figure 3.3 shows one instance of planar transformation from plane π to plane π', and as is shown, point x is mapped to point x'. Now we are going to check the linearity of this transformation. As in

Figure 3.3, $ABB'A'$ is a plane in 3D world that passes through center of projection O, plane π and plane π', which intersects plane π at line AB and plane π' at line $A'B'$. Furthermore, line $A'B'$ is the mapping of line AB from plane π to plane π', thus lines in $\mathbb{P}^2$ are mapped to lines in $\mathbb{P}^2$. Therefore, the projective transformation between two lines in $\mathbb{P}^2$ is a linear transformation on homogeneous 3D vectors, and it indeed can be represented by a non-singular 3 by 3 homogeneous matrix H [15]. In texture mapping, plane π in Figure 3.3 is the texture image, and $\boldsymbol{x}$ represents one pixel in the texture image. Meanwhile, plane π' is the scene image that is to be re-rendered, and $\boldsymbol{x'}$ represents one pixel in the scene image. Note that although $\boldsymbol{x}$ and $\boldsymbol{x'}$ are both points in $\mathbb{P}^2$, we are using homogeneous representations. Therefore $\boldsymbol{x}$ and $\boldsymbol{x'}$ are both 3D vectors. As a result, we denote $\boldsymbol{x}$ as $(x_1, x_2, x_3)^\top$ and vector $\boldsymbol{x'}$ as $(x'_1, x'_2, x'_3)^\top$. Then the transformation between vector $\boldsymbol{x}$ and $\boldsymbol{x'}$ can be expressed as:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.1}$$

Here we use h_{ij} to represent ith row and jth column element of homogeneous matrix H.

2.2.3 CALCULATING ENTRIES OF HOMOGENEOUS TRANSFORMATION MATRIX H

In Section 2.2, we discussed the homogeneous transformation matrix for mapping points between $\mathbb{P}^2$. The next step is how we calculate the elements in homogeneous matrix H. As is indicated in previous sections, we are given a set of points $\boldsymbol{x}$ represented by homogeneous 3D vectors in $\mathbb{P}^2$, and another set of points $\boldsymbol{x'}$ in $\mathbb{P}^2$, which are also represented by homogeneous 3D vectors. The homogeneous matrix H between $\mathbb{P}^2$ is then established such that $\boldsymbol{x'} = H\boldsymbol{x}$.

The first question is how many coordinate sets $(\boldsymbol{x}, \boldsymbol{x'})$ are needed for H calculation. We notice that matrix H has 9 entries, but the number of degrees of freedom for

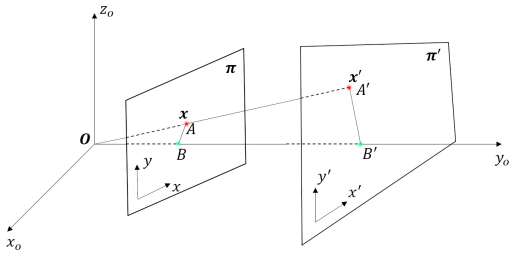


Fig. 2.3. Projective transformation between two planes. $x_0y_0z_0$ is world coordinate frame, and its origin O is the center of projection. $\boldsymbol{x}$ is a point in plane π, and $\boldsymbol{x'}$ is a point in plane π'. The mapping from plane π to π' is a linear mapping H between homogeneous coordinates such that $\boldsymbol{x'} = H\boldsymbol{x}$.

projective transformation between $\mathbb{P}^2$, is indeed 8 [15]. The reason is that suppose we have two homogeneous matrices H_1 and H_2, where $H_2 = aH_1$ and a is a scalar, and then we apply projective transformation on same homogeneous 3D vector $\boldsymbol{x}$. According to Equation 2.1, we will get $\boldsymbol{x'_1} = H_1\boldsymbol{x}$ and $\boldsymbol{x'_2} = H_2\boldsymbol{x}$. Since $H_2 = aH_1$, we have $\boldsymbol{x'_2} = a\boldsymbol{x'_1}$. According to Section 2.1, we already know the fact that homogeneous 3D vectors $\boldsymbol{x'_2}$ and $\boldsymbol{x'_1}$ correspond to the same point in $\mathbb{P}^2$. Therefore, we can draw the conclusion that homogeneous matrices H_1 and H_2 indeed are the same projective transformation between points in $\mathbb{P}^2$. Finally, we are able to draw the conclusion that homogeneous matrix H is defined up to a scalar, thus the 2D projective homogeneous transformation has 8 degrees of freedom.

Now we consider the way of calculating the entries' values for homogeneous transformation matrix H using Direct Linear Transformation (DLT) algorithm [15]. Actually, we can infer the fact from Section 2.1 that 3D homogeneous representation (x_1, x_2, x_3) for a point in $\mathbb{P}^2$ has 2 degrees of freedom. This is because x_3 just stands for an arbitrary none-zero ratio. Furthermore, we also infer from Section 2.2 that the degrees of freedom for point $\boldsymbol{x}$ must correspond to the degrees of freedom for its mapped point $H\boldsymbol{x}$. As a result, one coordinate mapping pair $(\boldsymbol{x}, H\boldsymbol{x})$ actually reduces the total degrees of freedom of the transform by 2. Therefore, in order to get efficient estimation for the entries of homogeneous transformation matrix H, it is

necessary that we obtain 4 pairs of corresponding coordinate points to fully specify the matrix H. Now we set out to solve homogeneous matrix for our texture mapping method whereby it denotes the transformation from pixels of $\mathbb{P}^2$ in texture image to corresponding pixels of $\mathbb{P}^2$ in scene image. Note that here we are using homogeneous 3-vectors for pixels in $\mathbb{P}^2$, as is explained in Section 2.1. Therefore, pixel (x_1, x_2) in texture image is represented as:

$$\boldsymbol{x} = (x_1, x_2, 1)^\top$$

Similarly, pixel (x_1', x_2') in scene image is represented as:

$$\boldsymbol{x'} = (x_1', x_2', 1)^\top$$

Consequently, our texture mapping between $\boldsymbol{x}$ and $\boldsymbol{x'}$ is the homogeneous transformation matrix H such that

$$\begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \tag{2.2}$$

where $h_i, i = 1, ..., 9$ stands for the entries of matrix H. Now we use the denotation $\boldsymbol{h^i}$ to represent row vectors for matrix H, that is:

$$\boldsymbol{h^1} = (h_1, h_2, h_3)$$
$$\boldsymbol{h^2} = (h_4, h_5, h_6)$$
$$\boldsymbol{h^3} = (h_7, h_8, h_9)$$

Since $\boldsymbol{x'} = H\boldsymbol{x}$, we infer that homogeneous vector $\boldsymbol{x'}$ and $H\boldsymbol{x}$ are indeed in same direction. As a consequence, we know that:

$$\boldsymbol{x'} \times H\boldsymbol{x} = \boldsymbol{0} \tag{2.3}$$

where $\boldsymbol{0} = (0, 0, 0)^\top$, is the null 3D vector. And actually $H\boldsymbol{x}$ can be expanded as:

$$H\boldsymbol{x} = \begin{pmatrix} \boldsymbol{h^1} \cdot \boldsymbol{x} \\ \boldsymbol{h^2} \cdot \boldsymbol{x} \\ \boldsymbol{h^3} \cdot \boldsymbol{x} \end{pmatrix}$$

Therefore

$$\begin{aligned} \boldsymbol{x'} \times H\boldsymbol{x} &= \begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} \times \begin{pmatrix} \boldsymbol{h^1} \cdot \boldsymbol{x} \\ \boldsymbol{h^2} \cdot \boldsymbol{x} \\ \boldsymbol{h^3} \cdot \boldsymbol{x} \end{pmatrix} \\ &= \begin{pmatrix} x'_2\boldsymbol{h^3} \cdot \boldsymbol{x} - \boldsymbol{h^2} \cdot \boldsymbol{x} \\ \boldsymbol{h^1} \cdot \boldsymbol{x} - x'_1\boldsymbol{h^3} \cdot \boldsymbol{x} \\ x'_1\boldsymbol{h^2} \cdot \boldsymbol{x} - x'_2\boldsymbol{h^1} \cdot \boldsymbol{x} \end{pmatrix} = \boldsymbol{0} \end{aligned} \tag{2.4}$$

Since:

$$\boldsymbol{h^i} \cdot \boldsymbol{x} = \boldsymbol{x}^\top \cdot \boldsymbol{h^{i\top}}$$

expressions in Equation 2.4 may actually be re-written as:

$$\begin{aligned} \boldsymbol{x'} \times H\boldsymbol{x} &= \begin{bmatrix} \boldsymbol{0}^\top & -\boldsymbol{x}^\top & x'_2\boldsymbol{x}^\top \\ \boldsymbol{x}^\top & \boldsymbol{0}^\top & -x'_1\boldsymbol{x}^\top \\ -x'_2\boldsymbol{x}^\top & x'_1\boldsymbol{x}^\top & \boldsymbol{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h^{1\top}} \\ \boldsymbol{h^{2\top}} \\ \boldsymbol{h^{3\top}} \end{pmatrix} \\ &= L\boldsymbol{h} = \boldsymbol{0} \end{aligned} \tag{2.5}$$

Note that here we have:

$$\boldsymbol{h} = (\boldsymbol{h^1}, \boldsymbol{h^2}, \boldsymbol{h^3})$$

is a 9-vector that is constituted by 9 entries of homogeneous matrix H. Furthermore, we notice the fact that if we left multiply matrix L by two elementary matrices in Equation 2.5:

$$L' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x'_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & x'_2 & 1 \end{bmatrix} L$$

we will end up getting

$$\begin{aligned} L'\boldsymbol{h} &= \begin{bmatrix} \boldsymbol{0}^\top & -\boldsymbol{x}^\top & x'_2\boldsymbol{x}^\top \\ \boldsymbol{x}^\top & \boldsymbol{0}^\top & -x'_1\boldsymbol{x}^\top \\ \boldsymbol{0}^\top & \boldsymbol{0}^\top & \boldsymbol{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h^{1\top}} \\ \boldsymbol{h^{2\top}} \\ \boldsymbol{h^{3\top}} \end{pmatrix} \\ &= \boldsymbol{0} \end{aligned} \tag{2.6}$$

As a consequence, we are able to reach the conclusion that Equation 2.5 actually give rise to 2 linearly-independent constraints for solving homogeneous transformation matrix H. At the beginning of Section 2.3 we explained that in order to efficiently calculate H, it is necessary that four corresponding coordinate pairs $(\boldsymbol{x_i}, \boldsymbol{x_i'})$ are given, where $i = 1, 2, 3, 4$. And following the denotations in Section 2.3, we have $\boldsymbol{x_i'} = (x'_{1i}, x'_{2i}, 1)$, where $(x'_{1i}, x'_{2i}),\ i = 1, 2, 3, 4$ are the pixel values in scene image, and $\boldsymbol{x_i} = (x_{1i}, x_{2i}, 1)$, where $(x_{1i}, x_{2i}),\ i = 1, 2, 3, 4$ are the pixel values in texture image. Consequently we will have 8 linear equations. And based on the result in Equation 2.6, the four equation sets for solving texture mapping matrix H are expressed as:

$$\begin{bmatrix} \mathbf{0}^\top & -\boldsymbol{x_i}^\top & x'_{2i}\boldsymbol{x_i}^\top \\ \boldsymbol{x_i}^\top & \mathbf{0}^\top & -x'_{1i}\boldsymbol{x_i}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h^{1\top}} \\ \boldsymbol{h^{2\top}} \\ \boldsymbol{h^{3\top}} \end{pmatrix} = \mathbf{0} \tag{2.7}$$

where $i = 1, 2, 3, 4$. Note that matrix H is defined up to a scalar, and in homogeneous 3D vector (x_1, x_2, x_3), component x_3 is actually the scaling factor, thus without loss of generality, we set $h_9 = 1$, in other words, $\boldsymbol{h^3} = (h_7, h_8, 1)$. Consequently, as denoted in Equation 2.2, we are able to calculate the remaining 8 entries h_j, where $j = 1, 2, ..., 8$ in texture mapping matrix H.

2.3 METHODOLOGY

2.3.1 USER SELECTION OF 4 COORDINATES SETS

In Section 2, we proved that in order to get the texture mapping matrix H, we need four corresponding pixel sets, separately from the texture image and the indoor scene image. Also we noticed in Figure 3.2 that different points A, B, C, D on $\mathbb{P}^2$ may lead to different vanishing points, thus corresponding to different orientations in 3D world. As a result, an arbitrary user selection of the four coordinate sets from images may not lead to a proper texture mapping matrix that corresponds to indoor room geometry layout. Consequently, judging the room layout while selecting points from images is challenging for normal users, which is not what we want for a user-

friendly interface. Hence in our method, we introduce a semi-automatic way to make it easy for users to select the reasonable coordinate sets from scene images and texture images.

The flow chart in Figure 2.4a illustrates an instance of how the user selects four pixels from the scene image. Firstly, the user interacts with our tool through a click-based segmentation framework for indoor scene images [17], which is capable of generating a mask for the components in indoor scene images onto which users would like to re-render new textures. In this case, a floor plane is picked out. Secondly, an embedded geometry layout estimation framework [16] is able to autonomously generate spatial estimation of room layout by presenting users parametrized cubic hypotheses that correspond to 3 primary orthogonal orientations of the room scene, as

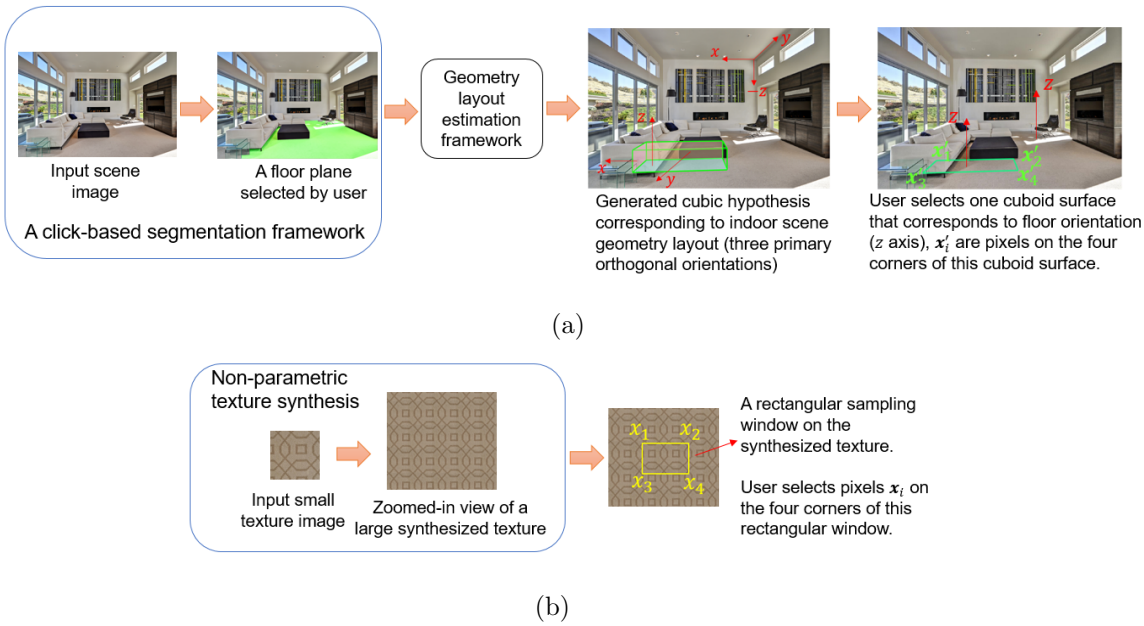


Fig. 2.4. Flow charts that illustrate how the user selects four corresponding pixels from the scene image and the texture image. (a) From left to right: A click-based segmentation framework [17], a geometry layout estimation framework [16], a generated cubic hypothesis from scene image and a user-selected cuboid surface. (b) From left to right: A non-parametric texture synthesis framework [18] and a rectangular window in texture image.

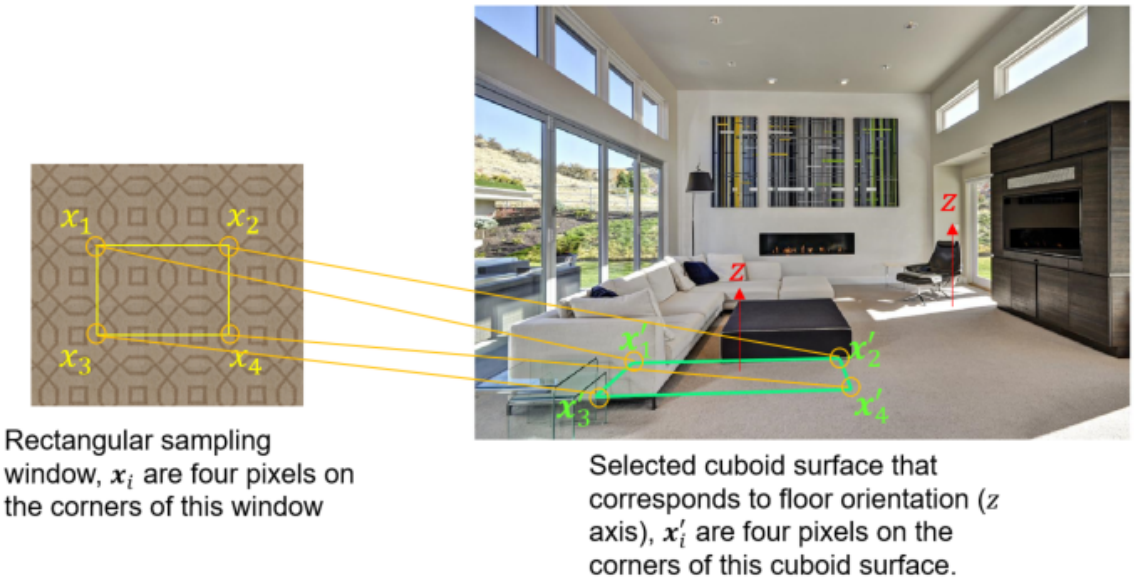


Fig. 2.5. Generating texture mapping matrix H, given four corresponding points in a texture image and a scene image. Through H, pixels at $\boldsymbol{x_i}$ are mapped to pixels at $\boldsymbol{x'_i}$, for $i = 1, 2, 3, 4$.

shown in Figure 2.4. In addition, each of the cuboid surfaces has the same orientation as a rectangular plane in world scene. In next step, the users is able to interact with our tool by selecting one of the surfaces from the generated cubic. Note that in Figure 2.4, the bottom surface of the cubic is selected, since it is corresponding to z axis in room scene configuration, which matches the orientation of the selected floor plane. Once the cubic surface is selected, acquiring the four coordinates from scene image is straight-forward. They are indeed the pixel values on the four corners of the selected cuboid surface.

Next we are going to discuss how the user selects four pixels from the texture image. We already know that four pixels selected from the scene image corresponds to the orientation of a rectangular plane in the world scene. Thus it is quite straightforward that a rectangular window on a texture image can be used to determine four pixels on the texture image. The flow chart in Figure 2.4b illustrates an instance

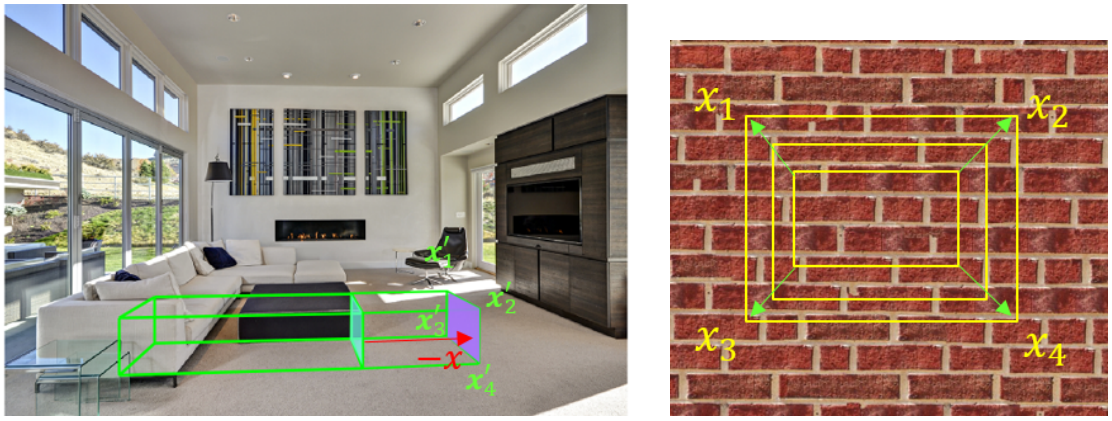
of selecting four pixels from a texture image. In the backend, an embedded texture synthesis framework [18] takes a small texture pattern as input and presents the user with a synthesized large texture pattern. Then the user is able to choose a rectangular window on the synthesized texture. Finally, the pixel values on the four corners of the rectangular window act as the four coordinates from texture image. After all four corresponding coordinate sets

$$(\boldsymbol{x_i},\ \boldsymbol{x_i'}),\ i = 1, 2, 3, 4$$

where $\boldsymbol{x_i}$ is the homogeneous 3-vector representing pixels in the texture image and $\boldsymbol{x_i'}$ is the homogeneous 3-vector representing pixels in the scene image, are picked, by using Equation 2.7, we are able to generate texture mapping matrix H (as shown in Figure 2.5). And then the tool is able to re-render the selected component (as in Figure 2.4a) with the new texture (as in Figure 2.4b).

2.3.2 RE-RENDERING ADJUSTMENTS

In this section, we talk about adjusting re-rendering result. Using the texture mapping matrix generated from Section 3.1 can be useful enough if we only consider the re-rendering result correspondence with the indoor scene (room) geometry layout. Nevertheless, some re-rendering results may be visually over-sized and thus unfitted for some certain scene configurations (as shown in Figure 2.8b). Thus, there is necessity for our tool to propose approaches to refine texture re-rendering results through user-interactions. And Figure 2.6 shows our proposed two methods, which include cubic surface shifting in the scene image and sampling window re-sizing in the texture image. As is seen in Figure 2.6a, the cubic surface before and after shifting has the same orientation (both perpendicular to x axis). Meanwhile, the sampling windows with different sizes obviously have same orientation, since the texture image itself is a plane. Consequently, through the adjustment, the scaling of the re-rendering result is changing, while the orientation of the re-rendered texture, with respect to room geometry layout, is not changing. Hence users can interact with our texture



(a) (b)

Fig. 2.6. Proposed methods for refining texture re-rendering results: (a) Cuboid surface shifting, the green surface is before shifting, and the purple surface is after shifting. The shifting direction, in this case, is along $-x$ axis. (b) Rectangular sampling window re-sizing, three yellow rectangular shapes corresponds to three sampling windows in texture image with different sizes. In this case, the window is expanding.

re-rendering tool in either of the two ways to get a satisfying scaling effect for the re-rendering result.

2.3.3 HIGH-RESOLUTION RENDERING

A critical case that we are especially interested in is to increase the re-rendering quality, while not increasing time consumption too much. Before our method, the contradictory point is that on one hand, for low resolution images, it is fast to generate cubic hypothesis (see Table 2.1), but the rendering quality is poor. The re-rendered texture may come with a bundle of artifacts which are certainly unwelcome by users. On the other hand, for high-resolution images, although it is time-consuming to generate cubic hypothesis (see Table 2.1), the rendering quality is satisfying. In practical cases, users can neither accept that it takes too long to get a result, nor that the rendering quality is of low-quality. Thus we propose a matrix scaling method

that enables high-resolution images to utilize the texture mapping matrices generated from low-resolution images, for the purpose of re-rendering textures on high-resolution images while reduce time consumption for matrices generation.

Let us now think of a pair of two images. One image has a resolution $m \times n$ pixels and the other image has a higher resolution $am \times an$ pixels (a>1). Therefore, one pixel (x_0, y_0) in the low-resolution image corresponds to a pixel (ax_0, ay_0) in the high-resolution image. According to Section 2.1, the 3-vector homogeneous representations for (x_0, y_0) and (ax_0, ay_0) are $\boldsymbol{x'_{Low\_Res}} = (x_0, y_0, 1)$ and $\boldsymbol{x'_{High\_Res}} = (ax_0, ay_0, 1)$, respectively. It is easy to see that the transformation between these 2 points is

$$\begin{pmatrix} ax_0 \\ ay_0 \\ 1 \end{pmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix} \tag{2.8}$$

Now, our method takes the low resolution image as input, and then follows the routine described in Section 3.1 to generate the texture mapping matrix from a texture image to the low-resolution image, which is denoted as H_{Low_Res}. Then, in order to get the texture mapping matrix from the same texture image to the high-resolution image, denoted as H_{High_Res}, we follow the discussion in Section 2.2:

$$\boldsymbol{x'_{Low\_Res}} = H_{Low\_Res}\boldsymbol{x}$$

and

$$\boldsymbol{x'_{High\_Res}} = H_{High\_Res}\boldsymbol{x}$$

where $\boldsymbol{x}$ is 3-vector homogeneous representation for pixels in texture image. By comparing Equation 2.8 and the above two relationships, we have

$$H_{High\_Res}\boldsymbol{x} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} H_{Low\_Res}\boldsymbol{x} \tag{2.9}$$

We know that $\boldsymbol{x}$ actually corresponds to a pixel in texture image, thus it cannot be a null vector. Thus, a straight-forward solution for Equation 2.9 is

$$H_{High\_Res} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} H_{Low\_Res} \tag{2.10}$$

From Equation 2.10 we know that H_{High_Res} can actually be acquired from H_{Low_Res}, just by left-multiplying H_{Low_Res} with a scaling matrix. In this way, we can avoid estimating H_{High_Res} in a time-consuming way.

2.4 EXPERIMENTAL RESULTS

In order to test the efficiency of our texture re-rendering tool, we firstly did texture re-rendering on different room configurations to check whether the newly applied texture corresponds to room spatial layout. And then we tested the efficiency of our tool with respect to re-rendering adjustment. Finally, we compared the results between low-resolution and high-resolution images.

Figure 2.7 shows the indoor scene image re-mixing results for different room configurations. The appearance of re-rendered textures in Figures 2.7a, 2.7b, 2.7c, 2.7e, 2.7g and 2.7i look plausible, because they not only visually correspond to room spatial layout, but also present a harmonious visual effect when their scaling in sizes are compared with the objects nearby. The appearance of re-rendered textures in Figures 2.7d, 2.7f, 2.7h look reasonable in the sense of correspondence with room geometry orientation. However, the results in these figures are basically generated from raw texture mapping matrix with less refinement and user adjustments. Therefore they present us an inharmonious visual effect when their sizes are compared with that of nearby objects. Nevertheless, scaling can be refined and adjusted through our user-interface that is introduced in Section 3.2.

Figure 2.8 actually shows how user interaction affects the scaling of re-rendering results. As the highlighted brick tiles suggest, re-rendering result in Figure 2.8b

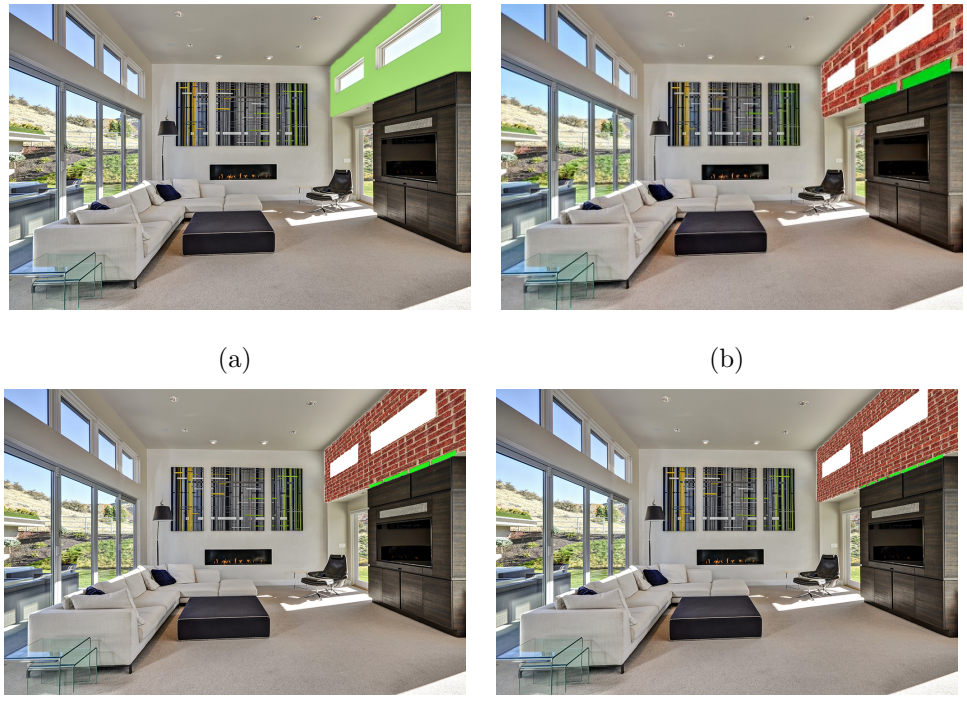
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fig. 2.7. Texture re-rendering and indoor scene image re-mixing results in various room layout configurations. (a) re-rendered floor with bricks texture. (b) re-rendered side wall with bricks texture. (c) re-rendered floor with carpet-like texture. (d) re-rendered floor with stone texture. (e) re-rendered accent wall with bricks texture. (f) re-rendered ceiling with tiles texture. (g) re-rendered accent wall with bricks texture. (h) re-rendered floor with tile texture. (i) re-rendered side wall with bricks texture.

is visually inharmonious because the brick tiles are oversized when compared with cabinet top edges. However, as is observed in Figures 2.8c and 2.8d, with the changes in cuboid surface shifting distance and the sampling window sizes, the scaling of the brick tiles also changes accordingly. We now discuss this in detail. As suggested by highlighted brick tiles, in Figure 2.8c, around 4 brick tiles are aligned with cabinet



(a) (b)

(c) (d)

Fig. 2.8. Illustration of re-rendering adjustment, the adjustments are introduced in Figure 2.6. We denote cuboid surface shifting distance as d in pixel, and sampling window size as win_size in pixels $\times$ pixels. Green blocks are highlighted brick tiles. (a) User-selected component to be re-rendered. (b) Re-rendering result when $d = 100$ and $win\_size = 800 \times 800$. (c) Re-rendering result when $d = 100$ and $win\_size = 1400 \times 1400$. (d) Re-rendering result when $d = 120$ and $win\_size = 1800 \times 1800$.

top edge. And in Figure 2.8d, around 6 brick tiles are aligned with cabinet top edge. So re-rendering results in Figures 2.8c and 2.8d are actually showing brick tile with different scaling in size. In addition, we notice the fact from Figure 2.8 that all the 3 different re-rendering results have the same geometry orientation, which visually corresponds to room layout.

Finally, we discuss the result for our proposed high-resolution re-rendering method. In the process of generating the results in Figure 2.9, we are actually using the same synthesized texture image with resolution 5000×5000 pixels for both low-resolution

scene images and high-resolution scene images. But as is shown in Figure 2.9, the quality of re-rendered texture on the low-resolution image is poor – a lot of artifacts are visually noticeable; while the re-rendering quality on the high-resolution image is plausible. The critical point here for re-rendering on the high-resolution image is that, although results in Figures 2.9b and 2.9c look very similar, the time consumed for generating these two results are quite different. As shown in Table 2.1, in order to generate the texture mapping matrix, time consumed for Figure 2.9b is more than 20 minutes while for Figure 2.9c, it is around 5 seconds. The results indicates that using our proposed matrix scaling method greatly enhances the efficiency of re-rendering on high-resolution images.

2.5 CONCLUSION

In the paper we propose a novel tool for texture re-rendering and indoor scene image re-mixing. The tool is able to autonomously align newly rendered texture with room spatial layout. Also it provides users an interface for adjusting the scaling of

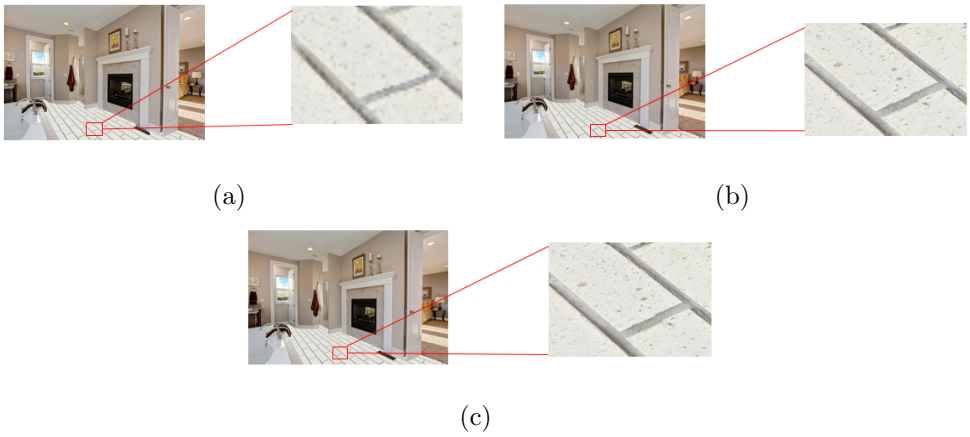


Fig. 2.9. High-resolution texture re-rendering and Low-resolution re-rendering with zoomed-in view. (a) Re-rendering result on 1024×683 pixels scene image. (b) and (c) Re-rendering result on 3861×2574 pixels scene image. (b) and (c) are different in the way of generating texture mapping matrix (see Table 2.1).

Table 2.1.
Table I: Time consumption comparison for different approaches to generate texture mapping matrix, experiment performed in MATLAB®

Image in Figure	**Image size (in pixels × pixels)**	**Approach for generating texture mapping matrix**	Time consumed for generating texture mapping matrix
a	1024 × 683	Geometry layout estimation [16]	~5 secs
b	3861 × 2574	Geometry layout estimation [16]	>20 mins
c	3861 × 2574	Our proposed matrix scaling method	~5 secs

the re-rendered texture in order to match with practical object sizes. In addition, our tool is capable of proving high-quality rendering result efficiently. In the future, we plan to research options to autonomously enable more realistic outcomes: one is to autonomously adjust the scaling of re-rendered texture, the other is to add light and shadow effects.

3. DIGITAL HALFTONING WITH HYBRID SCREEN FOR LASER ELECTROPHOTOGRAPHIC SYSTEMS WITH UNEQUAL RESOLUTION

3.1 INTRODUCTION AND RELATED WORK

The efficient iterative search based algorithm has been producing decent digital halftoning results, but due to the limit of computational resources, this kind of algorithm cannot be integrated into normal laser electrophotographic systems for real-time execution. The screening method for generating digital halftoning result is a more practical way for normal industrial laser printers at a low cost of computational resources, thus it can be executed in real time. Hybrid screen utilizes the search-based method for designing the screening algorithm. According to the textures generated by screen, they can be categorized into clustered-dot screens and dispersed-dot screens. A clustered-dot screen is made up of clustered individual printing pixels, which typically forms a periodic grid-like pattern. The frequency of the grid pattern is referred to as the screen frequency [2]. A number of grey levels can be produced through varying the size of the dot cluster. The dispersed-dot screen does not form cluster, and distinct grey levels are achieved through changing the density of printing pixels (printer addressable dots). The dispersed-dot screen is commonly integrated into the inkjet printer, which is able to easily generate stable and isolated printing pixels. Oppositely, for the laser printer based on electrophotograhic (EP) process technology whose rendering output is unstable, the clustered-dot screen is more robust compared to the dispersed-dot screen. However, there is also limitation about the clustered-dot screen. One of the constrains is that in order to implement a clustered-dot screen in printers, the screen frequency must be lower than the frequency of the printer such as to form clusters. This limits the capacity of such screens for detail resolving. An-

other disadvantage of the clustered-dot screen is that the number of gray levels are insufficient, since the number of gray levels should be no more than the number of printing pixels within a halftone cell. As a result, the output halftone patterns show visible contour artifacts in the region where gray levels are gradually changing due to the sharp gradient between different gray levels. The two disadvantages are actually coupled and there is a trade-off between them. If we increase the screen frequency, the capacity for the screen to render details will increase, but in the mean time the size of the halftone cell must decrease, thus the number of gray levels becomes less. On the other hand, if we increase the number of gray levels, the size of halftone cells must be increased as well, thus the screen frequency decreases and it decrease the screen's capability to render details. In order to overcome this limit, there has been a popular supercell approach [19]. In the supercell approach, the clustered-dot halftone cells that we discussed previously becomes *microcells* that are formed together as the a new halftone cell known as a *supercell* [2]. For supercells, as the absorptance increases, there are multiple dot clusters that are growing within the supercell. Unlike the conventional clustered-dot screen whose multiple cluster dots grow simultaneously, the clusters in the microcells within a supercell can either grow sequentially or a few of them once. Here the dot growing sequence within the microcell is denoted as the turn-on sequence of the microcell. Also, the sequence by which microcell clusters grow is denoted as the *macrocell* growing sequence [2]. The benefit of a supercell is that, the number of gray levels can be increased by adopting both microcell turn on sequence and macrocell turn on sequence. In this way, the gray level number is extended from the length of the microcell dot growing sequence to the product of the length of the microcell dot growing sequence and the length of the macrocell dot growing sequence. In this way, the total number of the gray level in a supercell can be as large as the pixel number in this supercell. Therefore, the advantage of a supercell screen is that we can keep the screen frequency while arbitrarily increase the number of gray levels. In my research of designing hybrid screen, this supercell screen is implemented. In addition, in order to allow even more gray levels, the printer ad-

dressable pixels have the property of partially turned on, thus allowing multiple bits of condition for a single printing pixel. In a supercell, the macrocell growing sequence can be a major impact on the quality of final halftone patterns. And there are several designing rules for macrocell growing sequence. One of them is the commonly-known Bayer Screen [20] when the dots are placed successively in microcells. However, due to the fact that the effective halftoning frequency is the same as the microcell screen frequency, the regular textures that are visible in Bayer Screen is exacerbated [2]. Another logical solution for designing macrocell growing sequence is to replace the Bayer Screen by a stochastic, blue noise screen [21] [22]. Nevertheless, the artifacts similar to that come from Bayer Screen also occur when the blue noise screen is adopted to control the macrocell growing sequence [2]. Another disadvantage about the blue noise screen is that it will create a maze like artifacts for the highlight texture due to the presence of parallel rows of dots [2]. The hybrid screen proposed by Lin and Allebach [3] is another method for designing macrocell growing sequence, which aims to solve the problem caused by the Bayer Screen and the blue noise screen. In the hybrid screen, there is a small region in the microcell defined as *core*. Within the core, pixels has the spatial freedom of moving around either within the core or from core to core, following specific constraints. This spatial freedom allows us to eliminate the maze like artifacts caused by blue noise screens [2]. After the cores are filled, the macrocell growing sequence then becomes a regular periodic, clustered-dot texture. Beyond the level at which all the cores are completely filled, the dot cluster then grows in a manner similar to typical clustered-dot screen, except the fact that the macrocell growing sequence is determined by the first dot turn on sequence in the core. Within the core of the microcells, the dot growing sequence is determined by Direct Binary Search (DBS) algorithm. Besides, DBS also helps determine the macrocell dot growing sequence. In addition, in order to let the designed microcell have the dot-hole complementary symmetry, the shadow core is also defined, which applies the same strategy to the shadow region. So far, our discussion on digital halftoning with hybrid is screen is based on the presumption that the printer only

has the capability of binary output, that is to say, the output condition of the printer addressable pixel is either filled with a colorant dot or not filled with a colorant dot. However, laser electrophotographic printers are commonly designed with a multilevel capacity [2], which is commonly referred to as the pulse width modulation (PWM) ability. With this ability, the multilevel halftone systems are able to generate even more gray levels, with supercell already implemented for binary /printer addressable pixel case.

Notice that our discussion about the digital halftoning so far is based on equal printer resolution on scan direction and process direction. For equal resolution, printer addressable pixel has the same dimension of stretching in both horizontal and vertical direction. Therefore, when applying digital halftoning algorithms, the dimension of the digital image pixel matches that of the actual printer addressable pixel. As a result, digital halftoning algorithm like DBS can be directly applied to the digital image pixels in order to get an illusion that is close to the real printing outcome. However, for printers whose addressable pixel has different dimension of stretching in horizontal and vertical direction, thus causing an unequal resolution of pixels horizontally and vertically, we use the pixels from digital image as the *subpixel* that represents pixel block with identical dimensions in horizontal and vertical direction, and several subpixels are aligned with different numbers horizontally and vertically, forming a printer addressable pixel with unequal resolution. Then the screening and search based halftoning algorithms are applied to the unequal resolution pixel by keeping the binary condition of the subpixels within the pixel identical to each other. In the following sections we will discuss the details for designing hybrid screen for printers with unequal resolution and PWM ability.

3.2 METHODOLOGY

3.2.1 UNEQUAL RESOLUTION PRINTER MODEL

We use $(\mathbf{x}) = (x, y)$ and $[\mathbf{m}] = [m, n]$ for representing continuous tone images and discrete spatial coordinates, respectively. And therefore the units for $(\mathbf{x})$ and $[\mathbf{m}]$ are pixels and inches, respectively. And in our denotation, we let $f[\mathbf{m}]$ be a continuous-tone image and $g[\mathbf{m}]$ be the halftone pattern corresponding to $f[\mathbf{m}]$. Here both $g[\mathbf{m}]$ and $f[\mathbf{m}]$ are expressed in terms of *absorptance level*, where 0 is white (no absorptance) and 1 is black (full absorptance). In this way, $f[\mathbf{m}]$ will take discrete values $a \in \{0, \frac{1}{L-1}, \frac{2}{L-1}, ..., 1\}$ where L is the number of gray levels. As for $g[\mathbf{m}]$, the value is either 0 or 1, since on halftone patterns there can only be black or white pixels. Notice that the *pixel* we discuss here on $g[\mathbf{m}]$ is actually the the subpixel. For 1 bit per printing pixel case, a subpixel in $g[\mathbf{m}]$ represents a printer addressable pixel, while for printers with PWM ability and for printers with unequal resolution, the cluster of subpixels in $g[\mathbf{m}]$ represents an actual printer addressable pixel. Suppose now we have an ideal continuous-tone printer that can produce a perfect rectangle dot with absorptance equal to that of the discrete image pixel value. Let $f[\mathbf{x}]$ be the version of $f[\mathbf{m}]$ that is rendered by the idea printer, then $f[\mathbf{x}]$ can be written as

$$f(x) = \sum_{\mathbf{m}} f(\mathbf{m}) p(x - mX, y - nY) \tag{3.1}$$

where $p[\mathbf{x}] = p(x, y) = rect(x/X, y/Y)$ is the spot profile function of the ideal printer with unequal resolution. X is the distance between addressable dots in vertical direction, and Y is the distance between addressable dots in horizontal direction. Thus the printer vertical resolution is 1/X and horizontal resolution is 1/Y. Likewise, $g[\mathbf{x}]$ and $g[\mathbf{m}]$ have the same relationship

$$g(x) = \sum_{\mathbf{m}} g(\mathbf{m}) p(x - mX, y - nY) \tag{3.2}$$

Note that a laser printer can actually have a high resolution such as $1/X = 400$ dpi (dot per inch) and $1/Y = 600$ dpi (dot per inch).

3.2.2 SUBPIXEL COMPOSITION OF UNEQUAL RESOLUTION PRINTER PIXEL AND HVS MODEL

In our research, subpixels with the same dimension in both horizontal and vertical direction in the digital image represents partial ideal printer addressable dot. And in fact, by tiling different number of subpixels horizontally and vertically, we will have a tiled pixel with different dimension in these two directions. Now suppose that the resolution for the printer is $1/X$ dpi vertically and $1/Y$ dpi horizontally, then the vertical dimension of the pixel is X inches and horizontal dimension of the pixel is Y inches. For printing pixel with binary condition (either fully filled or not filled), the resolution of the subpixel, denoted by $1/Z$ dpi, is chosen such that

$$\frac{1}{Z} = LCM(\frac{1}{X}, \frac{1}{Y}) \tag{3.3}$$

where LCM stands for lowest common multiple. Since $1/Z$ is divisible by both $1/X$ and $1/Y$, the height H_p and width W_p of the subpixel, in unit of number of subpixels, are

$$H_p = \left|\frac{1/X}{1/Z}\right| = \left|\frac{Z}{X}\right|, W_p = \left|\frac{1/Y}{1/Z}\right| = \left|\frac{Z}{Y}\right| \tag{3.4}$$

Fig 3.1 shows the composition of an unequal resolution printer dot with vertical dimension 1/400 inches and horizontal dimension 1/600 inches from tiling 3 by 2 subpixels that are perfect square pixels of $1/1200 \times 1/1200$ inches, in vertical and horizontal direction.

Now, for the continuous-tone image $f[\mathbf{m}]$ and the halftone image $g[\mathbf{m}]$ with resolution $X \times Y$ dpi, which are made up with $1/Z \times 1/Z$ inches subpixels, they can actually be recognized as images with equal resolution Z dpi with the characteristic that all the subpixels within an unequal resolution pixel must have the same binary condition, which means they are simultaneously turned on or off. Based on the discussions above, the HVS (Human Visual System) that is applied on the images with unequal resolution pixels that are tiled from subpixels can also be considered as the one that is applied on the images with equal resolution subpixels. Therefore, there is

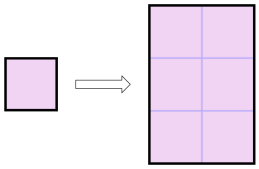


Fig. 3.1. An illustration of the composition for unequal resolution printer addressable pixel from subpixels. The left shows a subpixel which represents the physical $1/1200 \times 1/1200$ inches square block, the right shows the tiled pixel with height $1/400$ inches and width $1/600$ inches.

no need to change the human visual system when running HVS-dependent halftoning algorithms, such as DBS, on unequal resolution images composed by subpixels.

In the research of digital halftoning, the HVS is modeled as a linear, shift-invariant low-pass filter. And the spatial representation for the HVS are denoted as $h(\mathbf{x})$. When we consider that the HVS is applied to an image composed by subpixels where the distance between the subpixels is Z inches, then the perceived image $\tilde{f}(\mathbf{x})$ through the visual filter is obtained from the convolution between rendered image $f(\mathbf{x})$ and $h(\mathbf{x})$

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) * *h(\mathbf{x}) = \sum_{\mathbf{m}} f(\mathbf{m})\tilde{p}(\mathbf{x} - \mathbf{m}Z) \tag{3.5}$$

where $**$ denotes the 2-D convolution and $\tilde{p}(\mathbf{x}) = p(\mathbf{x}) * *h(\mathbf{x})$ is the perceived spot profile function of the ideal printer on the subpixels. Similarly,

$$\tilde{g}(\mathbf{x}) = \sum_{\mathbf{m}} g(\mathbf{m})\tilde{p}(\mathbf{x} - \mathbf{m}Z) \tag{3.6}$$

For a laser printer with unequal resolution $1/X = 400$ dpi and $1/Y = 600$ dpi, by raising equation 3.3, we are able to get the resolution of subpixel $1/Z = 1200$ dpi. The spot profile function $p(\mathbf{x})$ is sufficiently narrow when compared with $h(\mathbf{x})$, and $p(\mathbf{x})$ is also a rectangle function as defined in 4.1.2. Therefore we can make the assumption that $\tilde{p}(\mathbf{x}) = h(\mathbf{x})$ [2]. The HVS model that we use for monochrome halftoning is based on the luminance contrast sensitivity function that is based on Näsänen's model [2]. Kim and Allebach compared Näsänen's model with other three human visual model

and concluded that Näsänen's model give the best overall halftone quality when used together with DBS algorithm [23]. The luminance spatial frequency response $H(\bar{\mathbf{u}})$ for HVS based on Näsänen's model is given [24] by:

$$\bar{H}(\bar{\mathbf{u}}) = a\Gamma^b \exp\left(-\frac{\|\bar{\mathbf{u}}\|}{c\ln\Gamma + d}\right) \tag{3.7}$$

where $(\bar{\mathbf{u}}) = (\bar{u}, \bar{v})$ are the spatial frequency coordinates in the unit of cycles/degree. The parameters for the models are $a = 131.6$, $b = 0.3188$, $c = 0.525$, $d = 3.91$, and Γ is the average luminance of the light reflected from the print in the unit of cd/m^2. Considering the continuous-space Fourier transformation (CSFT) of $\bar{H}(\bar{\mathbf{u}})$, we can get the HVS filter in spatial domain, denoted as $\bar{h}(\bar{\mathbf{x}})$. Notice that here $(\bar{\mathbf{x}}) = (\bar{x}, \bar{y})$, which are degrees subtended at the retina. In order to better represent the HVS in spatial domain, we need convert the unit of cycles/degree in Equation 3.7 to cycles/inch, which is

$$\bar{x} = \frac{180}{\pi}\tan^{-1}\left(\frac{x}{D}\right) \approx \frac{180}{\pi D}x, \text{ for } x \ll D \tag{3.8}$$

By using this transformation, the spatial filter can be written [2] as:

$$\tilde{p}(\mathbf{x}) = h(\mathbf{x}) = \frac{1}{T^2}\bar{h}\left(\frac{\mathbf{x}}{T}\right) = \frac{1}{T^2}\bar{h}(\bar{\mathbf{x}}) = \frac{1}{T^2}\mathcal{F}^{-1}\{\bar{H}(\bar{\mathbf{u}})\} \tag{3.9}$$

where $T = \frac{\pi D}{180}$.

3.2.3 SCREENING AND DOT PROFILE FUNCTION

In this section we will talk about the screening process in using hybrid screen to generate halftone pattern. Screening is the process of comparing each input pixel value from continuous tone image with a spatially varying threshold. If the input pixel value is greater or equal than the threshold, the halftone image pixel is set to black. Otherwise, the pixel is set to white. The thresholds are usually stored in a 2-D array of size $H \times W$, where H and W are the height and the width of the screen, respectively. Through the screening process, a continuous-tone image is rendered to one of the halftone patterns that corresponds to one of the absorptance levels

$a \in \left\{0, \frac{1}{L-1}, \frac{2}{L-1}, ..., 1\right\}$. Note that here the absorptance level are normalized within the range of $(0, 1)$, and the absorptance of 0 is the white color, while the absorptance level of 1 is the black color. The dot profile function $p[\mathbf{m}; a]$ is defined as the family of halftone patterns corresponding to different absorptance level a. In the screening process, if a dot is present at $\mathbf{m}$ at level a_0, then the dot must appear at $\mathbf{m}$ for all the darker level $a > a_0$. That is to say, if $p[\mathbf{m}; a_0] = 1$, then $p[\mathbf{m}; a] = 1$ for all $a > a_0$. The rule is called the *stacking constraint* [2].

The dot-turn on sequence, or the index matrix $d[\mathbf{m}]$ can be acquired though [2]

$$d[\mathbf{m}] = L - 1 - \sum_{i=0}^{L-1} p\left[\mathbf{m}; \frac{i}{L-1}\right], \tag{3.10}$$

This equation suggests that, the greater the threshold for the absorptance level is, the greater the index value is. Also we are able to conclude that the number of elements in the index matrix is equal to the number of total gray levels minus one. From the index matrix, we are able to get the threshold matrix, which is given as

$$t[\mathbf{m}] = \frac{d[\mathbf{m}] + 0.5}{L - 1} \tag{3.11}$$

Note that the values in the threshold matrix are actually absorptance levels within the range of $(0, 1)$. Therefore, in order to generate the halftone pattern of the continuous-tone image from this screen, we firstly have to transform the gray scale values $\bar{f}[\mathbf{m}] \in [0, 255]$ from the continuous-tone image into normalized absorptance levels $f[\mathbf{m}] \in [0, 1]$, which follows

$$f[\mathbf{m}] = 1 - \frac{\bar{f}[\mathbf{m}]}{255} \tag{3.12}$$

And then the halftone image $g[\mathbf{m}]$ after the screening process is given as

$$g[\mathbf{m}] = \begin{cases} 1, \text{ if } f[\mathbf{m}] \geq t[\mathbf{m} \bmod [H, W]] \\ 0, \text{ otherwise} \end{cases} \tag{3.13}$$

3.2.4 DIRECT BINARY SEARCH AND WRAPAROUND EFFECT

DBS is a search based digital halftoning method that aims to generate the halftone pattern that has the minimal error between the perceived continuous-tone image and

the halftone image [2], which are the original images filtered by HVS filter defined in section 3.2.2.

Now we consider a trial change to a current subpixel with dimension $Z \times Z$ by either modifying the state of current subpixel (toggle on or off) or switch the states of the current subpixel with another neighboring pixel (swap). If any of these changes decrease the total error, then the change that decrease the total error the most is accepted and the halftone pattern is updated. If none of these changes decrease the total error, then the change that increase the total error the lease is accepted and the halftone pattern is updated. The perceived error $\tilde{e}(\mathbf{x})$ between the continuous-tone image and the halftone image is given [2] by

$$\tilde{e}(\mathbf{x}) = \tilde{g}(\mathbf{x}) - \tilde{f}(\mathbf{x}) = \sum_{\mathbf{m}} e[\mathbf{m}]\tilde{p}(\mathbf{x} - \mathbf{m}Z) \tag{3.14}$$

where $e[\mathbf{m}] = g[\mathbf{m}] - f[\mathbf{m}]$ is the error image before the perception of HVS. Then the mean squared error after the perception of HVS is given as

$$E = \int |\tilde{e}(\mathbf{x})|^2 \mathrm{d}\mathbf{x} = \sum_{\mathbf{m}} e[\mathbf{m}] c_{\tilde{p}\tilde{e}}[\mathbf{m}] \tag{3.15}$$

where $c_{\tilde{p}\tilde{e}}[\mathbf{m}] = e[\mathbf{m}] * * c_{\tilde{p}\tilde{p}}[\mathbf{m}]$. According to Section 3.2.2, $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$ is acquired by evaluating the autocorrelation function of HVS

$$c_{\tilde{p}\tilde{p}}[\mathbf{x}] = \int \tilde{p}(\mathbf{s})\tilde{p}(\mathbf{s} + \mathbf{x})\mathrm{d}\mathbf{s} \tag{3.16}$$

at at points on the printer lattice of subpixels , which is indeed $c_{\tilde{p}\tilde{p}}[\mathbf{m}] = c_{\tilde{p}\tilde{p}}[\mathbf{m}Z]$.

In the case of single toggling of a subpixel from $g[\mathbf{m}]$, the trial halftone image $g'[\mathbf{m}]$ after the toggling at subpixel located at $\mathbf{m}_0$ are expressed as

$$g'[\mathbf{m}] = g[\mathbf{m}] + a_0\delta[\mathbf{m} - \mathbf{m}_0] \tag{3.17}$$

where a_0 is the absorptance change at $\mathbf{m}_0$. Therefore $a_0 = 1$ if $g[\mathbf{m}_0] = 0$ and $a_0 = -1$ if $g[\mathbf{m}_0] = 1$. The trial error change is given [2] by

$$\Delta E = a_0^2 c_{\tilde{p}\tilde{p}}[\mathbf{0}] + 2a_0 c_{\tilde{p}\tilde{e}}[\mathbf{m}_0]. \tag{3.18}$$

If there are any trial changes that lead to $\Delta E < 0$, then among the trial changes considered, the toggling operation that decrease the total error the most is accepted. However, in some occasions, there are none of the trial changes that decrease the total error, thus $\Delta E \geq 0$ for all the trial changes. In this case, the toggling operation that increase the total error the lease is accepted. Consider ΔE as a function of $\mathbf{m}$, by which $\Delta E[\mathbf{m}]$ indicates the trial error change when the trial toggling is operated at $\mathbf{m}$. Then for both of the circumstances of ΔE discussed above, A very general expression for the chosen toggling operation at $\mathbf{m}$ is such that

$$\mathbf{m} = \arg\min_{\mathbf{m}} \Delta E[\mathbf{m}] \tag{3.19}$$

Then after the toggling change is accepted, if the continuous tone image $f[\mathbf{m}]$ to which the halftone image $g[\mathbf{m}]$ corresponds is *not* changing, then we can update $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ as

$$c'_{\tilde{p}\tilde{e}}[\mathbf{m}] = c_{\tilde{p}\tilde{e}}[\mathbf{m}] + a_0 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_0]. \tag{3.20}$$

In case of a swapping operation between two subpixels, the values of the current subpixel and the neighboring pixels are switched. If the subpixel at $\mathbf{m}_0$ and $\mathbf{m}_1$ are being swapped, then the halftone image after this trial swap is

$$g'[\mathbf{m}] = g[\mathbf{m}] + a_0\delta[\mathbf{m} - \mathbf{m}_0] + a_1\delta[\mathbf{m} - \mathbf{m}_1], \tag{3.21}$$

where a_0 and a_1 indicate the absorptance level changes at subpixels $\mathbf{m}_0$ and $\mathbf{m}_1$, respectively. Hence $a_0 = -a_1$. The value of a_0 can be acquired through evaluating Equation 3.17. The trial error change is given [2] as

$$\Delta E = (a_0^2 + a_1^2)c_{\tilde{p}\tilde{p}}[\mathbf{0}] + 2a_0 c_{\tilde{p}\tilde{e}}[\mathbf{m}_0] + 2a_1 c_{\tilde{p}\tilde{e}}[\mathbf{m}_1] + 2a_0 a_1 c_{\tilde{p}\tilde{p}}[\mathbf{m}_0 - \mathbf{m}_1]. \tag{3.22}$$

If $\Delta E < 0$ then the trial swap that decrease the error the most is accepted, and $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ is updates [2] as

$$c'_{\tilde{p}\tilde{e}}[\mathbf{m}] = c_{\tilde{p}\tilde{e}}[\mathbf{m}] + a_0 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_0] + a_1 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_1]. \tag{3.23}$$

Now we discuss the wraparound effect when using DBS for the screen design. Suppose the screen size is $H \times W$, indicating that the size for both $f[\mathbf{m}]$ and $g[\mathbf{m}]$ is

$H \times W$. Then $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ has the same size of $f[\mathbf{m}]$ and $g[\mathbf{m}]$, which is also $H \times W$. By considering the wrap around effect, the halftone pattern looks to the DBS algorithm as though it repeats with period $H \times W$. Note that in Equation 3.20 through 3.23, in order to evaluate ΔE and update $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$, we need to look up values from the autocorrelation of the PSF (point spread function) of the HVS, denoted $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$. In addition, the wraparound effect of $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ also needs to be considered when subpixel are out of the boundary of $H \times W$. Let's now assume that PSF for the human visual system model has a support of $P \times P$, then $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$ is of size $2P-1 \times 2P-1$ for a perceived spot $\tilde{p}$. When wrap around effect is taken in account, by noticing the denotation that $\mathbf{m} = [m, n]$, Equation 3.20 is re-written as

$$c'_{\tilde{p}\tilde{e}}[m \bmod H, n \bmod W] = c_{\tilde{p}\tilde{e}}[m \bmod H, n \bmod W] + a_0 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_0]. \tag{3.24}$$

And Equation 3.23 is re-written as

$$c'_{\tilde{p}\tilde{e}}[m \bmod H, n \bmod W] = c_{\tilde{p}\tilde{e}}[m \bmod H, n \bmod W] + a_0 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_0] + a_1 c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_1]. \tag{3.25}$$

Now we consider the wraparound effect in looking up the values of $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$ in Equation 3.22, which occurs when $\mathbf{m}_0 - \mathbf{m}_1$ are out of the cover range of the PSF of HVS. By taking the denotation that $[\mathbf{m}_0 - \mathbf{m}_1] = [m_0 - m_1, n_0 - n_1]$, the equation is re-written as

$$\Delta E = (a_0^2 + a_1^2) c_{\tilde{p}\tilde{p}}[\mathbf{0}] + 2a_0 c_{\tilde{p}\tilde{e}}[\mathbf{m}_0] + 2a_1 c_{\tilde{p}\tilde{e}}[\mathbf{m}_1] + 2a_0 a_1 c_{\tilde{p}\tilde{p}}[|m_0 - m_1| \bmod P, |n_0 - n_1| \bmod P]. \tag{3.26}$$

The above equations suggest that wraparound occurs for $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ and $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ when we calculate ΔE and update $c_{\tilde{p}\tilde{e}}$. Figure 3.2 shows the wraparound effect that happens when we look up values in $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$ and update the values in $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$. And the figure also illustrates the outcomes of the modulo operator in Equation 3.24 through 3.26, which functions as giving DBS the illusion that the $H \times W$ halftone pattern are repeated with a period of $H \times W$.

3.2.5 UNEQUAL RESOLUTION DBS

Unequal resolution DBS is operated on the unequal resolution pixel which is actually a block of subpixels. Referring to the subpixel composition that is described in section 3.2.2, DBS on the pixels with unequal resolution $1/X \times 1/Y$ dpi can indeed be recognized as DBS on the subpixels with equal resolution $1/Z \times 1/Z$ dpi, where Z is given by Equation 3.3. Given this presumption, for a trial toggle in the operation of the unequal resolution DBS, all the subpixels within the unequal resolution pixel is simultaneously turned on or off. And for a trial swap in the operation of the unequal resolution DBS, all the subpixels within one unequal resolution pixels are simultaneously switched condition with all the subpixels within the other unequal resolution pixels. In other words, all the subpixels with dimension $Z \times Z$ inches that compose the unequal resolution pixel with dimension $X \times Y$ gather as a single unit, as shown in Figure 3.3. Therefore, we can reach the fact that the unequal resolution DBS is indeed a special case of the equal resolution DBS with certain constraints. Kacker *et al.* has developed a version of the DBS algorithm that is suitable for the case of multiple subpixel changes [25]. Lee and Allebach [2] also proposed the equations of DBS toggle and swap for $(q-1) \times 1$ subpixel blocks. Based on these algorithm. Now we presume that the unequal resolution pixel has $m \times n$ subpixels. Then we are able to write the DBS toggle and swap equations for the unequal resolution pixel, as well as the $m \times n$ subpixel block. Suppose that we toggle the unequal resolution pixel at m_0 or swap the unequal resolution pixel at $\mathbf{m}_0$ and $\mathbf{m}_1$, which means that we are also toggling or swapping the subpixel blocks. In the following discussions, we denote the subpixels in the unequal resolution pixel at $\mathbf{m}_0$ as $\mathbf{m}_{0ij}$, where $i \in \{0, 1, ..., m-1\}$ and $j \in \{0, 1, ..., n-1\}$. Likewise, we denote the subpixels in the unequal resolution pixel at $\mathbf{m}_1$ as $\mathbf{m}_{1kl}$, where $k \in \{0, 1, ..., m-1\}$ and $l \in \{0, 1, ..., n-1\}$. Then the equations for $g[\mathbf{m}]$, ΔE and $c'_{\tilde{p}\tilde{e}}[\mathbf{m}]$ in Equations 3.17 - 3.18 and 3.20 - 3.23 is re-written as

$$g'[\mathbf{m}] = g[\mathbf{m}] + a_0 \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \delta[\mathbf{m} - \mathbf{m}_{0ij}] + a_1 \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} \delta[\mathbf{m} - \mathbf{m}_{1kl}] \tag{3.27}$$

$$\Delta E = 2a_0 \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} c_{\tilde{p}\tilde{e}}[\mathbf{m}_{0ij}] + 2a_1 \sum_{k=0}^{m-1}\sum_{l=0}^{n-1} c_{\tilde{p}\tilde{e}}[\mathbf{m}_{1ij}] + a_0 a_1 \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{m-1}\sum_{l=0}^{n-1} c_{\tilde{p}\tilde{p}}[\mathbf{m}_{0ij} - \mathbf{m}_{1kl}]. \tag{3.28}$$

$$c'_{\tilde{p}\tilde{e}}[\mathbf{m}] = c_{\tilde{p}\tilde{e}}[\mathbf{m}] + a_0 \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_{0ij}] + a_1 \sum_{k=0}^{m-1}\sum_{l=0}^{n-1} c_{\tilde{p}\tilde{p}}[\mathbf{m} - \mathbf{m}_{1kl}], \tag{3.29}$$

where a_0 and a_1 are the absorptance level changes at pixel $\mathbf{m}_0$ and $\mathbf{m}_1$, respectively. As a result, $a_0 = 1$ and $a_1 = 0$ in a toggle operation, and $a_1 = -a_0$ in a swap operation. A special instance for the toggle and swap operation on the pixel that is made up of 3×2 subpixels are shown in Figure 3.3.

3.3 BILEVEL HYBRID SCREEN DESIGN

In this section we will discuss the designing for hybrid screen integrating DBS approach for electrophotographic printers that does not have pixel modulation technology, meaning that each of the printer addressable pixel is of bilevel output (either fully toggled on or off).

3.3.1 TILING VECTORS, PERIODICITY MATRIX AND SCREEN ANGLE

Generally, a periodic screen can be associated with two vectors $\mathbf{z} = [z_i, z_j]$ and $w = [w_i, w_j]$, as shown in Figure 3.4. These vectors are defined as the screen tile vectors [2], which are restricted to be integer pairs. By shifting and copying the vector set using $n_1\mathbf{z} + n_2\mathbf{w},\ (n_1, n_2) \in \mathbb{R}^2$, we are able to cover the whole spatial domain. In addition, in order to span the whole space with the tile vectors, the vectors must be linear independent. Figure 3.4 actually shows an example of the tile vectors $\mathbf{z} = [2, 3]$ and $\mathbf{w} = [2, -3]$. The *periodicity matrix* is defined as $\mathbf{N} = [\mathbf{z}^T|\mathbf{w}^T]$. Another import concept for hybrid screen design is the *screen angle*, which is defined [2] as the angle between a tile vector and the $+j$ axis, which is the horizontal axis shown in Figure 3.4. It is worth mentioning that here we are considering the unequal resolution printer dots with dimension $X \times Y$ inches. Therefore, in order to calculate the screen angle

for unequal printing resolution, we have to consider the stretch caused by the non-uniform dimension. The stretching factor γ is given as the ratio between vertical resolution and horizontal resolution, which is indeed

$$\gamma = \frac{1/X}{1/Y} = Y/X, \tag{3.30}$$

and then screen angle θ is given as

$$\theta = \arctan(\gamma |w_i|/|w_j|) \tag{3.31}$$

For an electrophotographic system with unequal resolution 400 dpi $\times$ 600 dpi, according to the equations above, we can get $\gamma = 600/400 = 3/2$. Thus for the tile vectors $\mathbf{z} = [2, 3]$ and $\mathbf{w} = [2, -3]$ as shown in Figure 3.4(a), the screen angle is $\theta = \arctan(2/3 * 3/2) = 45^{\circ}$.

3.3.2 MICROCELL AND BSB

Once the screen tile vectors are set and the basic parameters are determined, we define [2] the block that roughly matches the parallelogram formed by the tile vector as the *microcell*. Figure 3.4(a) shows a microcell with a shape of cross, which is formed by the tile vector $\mathbf{z} = [2, 3]$ and $\mathbf{w} = [2, -3]$. The number of pixels, denoted as N_m in the microcell is equal to the area of the parallelogram formed by the tile vactors, and is given [2] as

$$N_m = |\det \mathbf{N}| = |z_i w_j - z_j w_i| \tag{3.32}$$

The number of output gray levels per microcell is $N_m + 1$. For the microcell in Figure 3.4(a), $N_m = |2 \cdot (-3) - 3 \cdot 2| = 12$.

Since the microcell shape is not rectangular, when they are tiled together, the boundary is also not rectangular. Nevertheless, it will be convenient in most screening implementations to store the screen in the form of a 2-D array. Hence we use the equivalent rectangular screen from the non-rectangular screen using the method introduced in [2]. The *basic screen block* (BSB) is defined as the smallest rectangle

screen that can be tiled in the vertical and horizontal directions. Figure 3.4(b) shows a BSB that is formed by tiling two microcells. The height H_B and the width W_B of the BSB is given [2] as

$$H_B = \left| \frac{N_m}{\gcd(z_j, w_j)} \right|, \; W_B = \left| \frac{N_m}{\gcd(z_i, w_i)} \right|, \tag{3.33}$$

where gcd stands for greatest common divisor. For the vectors **z** and **w** given in Figure 3.4(a), $H_B = 4$ and $W_B = 6$, thus a 4×6 BSB is formed.

3.3.3 SUPERCELL AND CORES

A *supercell* is a block created by grouping the microcells together. Given the BSB height H_B and width W_B, the screen height and width can be obtained [2] by

$$H = M_H H_B, \; W = M_W W_B, \tag{3.34}$$

where M_H and M_W are the arbitrarily chosen integer multiplication factors in the horizontal and vertical direction. In order to hide repeating patterns in the halftone output, H and W are generally set to be more than a hundred levels [2].

Next we will discuss the creation for the highlight core and shadow core for stochastic-dot texture. The *core* is the region where the original cluster growing sequence given by Equation 3.10 is ignored and chosen by DBS instead. The core region is very flexible, and in our design, we incorporated a 2×2 square core design as shown in Figure 3.5, which will be suitable for growing a conventional round dot-cluster. The region outside of the core area is called the *midtone*, whose growing sequence can follow a conventional clustered dot growing order.

3.3.4 DOT PROFILE FUNCTION GENERATION FOR HIGHLIGHTS AND SHADOWS

As described by the sections before, the highlight and shadow textures are determined by DBS. Before applying DBS, we firstly have to decide the desired number of

levels L_d. Commonly the input image pixel has 256 levels that range from $[0, 1, ...255]$. However, in order to correct for the nonlinear relationship between the absorptance of the ideal halftoning process and the absortance that is measured from the printed page [2], L_d is often set to be more than a thousand levels. And then during the calibration process, a subset of 256 levels is selected from these levels [2]. Now we denote L_M as the length of the macrocell growing sequence as described in Section 4.1 $\{0, 1, ..., L_M - 1\}$. It is actually the same as the level where each microcell in the supercell contains a single dot. Hence the total number of output levels is given as

$$L = L_M N_m + 1, \tag{3.35}$$

where $L \geq L_d$.

In order to generate the index matrix $d[\mathbf{m}]$ as described in Equation 3.10, we generate each level of the dot profile function $p[\mathbf{m}; \frac{i}{L-1}]$ in sequential order. That is to say, in order to create $p[\mathbf{m}; \frac{i}{L-1}]$, we either start from $p[\mathbf{m}; \frac{i-1}{L-1}]$ and add dots to it or start from $p[\mathbf{m}; \frac{i+1}{L-1}]$ and remove dots to it. In this way the stack constraint is guaranteed.

Now suppose the highlight core consists of N_h pixels. Generating the dot profile function in the highlight core region defines $p[\mathbf{m}; \frac{i}{L-1}],\ i = 1, ...N_h L_M$, recalling that L_M is the length of the macrocell turn on sequence. The designing for the first L_M levels, which corresponds placing the first dot in every microcell, is called the *FM seeding* [2]. The designing for the remaining levels $p[\mathbf{m}; \frac{i}{L-1}],\ i = L_M + 1, L_M + 2...N_h L_M$, which corresponds to placing dots sequentially in the microcell until the core is fully filled, is called the *AM growing* [2].

In order to implement FM seeding and AM growing, we use a constrained DBS algorithm to design the dot profile function in the highlights. This DBS algorithm has the restriction of only being able to place dots inside the core region. In addition, the number of dots in each core is also restricted so that dots are evenly distributed all across the screen. Specifically, for $p[\mathbf{m}; \frac{i}{L-1}]$, the maximum number of dots allowed in each core is $\lceil i/L_M \rceil$, which means that when $i \in \{1, 2..., L_M\}$, each core has maximum one dot, and when $i \in \{L_M + 1, ..., 2 * L_M\}$, each core has maximum two dots, until

when $i \in \{(N_h - 1)L_M + 1, ..., N_h L_M\}$, each core has maximum N_h dots, and finally each core has N_h dots, meaning that core region are fully filled.

Generating a level of dot profile function using the constrained DBS is composed of two parts:

(i) Constrained DBS swap for determining $p[\mathbf{m}; \frac{L_M/2}{L-1}]$: In this step, we firstly generate a random halftone pattern corresponding to level $\frac{L_M/2}{L-1}$, where half of the cores have one dot and half of the other cores are empty. We use the unequal resolution DBS described in section 3.2.5 to calculate ΔE and update $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$. For a trial swap, each of the pixel (dot) within the core is able to either swap with the neighboring pixels within the same core or swap with the pixels located at empty cores, and then we calculate ΔE according to Equation 3.29. If any $\Delta E < 0$, then the trial swap that decrease the error the most is accepted; and $g[\mathbf{m}]$ and $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$ are updated according to Equation 3.27 and 3.28. The algorithm stops if $|\Delta E|$ becomes smaller than a threshold and/or the number of iteration exceeds the maximum number of iterations allowed.

(ii) Constrained DBS toggle for generating $p[\mathbf{m}; \frac{i}{L-1}], i \in \{1, 2, ..., L_M/2 - 1\}$ and $p[\mathbf{m}; \frac{i}{L-1}], i \in \{L_M/2 + 1, L_M/2 + 2, ..., N_h L_M\}$. The candidate toggle position must satisfy the constrains that we have discussed in FM seeding and AM growing. And the best trial toggle position is selected by Equation 3.19.

After generating all levels of the dot profile function in the highlight, we now have the index matrix for the highlight core region, written [2] as

$$d^h[\mathbf{m}] = \begin{cases} N_h L_M - \sum_{i=1}^{N_h L_M} p\left[\mathbf{m}; \frac{i}{L-1}\right] & , \text{ if } \mathbf{m} \in \Omega_h \\ 0 & , \text{ otherwise} \end{cases}, \tag{3.36}$$

where Ω_h denotes the highlight core region.

The shadow core is actually designed by following a similar procedure. The difference here is that the dots become holes. Hence we are still able to follow a similar

designing routing to the one described for highlight core. In spite of this, Lee and Allebach proposed an alternative approach [2] for creating the dot profile function for shadow core, which is based on inverting and offsetting the dot profile function of highlight core. Suppose the offset from the highlight core to the shadow core is $[m - m_c, n - n_c]$. The the dot profile function is written as

$$d^s[\mathbf{m}] = \begin{cases} L - 1 - d^h[(m - m_c) \bmod H, (n - n_c) \bmod W] & , \text{ if } \mathbf{m} \in \Omega_s \\ 0 & , \text{ otherwise} \end{cases}, \quad (3.37)$$

3.3.5 SCREEN GENERATION

In this section we will discuss the generation of midtones, thus determining the dot profile function $p[\mathbf{m}; \frac{i}{L-1}], i \in \{N_h L_M + 1, N_h L_M + 2, ..., (N_m - N_s)L_M\}$, where N_h is the number of pixels in the highlight core and N_s is the number of pixels in the shadow core. For bilevel hybrid screen design, we do not use DBS to design dot growing sequence. Instead, we use the first dot turn-on sequence that is determined in the FM seeding process as the macrocell turn-on sequence. In this way, in the process of generating midtones, we have two sets of growing sequence: the first is the macrocell turn-on sequence $d_M[\mathbf{m}] \in \{0, 1, ..., L_M - 1\}$, and the second is the microcell growing index $d_m[\mathbf{m}] \in \{N_h, N_h + 1..., N_m - N_s - 1\}$. We firstly turn on the dots located at microcell index $d_m[\mathbf{m}] \in \{N_h\}$ following the macrocell index, then the dots located at microcell index $d_m[\mathbf{m}] \in \{N_h + 1\}$ are turned on following the, macrocell index. Sequentially, all the dots in the midtones are turned on. Therefore, we have the index matrix for the midtone region as [2]

$$d^m[\mathbf{m}] = \begin{cases} L_M d_m[\mathbf{m}] + d_M[\mathbf{m}] & , \text{ if } \mathbf{m} \in \Omega_m \\ 0 & , \text{ otherwise} \end{cases}, \quad (3.38)$$

where Ω_m is the midtine region. Finally, the overall index matrix is generated [2] by

$$d[\mathbf{m}] = d^h[\mathbf{m}] + d^m[\mathbf{m}] + d^s[\mathbf{m}] \quad (3.39)$$

And the screen or the threshold matrix is generated using Equation 3.11.

3.4 MULTILEVEL HYBRID SCREEN DESIGN

In this sections we will discuss the designing for the hybrid screen levels for pixels with multilevel outputs. As discussed in Section 4.1, it is becoming increasingly common for laser eclectrophotographic printers to support multilevel outputs in order to improve image quality by adopting pixel modulation technique. And in the screening process , multilevel rendering is achieved by comparing the input image pixel value to the thresholds of multiple screens so as to determine the output.

3.4.1 PRELIMINARIES

Figure 3.6 shows a pixel 4 output levels, for such kind of pixel, there are three thresholds that correspond to the different rendering output. In general, if a pixel has q possible output levels, then the number of screens needed to produce q output levels per pixel is $q-1$ [2]. According to Lee and Allebach, the multilevel screens $t_i[\mathbf{m}]$ are given by

$$t_i[\mathbf{m}] = \frac{d_i[\mathbf{m}] + 0.5}{L' - 1}, i = 0, 1, ..., q-2 \tag{3.40}$$

where $d_i[\mathbf{m}], i = 0, 1, ...q-2$ be the multilevel screen index matrices and L' be the number of output levels, which is given [2] as $L' = (L-1)(q-1)+1$, and the halftoning operation corresponding to multiple thresholds are given [2] as

$$g[\mathbf{m}] = \begin{cases} 0, & \text{if } 0 \le f[\mathbf{m}] < t_0[\mathbf{m}] \\ \dfrac{1}{q-1}, & \text{if } t_{i-1}[\mathbf{m}] \le f[\mathbf{m}] < t_i[\mathbf{m}] \\ 1, & \text{if } f[\mathbf{m}] \ge t_{q-2}[\mathbf{m}] \end{cases}, \tag{3.41}$$

Each output pixel multitone level is determined by two parameters: partial dot width and *justification*. Justification is the offset of the partial dot from the native printer

resolution grid, which is used to determine the rule for adding partial dots. Figure 3.6(b) shows an example of four justification modes for a pixel with 4 output levels: left, right, center and split. The justification mode is determined after screening. One method [2] of determining the justification for current pixel is based on the information of the partial-dot width of the left and right pixels. In our method, the determination of the justification for current pixel is based on the rule that the newly formed pattern should be *compact* and its *centroid* must shift from the previous one to the minimum extension, thus forming a symmetric pattern. A simple instance for the compact rule is that, if one of the neighboring pixels is a full dot while the the pixel on the other side is not, then the current pixel is justified toward the full dot, or in other words, the newly added partial dot is adjacent to the full dot. A good illustration of the centroid rule and the compact rule is shown in Figure 3.7. As we can see, the halftone pattern in Figure 3.7(b) has a greater centroid shift than the pattern in Figure 3.7(a); and the halftone halftone pattern in Figure 3.7(d) is less compact than that in Figure 3.7(c). Figures 3.8(a) and (b) show actual halftone patterns generated with and without the centroid rule. Figure 3.8 (c) and (d) are the Gaussian-filtered version of the halftone patterns in Figures 3.8 (a) and (b), respectively. The Gaussian filterer is used as a rough approximation of the visual outcome for the actual printing results. We are able to notice that there are maze-like artifacts in Figure 3.8 (c) but in Figure 3.8 (d) these artifacts are greatly reduced.

3.4.2 DOT PROFILE FUNCTION GENERATION FOR THE MULTI-LEVEL PIXEL

As discussed in Section 3.4.1, in order to generate a screen from the pixels with multilevel output, we have to follow the compact rule and the centroid rule. In addition, for the pixel with multilevel output as shown in Figure 3.6(a), the three partial dots, or the three *slivers* on the left, right and in the middle can be treated as unequal resolution pixels, and thus we can apply the unequal resolutio DBS algorithm

described in Section 3.2.5 to the slivers such as to determine where they should be added when designing the dot profile function. However, when we let DBS recognize these slivers as independent unequal resolution pixel that are freely to be toggled and swapped, it is very likely that the generated halftone is not following the compact rule, as shown in Figure 3.9(a). As a result, we need to run a constrained DBS algorithm to generate the dot profile function for multilevel cases. Unlike the bilevel hybrid screen design where we let DBS generate highlight and shadow patterns and let the midtone growth follow the macrocell sequence, for the multilevel cases, the dot growing sequence is all determined by DBS that is restricted to place new partial dots at places where the design rules described in Section 3.4.1 satisfy. In order to create $p[\mathbf{m}; \frac{i}{L'-1}]$, if we start from $p[\mathbf{m}; \frac{i+1}{L'-1}]$, then we just remove a sliver using DBS, which is similar to what we do in Section 3.3.4. If we start from $p[\mathbf{m}; \frac{i-1}{L'-1}]$, then the newly added sliver must be located at where the compact rule and the centroid rule, as shown in Figure 3.7, are satisfied.

Now suppose there are $L-1$ full dots in the screen and the highlight has N_h full dots that has $q-1$ slivers each. Then generating the dot profile function defines $p[\mathbf{m}; \frac{i}{L'-1}]$ where $L' = (L-1)(p-1)+1$ and $i \in \{1, ..., N_h(p-1)L_M\}$, recalling that L_M is the length of macrocell turn on sequence. Notice that the number of slivers in each core is also restricted so that they are evenly distributed all across the screen. Specially, for $p[\mathbf{m}; \frac{i}{L'-1}]$, the maximum number of slivers allowed for each core is $\lceil i/L_M \rceil$. For instance, when $i \in \{(N_h-1)(p-1)L_M+1, ..., N_h(p-1)L_M\}$, each core has a maximum of $N_h(p-1)$ slivers, and finally all the cores have $N_h(p-1)$ slivers, which means that the$\lceil i/L_M \rceil$ core region are fully filled. In conclusion, presuming that the shadow core has N_s full dots , then generating a level of dot profile function for multilevel pixels using constrained DBS is described as:

(i) Constrained DBS swap for determining $p[\mathbf{m}; \frac{L_M/2}{L'-1}]$, where L_M is the length of macrocell turn on sequence, or the number of microcells in the supercell. We follow a similar routine to the one described in Section 3.3.4. The only difference

is that now we use the unequal resolution DBS described in Section 3.2.5 to swap slivers that are composed by subpixels.

(ii) constrained DBS toggle for generating $p[\mathbf{m}; \frac{i}{L'-1}], i \in \{1, 2, ..., L_M/2 - 1\}$ and $p[\mathbf{m}; \frac{i}{L'-1}], i \in \{L_M/2 + 1, ..., (L - 1 - N_s)(p - 1)L_M\}$. It is worthwhile to be noticed that here we are using DBS toggle to determine midtone dot profile function $p[\mathbf{m}; \frac{i}{L'-1}], i \in \{N_h(p - 1)L_M + 1, ..., (L - 1 - N_s)(p - 1)L_M\}$. The candidate toggle positions are selected with the compact and the centroid rule.

The shadow core dot profile function $p[\mathbf{m}; \frac{i}{L'-1}], i \in \{(L-1-N_s)(p-1)L_M+1, ..., (L-1)(p - 1)L_M\}$ is generated using the same shift-based method described in Section 3.3.4. After we generate $p[\mathbf{m}; \frac{i}{L'-1}], i \in \{1, 2, ..., (L - 1)(p - 1)L_M\}$, the turn on sequence for the slivers is determined by

$$d[\mathbf{m}] = L' - 1 - \sum_{i=0}^{L'-1} p\left[\mathbf{m}; \frac{i}{L' - 1}\right], \tag{3.42}$$

Notice that here we are getting the sliver index matrix. Suppose that the screen size is $H \times W$, then sliver index matrix is of size $H \times (p - 1)W$, recalling that p is the number of output levels per dot. In order to generate multilevel screen index matrices $d_i[\mathbf{m}]$ with size $H \times W$, where $i = 0, 1, ...q - 2$, we have to place the entries in every connected non-overlapping $1 \times (p - 1)$ unit of $d[\mathbf{m}]$ in ascending order. And then the entries of $d_i[\mathbf{m}]$ are the ith elements in each of the $1 \times (p - 1)$ units, where $i \in \{0, 1, ..., p-2\}$. Given $d_i[\mathbf{m}]$, the multilevel screen is determined by Equation 3.40.

3.5 EXPERIMENTAL RESULTS

In this section, we will talk about the experimental result for the hybrid screen design using unequal resolution DBS. Firstly I will show the results of the unequal resolution DBS. Secondly I will show the results of bilevel unequal resolution hybrid screen design and finally I will show the results of multilevel unequal resolution hybrid screen design. Figure 3.10 shows the result of the generated halftone patterns that correspond to the same absorptance level, using equal and unequal resolution DBS.

We observe in Figure 3.10(c) that the halftone pattern created from equal resolution DBS has horizontal and vertical block patterns, which are reasonable to look at in the equal resolution case where there is no distortion in pixels. However, when looking at the unequal resolution rendering of the halftone pattern in Figure 3.10(c), which is as shown in Figure 3.10(b), we notice that the unequal dimension of the pixel in horizontal and vertical direction makes the horizontal and vertical pixel block patterns look strange. And indeed, in Figure 3.10(a), we noticed that when the unequal resolution DBS is applied to generate the unequal resolution halftone pattern, the algorithm tends to place dots in the form of diagonal patterns, which greatly increases the visual quality of the halftone pattern.

Figure 3.11 through 3.14 show the halftone ramp images that are generated from different unequal resolution hybrid screen and they are all rendered in unequal resolution with subpixels. The continuous ramp image on which we do the screening is an image whose gray scale value changes continuously from 255 to 0. Figure 3.11 shows the halftone ramp image generated from a supercell hybrid screen designed by the approaches in 3.3.4. We observe from the pattern that the the highlight region actually gives us an illusion that it is generated from conventional mono-chrome DBS algorithm, however in fact it is generated from screening. Figure 3.12 shows a halftone ramp image generated from a supercell hybrid screen designed by the approaches in 3.3.4 as well, but with a smaller supercell size. This time we observe that due to the deduction of the number for tone levels, the halftone output is not as visually homogeneous as the one shown in Figure 3.11. Figure 3.13 and 3.14 show the halftone ramp images generated from multilevel hybrid screen, one without considering the centroid rule and one considering the centroid rule. Although the supercell size is even smaller, due to the multilevel output per-pixel, the generated halftone quality look similar to the ones shown in Figure 3.11 and 3.12. Also we notice that after considering the centroid rule in multilevel hybrid screen design, the halftone patterns in Figure 3.14 is showing some visual improvements than that in 3.13, especially at the upper right area.

3.6 CONCLUSION

In this work we developed a novel hybrid screen design approach integrating the concept of superpixels and unequal resolution DBS algorithm. The integration of DBS in determining the dot growing sequence help produce decent halftoning quality. Our experiments show that the generated halftone images show plausible visual outcome for both bilevel screen and multilevel screen.

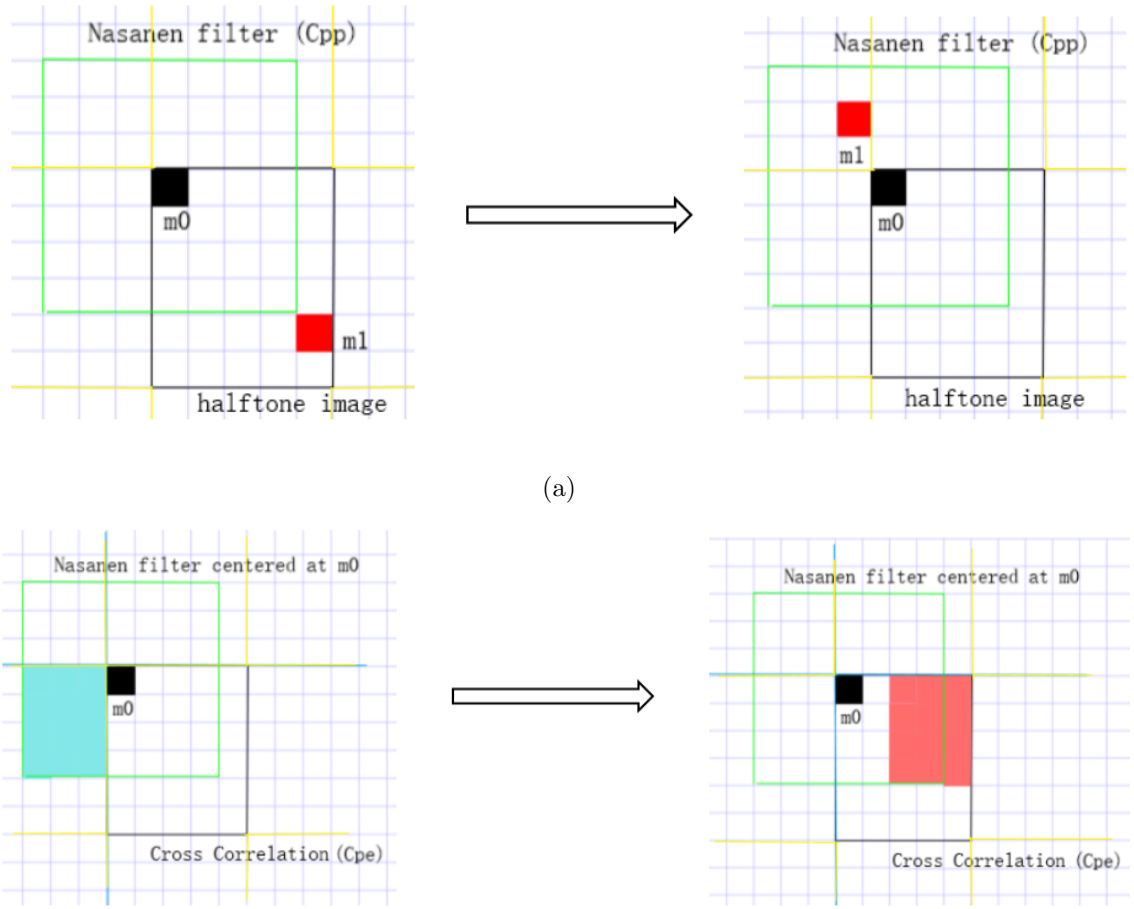
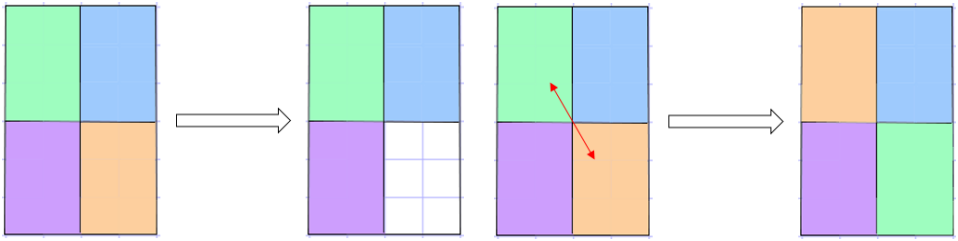
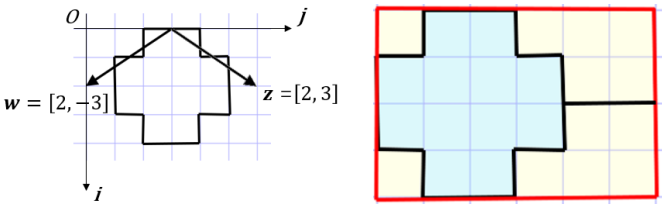


Fig. 3.2. Illustration of wraparound effect on a halftone image with size 6×5 and $c_{\tilde{p}\tilde{p}}$ with the size 7×7. (a) The wraparound effect when look up values from $c_{\tilde{p}\tilde{p}}[\mathbf{m}]$, this figure shows the instance where DBS tries to evaluate $c_{\tilde{p}\tilde{p}}[\mathbf{m}_0 - \mathbf{m}_1]$, which is already beyond the boundary of the cover range of $c_{\tilde{p}\tilde{p}}$. Then by using the modulo operator on $[\mathbf{m}_0 - \mathbf{m}_1]$ in both horizontal and vertical direction, we can see that the dot $\mathbf{m}_1$ as seen by the $c_{\tilde{p}\tilde{p}}$ has been moved from bottom right to upper left. And then $[\mathbf{m}_0 - \mathbf{m}_1]$ is now within the cover range of $c_{\tilde{p}\tilde{p}}$. (b) The wraparound effect when updating $c_{\tilde{p}\tilde{e}}[\mathbf{m}]$. As suggested by the figure on the left, the azure block is already out of the boundary of $c_{\tilde{p}\tilde{e}}$, however, the modulo operation on the indices of $c_{\tilde{p}\tilde{e}}$ will move the azure block to the area within the boundary of $c_{\tilde{p}\tilde{e}}$, as denoted by the red block. Then actually is it the subpixels within the red block that are updated. In both cases (a) and (b), the modulo operations will let the halftone pattern as seen by DBS be repeated with periodic 6×5.



(a) (b)

Fig. 3.3. Illustration of trial changes in the unequal resolution DBS in a halftone image that consists of 2×2 unequal resolution pixels. Each of the unequal resolution pixel is constituted by 3×2 subpixels. Our presumption is each of the subpixels represents a square block with physical size $1/1200 \times 1/1200$ inches, thus each unequal resolution pixel is of size $1/400 \times 1/600$ inches. Therefore, the DBS resolution as shown in the figure is 400×600 dpi. (a) A trial toggle in the unequal resolution DBS, the pixel on the bottom right of the halftone image get toggled off. Notice that all the 3×2 subpixels within this orange block are turned off as a unit. (b) a trial swap in the unequal resolution DBS. The unequal resolution pixels on the upper left and bottom right of the halftone image get swapped. And for this single swap operation, all the subpixels within a pixel are swapped as a unit. Therefore, for a single swap trial in this instance of unequal resolution DBS, there are actually 12 subpixels that are swapped.



(a) (b)

Fig. 3.4. Illustration of the microcell, tile vector, BSB, and cores. (a) A microcell formed by the tile vector $\mathbf{z} = [2, 3]$ and $\mathbf{w} = [2, -3]$. O is origin, $\mathbf{i}$ is vertical axis and $\mathbf{j}$ is horizontal axis. (b) A BSB containing 2 microcells, which are separatedly shown in the azure and yellow block.

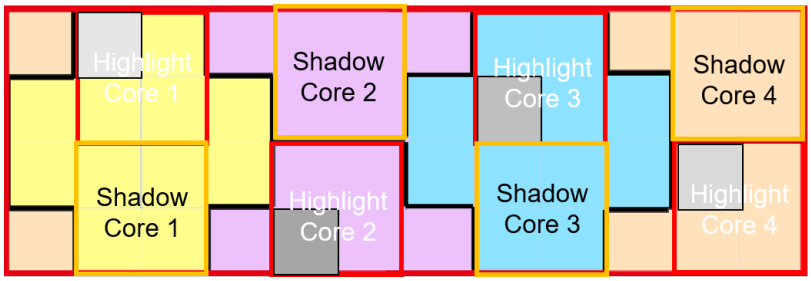
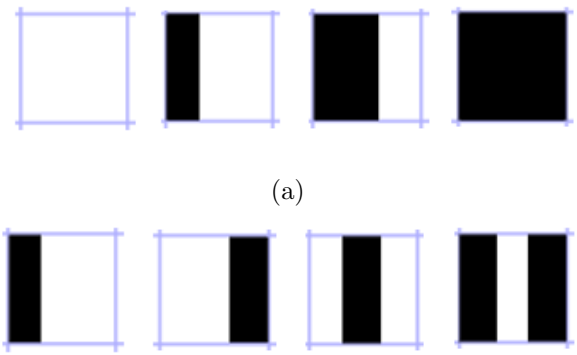


Fig. 3.5. An illustration of a 4×12 supercell and cores. The supercell contains four microcells, as shown in the color blocks. Each microcell contains a 2×2 highlight core and a 2×2 shadow core. Thus the supercell has 8 cores (4 highlight cores and 4 shadow cores) in total. The gray blocks are the candidate location for placing the first dot in each highlight core.



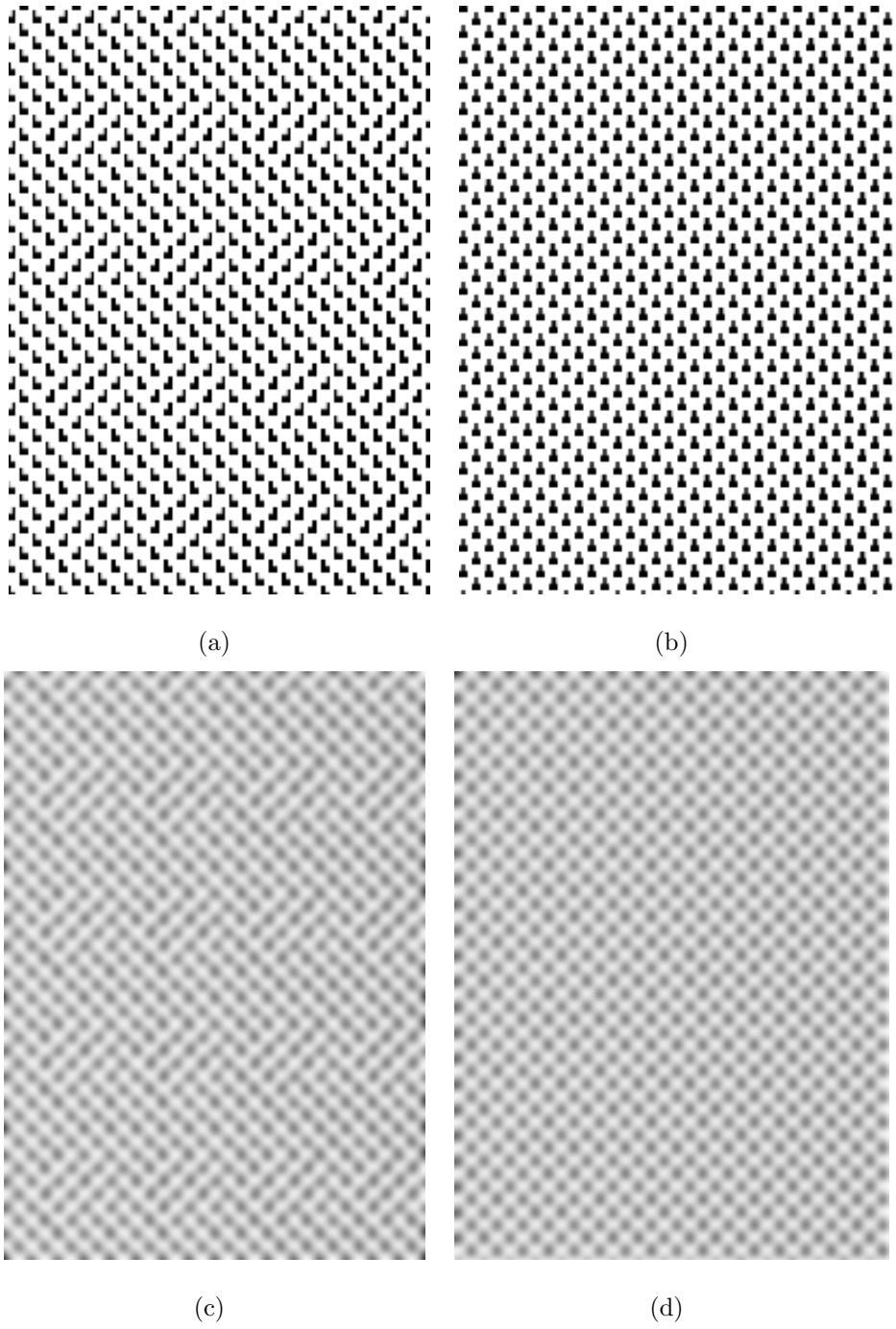
(a)

(b)

Fig. 3.6. An Illustration of multilevel output and justification modes. (a) A printer addressable pixel with four output levels and three thresholds. When the absorptance level is less than the lowest threshold, the pixel is not rendered; when the absorptance level is greater or equal than the lowest threshold but less than the medium threshold, 1/3 of the pixel is rendered; when the absorptance level is greater or equal than the medium threshold but less than the highest threshold, 2/3 of the pixel is rendered; when the absorptance level is greater or equal than the highest threshold, the pixel is fully rendered. (b) Four different justification modes for printer pixel. From left to right: left, right, center and split justified.

(a) (b) (c) (d)

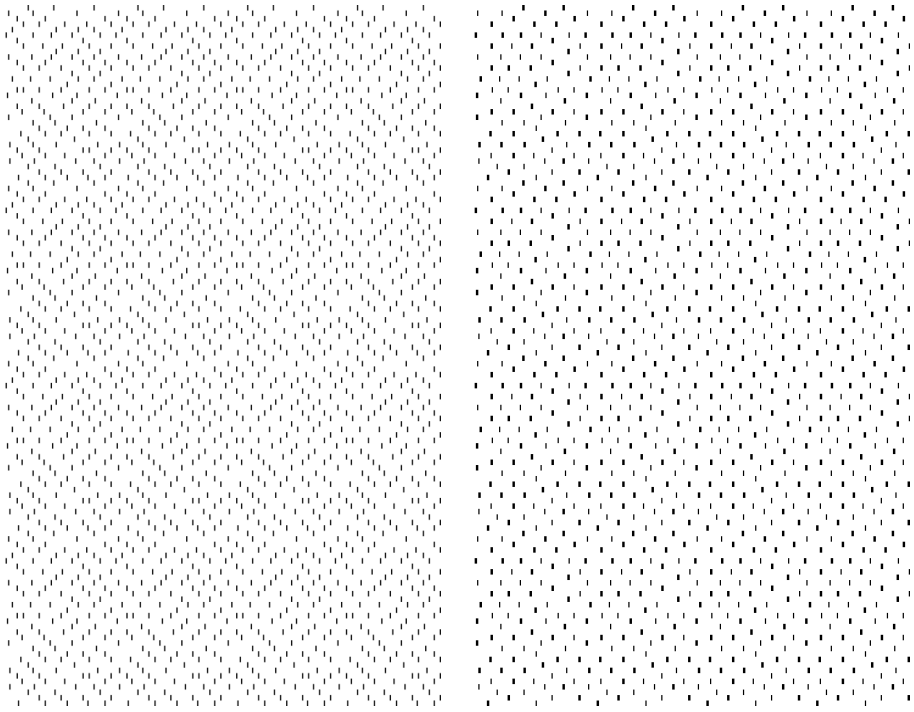
Fig. 3.7. A illustration of the symmetric rule and the compact rule in justification mode. The black blocks are full dots and the magenta blacks are newly added partial dots. (a) The newly added partial dots are located right below the full dot, obeying the symmetric rule. (b) The newly added partial dots are blow the full dot on the left, disobeying the symmetric rule. (c) The newly added partial dots are adjacent to full dots, obeying the compact rule. (d) The newly added partial dots are nonadjacent to full dots, disobeying the compact rule.



(a) (b)

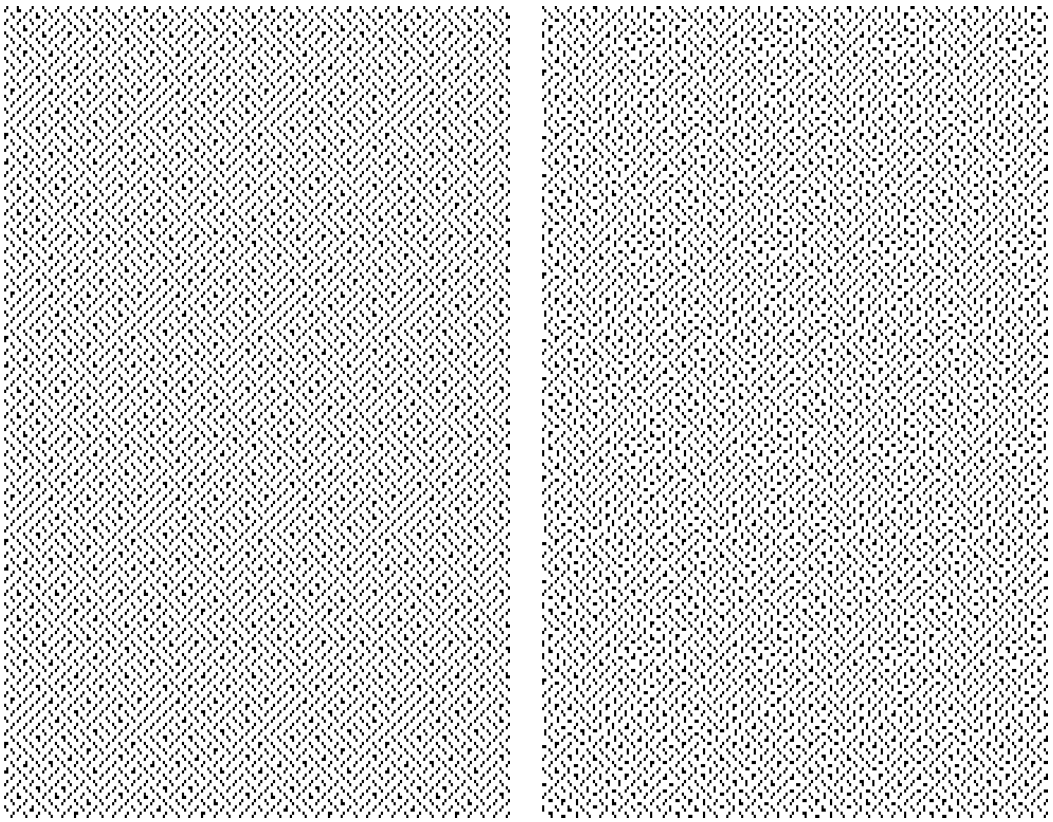
(c) (d)

Fig. 3.8. A comparison of the halftone patterns corresponding to the same gray level. (a) The halftone pattern generated from multilevel pixels, without the centroid rule in the partial dot justification. (b) The halftone pattern generated from multilevel pixels, with the centroid rule in the partial dot justification. (c) The halftone pattern in (a) after Gaussian filtering. (d) The halftone pattern in (b) after Gaussian filtering.



(a) (b)

Fig. 3.9. A illustration of two halftone patterns corresponding to the same gray level. (a) The halftone pattern generated by the unequal resolution DBS without the compact constraint. DBS is free to place the 1/3 partial dot (slivers) at any location within the highlight cores. (b) The halftone pattern generated by the unequal resolution DBS with the compact constraint. The new 1/3 partial dot (slivers) has to be placed adjacent to the existing slivers, and DBS choose the optimal location for this newly added sliver.



(a) (b)

(c)

Fig. 3.10. A comparison of the halftone patterns. (a) The halftone pattern generated by unequal resolution DBS and rendered in unequal resolution with subpixels. (b) The halftone pattern generated by equal resolution DBS and rendered in unequal resolution with subpixels. (c) The halftone pattern generated by equal resolution DBS and rendered in equal resolution.

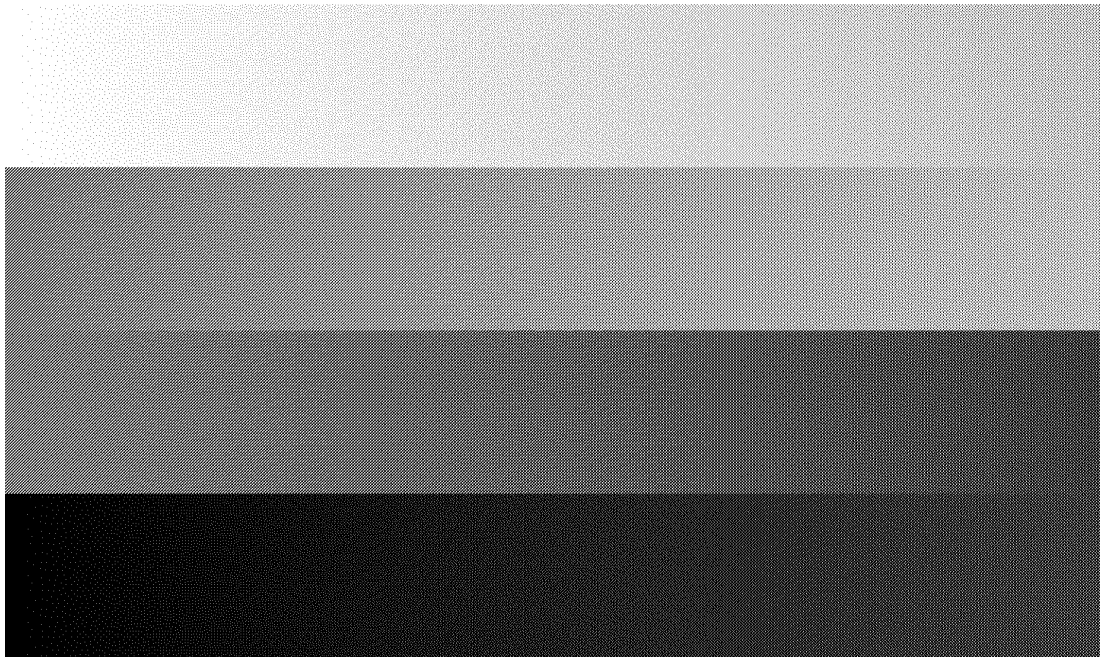


Fig. 3.11. Halftone pattern generated from a 64×60 supercell hybrid screen with bilevel unequal resolution pixel.

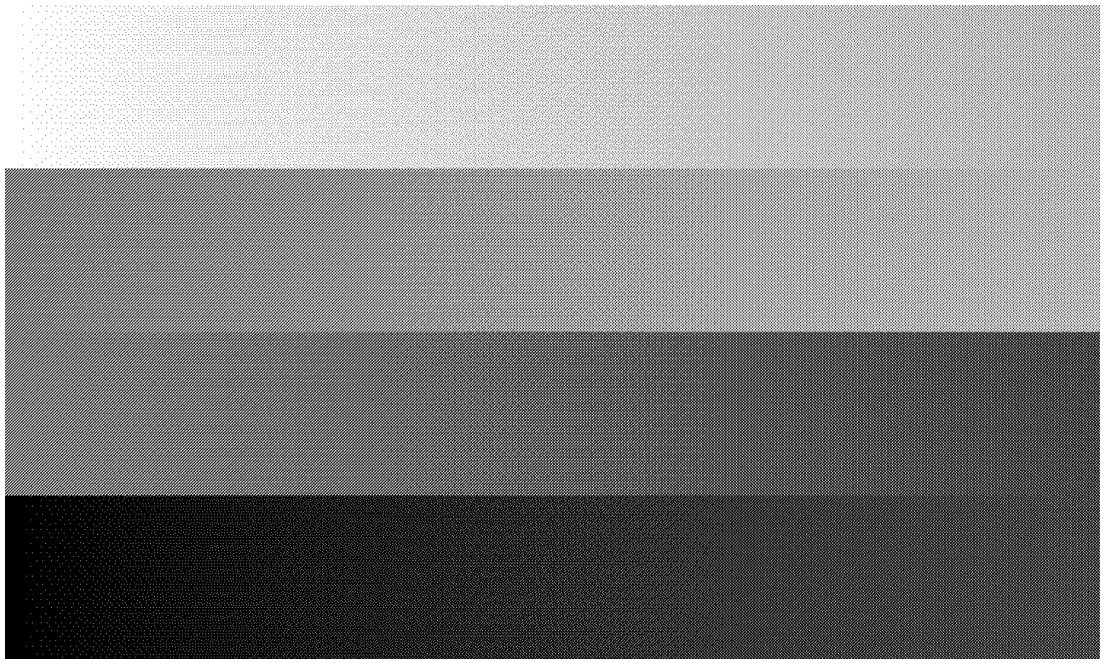


Fig. 3.12. Halftone pattern generated from a 32x30 supercell hybrid screen with bilevel unequal resolution pixel.

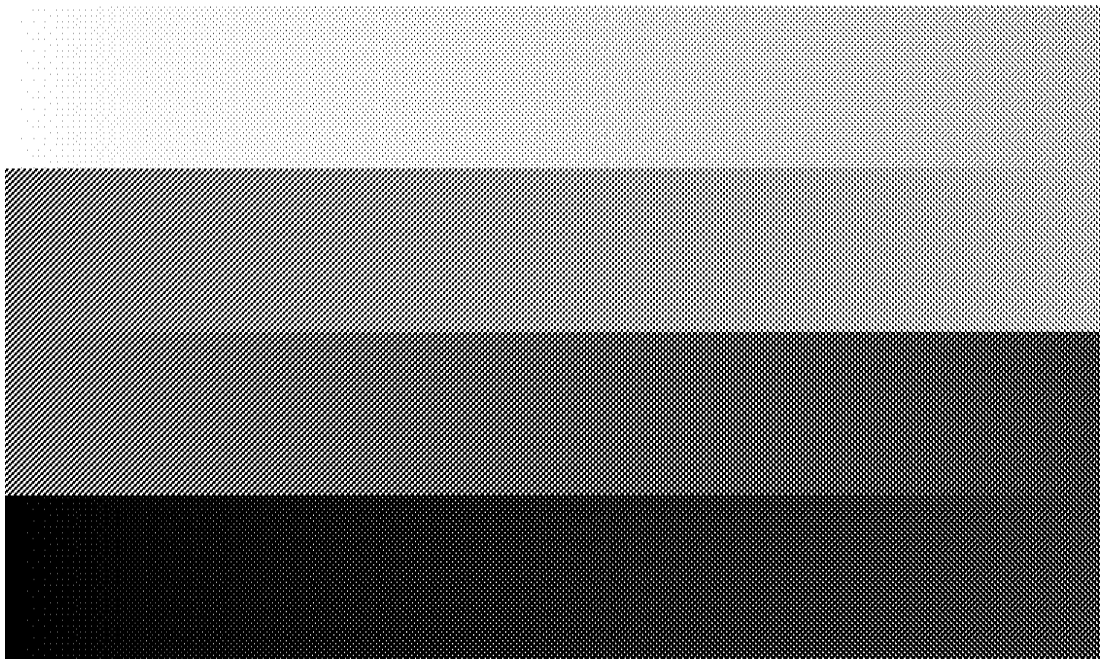


Fig. 3.13. Halftone pattern generated from a 20×24 supercell hybrid screen with 4-level unequal resolution pixel, with the compact rule but without the centroid rule.

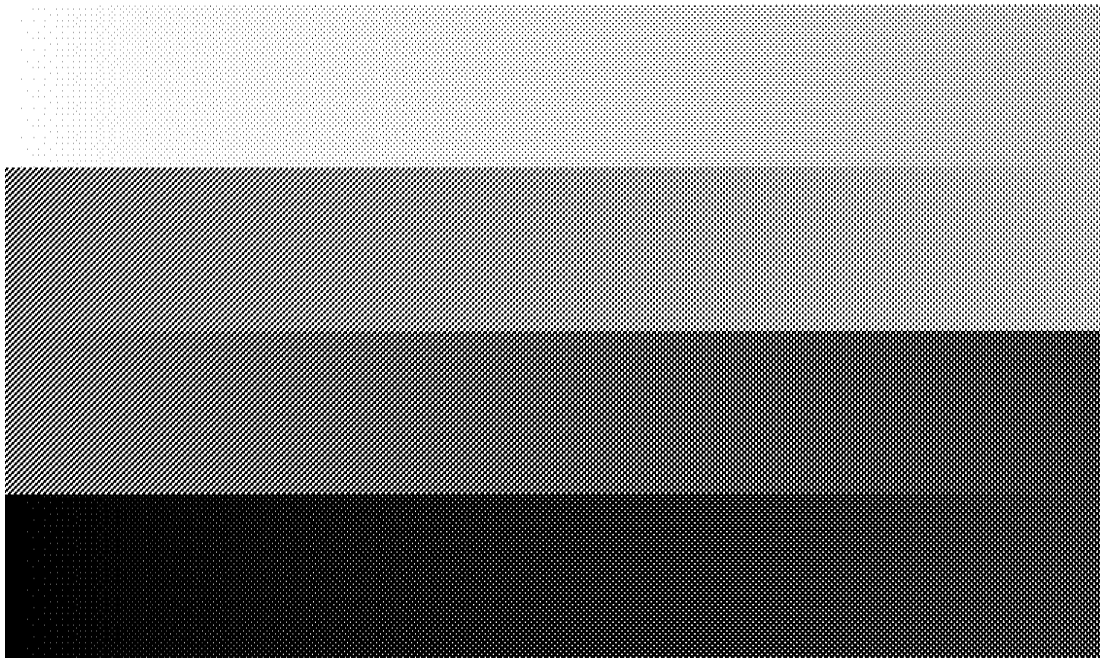


Fig. 3.14. Halftone pattern generated from a 20×24 supercell hybrid screen with 4-level unequal resolution pixel, with both the centroid rule and the compact rule.

4. FACE SET RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS

4.1 INTRODUCTION AND RELATED WORK

4.1.1 OVERVIEW OF FACE SET RECOGNITION

The research on face set recognition has attracted more and more attention from the computer vision community [26] [27] [28] [29] [30] [31]. Different from single face image recognition, which basically deal with single face feature extraction and simi-

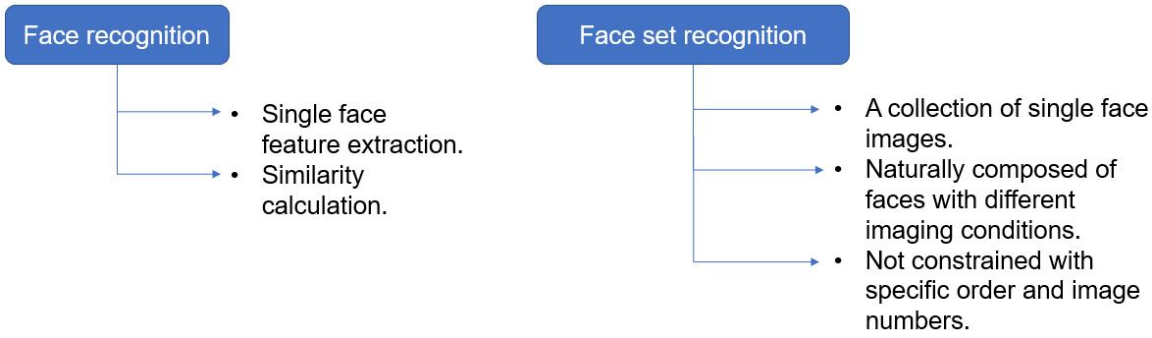


Fig. 4.1. The difference of face recognition and face set recognition. We see that the problem of face recognition essentially deals single face comparison, in which we are trying to find single feature vectors corresponding to each of the original faces, and then perform similarity or distance calculation for these feature vectors. For face set recognition, rather than dealing with single faces, the problem becomes more complicated as we are comparing different clusters of face images. The face images in each cluster have different pose orientation, lighting, sharpness, as well as number and order of images. As a result, the key problem of face set recognition is to develop an efficient and appropriate representation of face sets to gather dominant information in a set while disregarding noisy information.

larity calculation, more information can be extracted from sets of faces from different identities, which are naturally composed of faces with variations in image quality, facing directions and illumination conditions, etc. Figure 4.1 clearly shows the differences of face recognition and face set recognition. The face sets that we are discussing are usually from face video frames and face image clusters, which are naturally not constrained with specific orders and numbers of face images. Hence the key issue in face set recognition is to develop an efficient and appropriate representation of face sets, such that it can effectively gather the dominant information in a face set (e.g. information from sharper, more frontal face images) while disregarding information from noisy information.

One naive method for the face set recognition is to recognize the face image set as a cluster of face features that are extracted by deep CNN [32] [33], and hence in order to compare two face image sets, one needs the fused matching results from individual face feature pairs. Suppose n be the average number of face images in the face set, then a computational complexity of $O(n^2)$ is required per similarity comparison, which dramatically increases as the number of images goes up. This will be a critical problem when we want to build time-sensitive application such as real time video face recognition systems. Therefore, in our discussion afterwards, we are not considering such methods as a challenging competitor to our method.

We propose that it is desirable to develop a fixed-dimensional compact feature representation for the face sets, which is not related to the indexing and number of the images on each set, as is shown in figure 4.2 Such features should consider the information from all the images in the set while emphasize on the information from higher quality face images. Then it will allow direct, immediate computation for the set similarity or distance. One straightforward solution for generating such representation is the strategy to take the average of the features in a set. Although state-of-the-art deep neural networks are already able to generate very efficient feature representations for different identities (meaning that taking average of face feature is already very challenging), we believe that features generated from more frontal faces

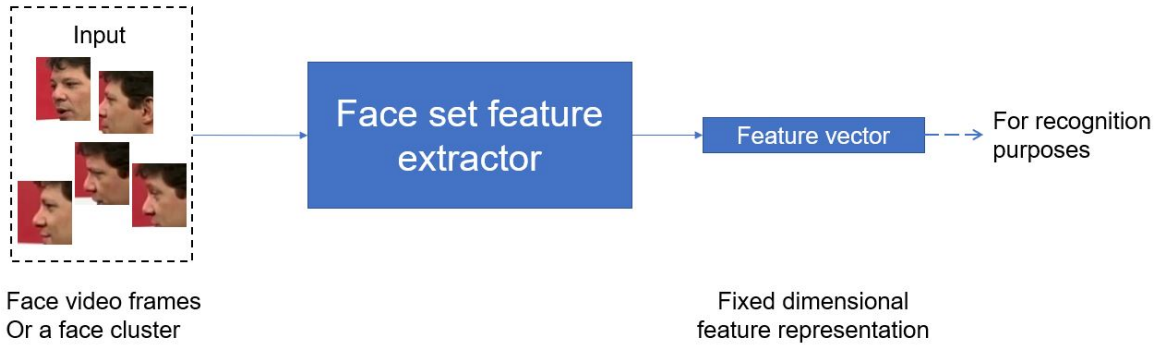


Fig. 4.2. The pipeline for face set recognition in imaging systems. On the left it shows the input to the system, which can be either face videos or face clusters. It typically contained frames of face images with different poses, sharpness, illumination, etc. Then the input face set is forwarded to a feature extractor, which is the core component that is going to be presented in the thesis. And then the desired output for the system is fixed dimensional feature vector representation. Fianlly the distances or the similarities between those feature vectors can be computed, thus those extracted feature vectors can be directly used for recognition.

and sharper images should be preferably considered over the features generated from occluded, non-frontal and blurred face images. Hence we are looking for a smart algorithm to capture these characteristics from input images and accordingly assign different weights to their corresponding feature vectors.

There has been active studies on the face set and video face recognition in the past. Many previous approaches have attempted to represent the face set manifolds or subspaces and compute the manifold similarity or distance for recognition [34] [35]. These methods may work well in some constrained scenarios but is has limited capabilies for handling more casual and complex situations where there are large variations between image frames. Besides these manifold-based methods, there also has been prior research on aggregating features using CNNs. Yang et al. [26] proposed a method of using the attention blocks as the universal face feature quality assessment, and then aggregate them, know as Neural Aggregation Network (NAN), as shown in

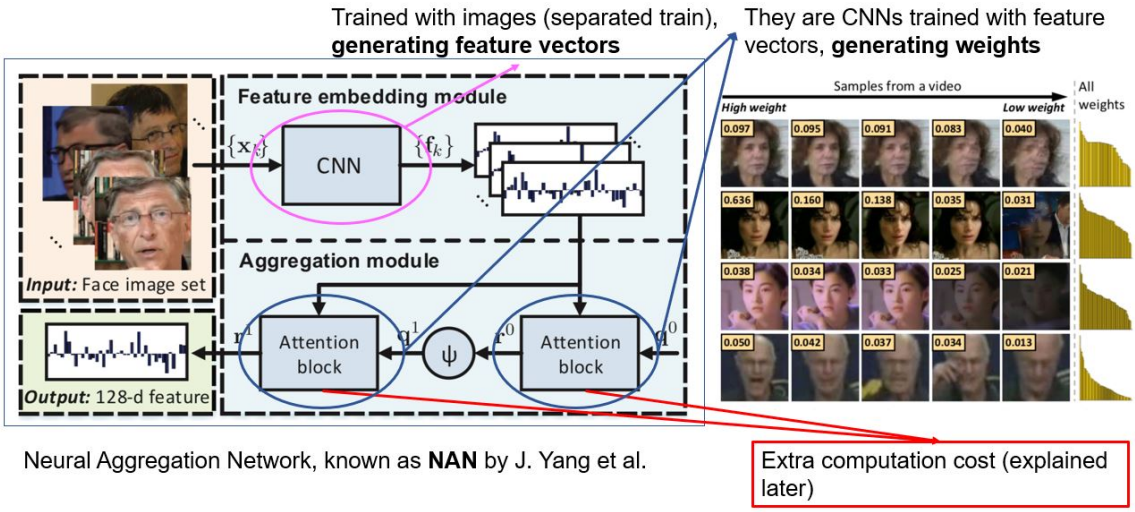


Fig. 4.3. The pipeline of Neural Aggregation Network (NAN) and examples of its output. NAN is one of the current state-of-the-art method for adopting CNN to smartly capture characteristics from face features and accordingly assign different weights to them to generate a fixed-dimensional feature representation for different face sets. Firstly, the input face images are forwarded to a single face embedding CNN to generate corresponding feature vectors, and then these vectors are forwarded to two cascaded attention blocks to generate according weights to these features, as shown on the right hand side. However, these two cascaded attention blocks will need to read all the input features and then generate linear weights for them. Thus for each time NAN operates on the input face set, extra running time and memory space is required.

figure 4.3. Their method borrows the differentiable memory addressing mechanisms from the Neural Turing Machine. Although they are also building a feature aggregation method based on smart weighing, the nature of the attention block that they used essentially needs to read all the feature vectors from the input before generating linear weights for them. Therefore, their proposed method needs preallocated memories and extra running time each time their method performs aggregation. In addition, the *weights* generated by their network is more like an arbitrary value that is assigned to each of the feature vector depend on the context of the input, thus

the relationship between these generated values and the *quality* of the original images are weak. Therefore the network cannot be treated as a universal face image feature quality assessment. To compensate for this shortcoming, Liu et al. [27] proposed a feature aggregation network called Quality Aware Network (QAN) which uses a Fully Convolutional Network (FCN) [36] to simultaneously generate face feature representation and assign weights to them. Even though this method emphasizes more on the relationship between the generated feature weights and face image quality, the combination of feature generation network essentially makes their aggregation network inflated. In practical scenarios where thousands of competitive CNN face feature extractors are available, this method shows its limit of not taking advantage of these fast marching approaches for single face embedding.

In summary, we want to look for an efficient and smart adaptive weighting schematics to linearly combine single face features from a face set together to form a discriminate face set representation. We designed a neural network to adaptively weigh features depending on the assessment of their original face images. We named our network Cluster Aggregation Network (CAN), whose parameters are trained through supervised learning using only the information of a normal face recognition task, e.g. face identities without any other extra supervised signals.

4.1.2 BRIEF INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS (CNN)

The Convolution Neural Network (CNN) [37] is Deep Learning algorithm which can take images as its input and assign learnable weights and biases to various aspects/objects in the image and be able to make difference on multiple images. In fact, the images are matrices containing pixel values, so the advantage of convolution operation is that by applying the 2 dimensional filter, both spatial and temporal dependencies in an image through the application of relevant filters are easier to be captured.

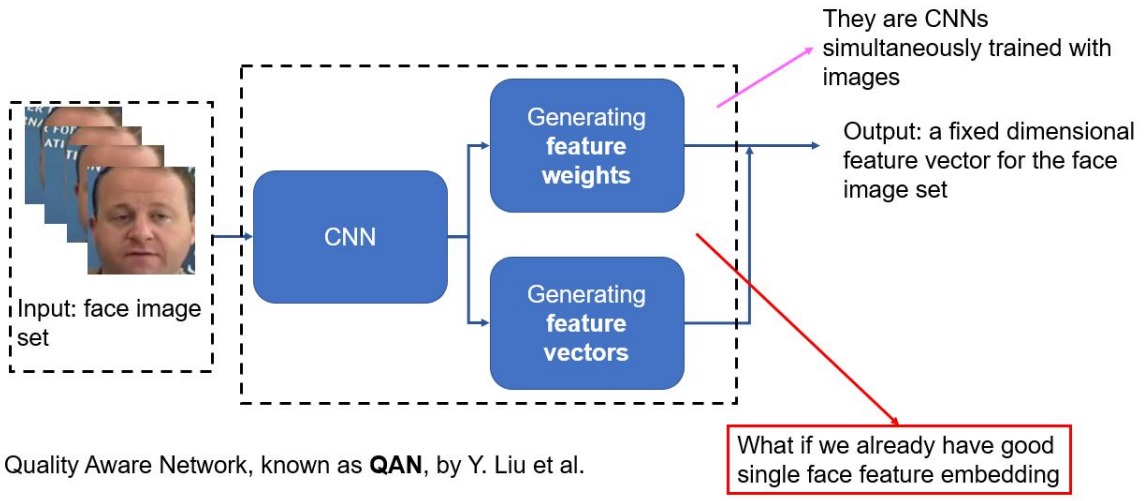


Fig. 4.4. The pipeline of Quality Aware Network (QAN). QAN is another state-of-the-art method for aggregating face images in a set and generating a compact feature representation for the set as to do the recognition. The input of the network is the face image set, which are firstly forwarded to an intermediate CNN, from where its output data are forwarded simultaneously to two parallel convolutional neural networks. The first CNN is for generating feature vectors from the intermediate output, and the second CNN is for assigning weights to the features. The QAN is huge and inefficient. And it is also not flexible to multiple more advanced face feature extractors.

The architecture of a CNN is inspired by the connectivity pattern of neurons in the human brain [37]. When processing images, individual neurons in the network respond to stimuli only in a restricted region, which is known as the *receptive field*. As a result, when processing the convolution operation on an image, neurons with different receptive fields forms different *feature maps*. Typically, the size of the feature maps are smaller than input images. When further convolution operation is applied to the feature maps, the succeeded feature maps will be even smaller in size, and eventually, the size of the feature maps are small enough in sizes that, they can be fully connected together to from a compact representation for the original image, and this is what we call a *feature vector*.

The name *pooling* sounds a little bit weird to the image and signal processing community, but in more familiar word it is just a 2-dimensional subsampling process. Similar to the aim of convolution operations, the pooling operation is responsible for reducing the spatial size of the convolved features. The main purpose for this is to decrease the computational power that is required to process the data through reducing dimensions. More importantly, it is useful for extracting dominant features that are rotational and positional invariant. And in this way the effectiveness of the trained model will be maintained. Depending on the pooling type, there are *max pooling* and *average pooling*. Max pooling performs as a noise suppressant thus we usually use it before average pooling.

4.1.3 KERNEL, STRIDE AND PADDING

Kernel, stride and padding are important concepts in convolutional neural networks. The *kernel* is the convolution filter matrix and kernel size is the filter's size horizontally and vertically. The stride decides how many steps a convolution core moves for each convolving operation. Padding adds arbitrary value to fill the borders of input images. The reason of padding is to preserve as much information as possible at the early stage of the neural network. A illustration for how the kernel, stride and padding affects the output of convolution layers is shown in Figure 4.5.

4.1.4 GENERAL TRAINING PROCESS FOR CNN

During the training process, the weights of the neurons in CNN are updated through a process called *backpropagation* [38], which can be separated into four parts: the forward pass, the loss function, the backward pass, and the weights updating. During the forward pass, you take a batch of the training data and set it as the input to the whole neural network. And then we compute the loss function L by calculating the distance from the output to the ground truth data. If we want to adjust the weights as to minimize the loss the function, we will have to do the derivative from

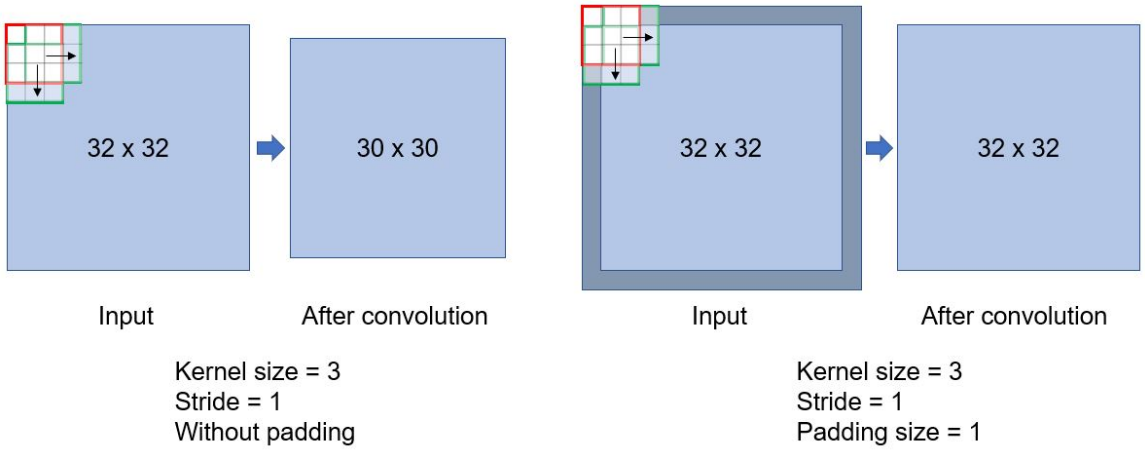


Fig. 4.5. Illustration of a convolution layer that operates on a 32 × 32 input image. On the left hand side it shows a convolution layer with kernel size 3, stride 1 and without padding. We see that without padding, the output size is reduced to 30 × 30, with the information for the pixels on the corners of the original images lost. On the right hand side it shows a convolution layer with kernel size 3, stride 1 and with padding size 1. We see that padding essentially adds value (typically zeros) on the border of the input image and by doing this the output size preserves the same as the input image. Therefore the information for the corner pixels from the input is maintained.

the loss function to the weights. Now considering the weights of the CNN that we want to train as w, and the initial neuron weights in the convolutional layers as w_i, then the backward pass process literally finds the derivative $\frac{\partial L}{\partial w}$ layer by layer. After that we do the weight update, which can be written as

$$w = w_i - \eta \frac{\partial L}{\partial w}. \tag{4.1}$$

The η here is known as the training rate. And it is often chosen manually to make sure it is neither too big or too small.

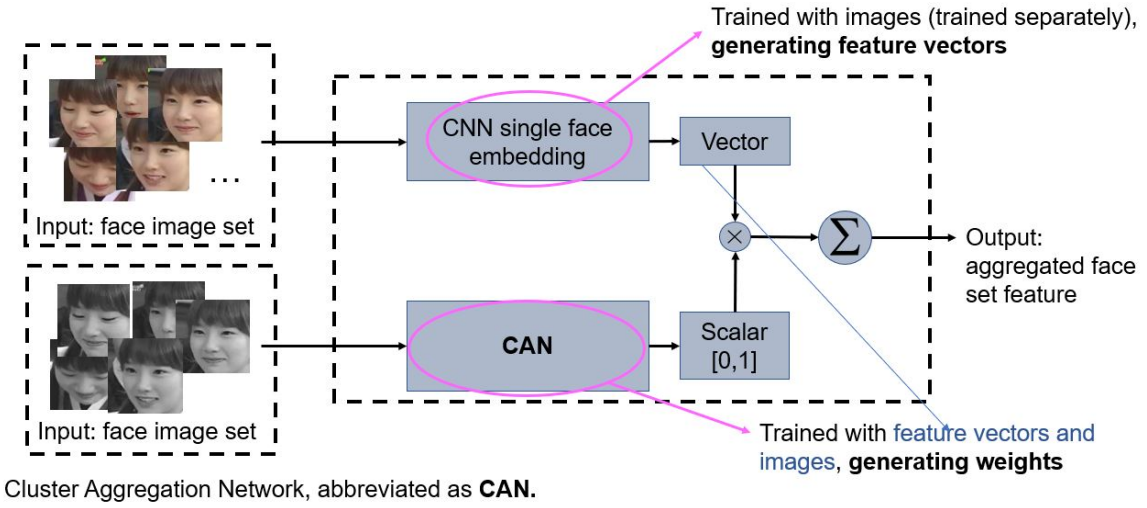


Fig. 4.6. The pipeline of Cluster Aggregation Network (CAN). CAN is the proposed method in the thesis for aggregating face images in a set and generating a compact feature representation for the set as to do the recognition. As shown in the figure, the method contains two independent CNNs and the input of the networks are both sets of face images. The CNN at the top is the single face feature extractor which essentially maps each of the input face images to feature vectors. The CNN at the bottom is CAN, which takes the images that are resized and transformed to grayscale from the upper input as its own input and generate a scalar between [0,1] to each of the images, depending on the network's assemssent for the input face images sharpness, facing directions, illuminations, etc. In brief, the network at the top maps images to vectors and the network at the bottom maps the same set of images to scalars, then the aggregated feature is calculated as the weighted summation combining the vectors and the scalars.

4.2 PROPOSED METHOD FOR FACE SET RECOGNITION

Recalling that in the last section, we have discussed about the disadvantages for the previous methods. Hence it is desirable to develop a more efficient novel face set recognition.

In the following discussions, a novel feature aggregation method is introduced. The core component of the proposed method is presented as *Cluster Aggregation*

Table 4.1.
Evaluation result for SphereFace face feature extractor model on LFW dataset

Fold	**Accuracy**
1	99.17%
2	99.00%
3	99.00%
4	99.50%
5	99.00%
6	99.17%
7	99.17%
8	99.00%
9	99.83%
10	99.33%

Network, abbreviated as CAN. The method uses CNNs and it not only utilizes the state of the art face feature extractor but also serves as a universal face image quality measurement. The proposed method for face set recognition method contains two methods, as shown in figure 4.6. Each module is a CNN which are trained separately. The first one is a face feature extractor using a deep CNN. The second one is the aggregation module which incorporates a quality assessment neural network that serves as weights generator for the features generated in the first module. As the key component in our feature aggregation module, it will be discussed in details next.

The succeeding chapters are arranged as the following: Firstly we will discuss the network structure for the CNNs, and then we will discuss the data preparation and the training procedure for the networks, finally we will discuss the testing results.

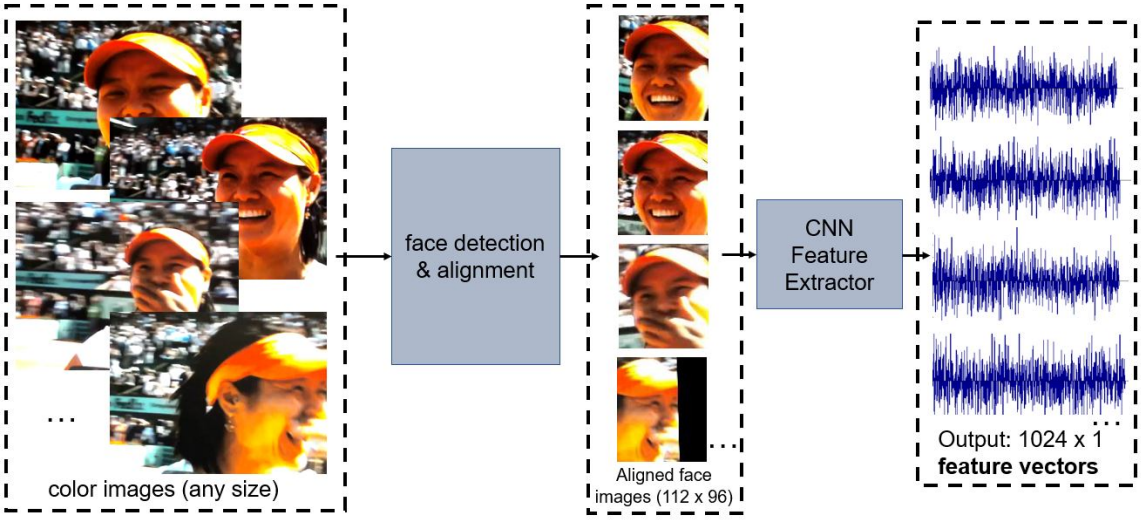


Fig. 4.7. The pipeline of single face embedding. It contains two modules: face detection and alignment and face feature extraction. For the first module, color images are passed to the neural networks to detect the faces that are present in these images. Then the faces are aligned such that their eyes and noses are in the same horizontal and vertical level. All detected and aligned face images are in the size 112 × 96. After that, the detected and aligned face images are then fed to the SphereFace netowrk to get fixed-dimensional feature vectors corresponding to each detected and aligned faces. The size for the output feature vectors are 1024 × 1 in size.

4.2.1 SINGLE FACE EMBEDDING

The face feature image embedding module of our method is a deep CNN, which embeds each image from a face set to a feature vector representation. In order to leverage modern CNNs with state of the art performance, we adopt the recent proposed SphereFace feature extractor [39], which produces 1024-dimension feature vector for each of the input face images, as shown in Figure 4.7. In the rest of the paper, we will refer to the employed SphereFace as CNN. The training of the CNN took a standard face classification and verification process and we trained it on the publicly available CASIA WebFace dataset [40]. Figure 4.8 shows an example of the CASIA webface

Fig. 4.8. An example of CASIA face dataset that is used for training SphereFace model for single face embedding. As shown in here, the dataset contains faces of different angles, illumination, ethnics, qualities. Therefore there is enough information in the dataset to train an efficient SphereFace single face feature extractor.

dataset. For evaluating the efficiency of our trained CNN, we tested its accuracy on 10 different folds of LFW dataset [41], as shown in table 4.1. Figure 4.9 shows an example of LFW dataset. We can see that SphereFace CNN model is accurate in identifying single faces and thus we can use it as a robust and efficeint single face embedding.

4.2.2 FACE FEATURE AGGREGATION MODULE

In this section we will be discussing the feature aggregation module which essentially takes the feature vectors from the feature embedding module and then generate face set represention. Consider that we are recognizing n pairs of face image sets with varying image number K_i. Therefore each face image set X^i is represented as $X^i = \{x_1^i, x_2^i, ..., x_k^i\}$ where $x_k^i, k = 1, 2, ..., K_i$ is the k-th image in the face set. Each image has a corresponding feature representation ϕ_k^i which is extracted by the feature embedding module. As for better readability, the upper index i is omitted in the following text. By forwarding the images $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k\}$ to the quality assessment network, we will have a quality score normalized to zero and one $\{\sigma_1, \sigma_2, ..., \sigma_k\}$ which

Fig. 4.9. An example of LFW face dataset that is used for testing the face recognition accuracy of our trained SphereFace model. We can see that the dataset is challenging for model evaluation because it contains faces images of different conditions.

corresponds to each of the input images. We then set $\sigma_j, j = 1, ..., k$ as the *weights* for the feature vectors $\phi_j, j = 1, ..., k$. Hence the aggregated feature $\mathbf{r}$ is represented as

$$\mathbf{r} = \frac{\sum_{j=1}^{k} \sigma_j \phi_j}{\sum_{j=1}^{k} \sigma_j}. \tag{4.2}$$

In this way, we can see that the aggregates feature vector is a weighted summation of the input single face feature vectors. Therefore the output feature vector is of the same dimension as a single face feature vector, and it is irrelevant to the order or the number of input face images in the face set.

Notice that here the *weights* that we assign for the feature vectors are directly related to the face image quality. In other words, the images that are preferred by our quality assessment network should be more frontal and sharper, thus the network tends to assign higher quality scores to those images. In the mean time, blurred and non-frontal face images are less preferred, leading to the result that the network tends to assign lower quality scores to them. We presume that sharp and frontal face

images are much more critical in the decision of recognition than blurred and side faces. Therefore, equation 4.2 will make the face set recognition system focus on good images while ignoring bad ones.

4.2.3 CLUSTER AGGREGATION NETWORK (CAN)

Our Cluster Aggregation Network (CAN) for face images is actually a Convolutional Neural Network, and its structure is shown in Figure 4.19. This network has a simple structure, which is partially inspired by [27] and [42]. And the process of generating quality score is a one-time network forwarding. In the earlier design, the network takes either RGB color images with size $224 \times 224 \times 3$ and the sigmoid layer's output has size 1×3, which corresponds to the Red, Green and Blue channels from the input images. In the latest design, we have reduced the input image size to 128×128 grayscale to better enhance the efficiency of CAN. Different from NAN which depends on input context to generate feature weights, our quality score is only related to one input image and the parameters of the neurons in the network, which are trained with standard face recognition techniques *without* any other supervision signals such as how good or bad the image is. The layer configuration for CAN is shown in table 4.2.

4.3 PREPARING TRAINING DATA FOR CAN

In this section we will discuss the training of CAN. The following sub-suctions are organized as the following: firstly we discuss training dataset, next we discuss the details of data preparation and pre-processing.

4.3.1 TRAINING DATASET

The training dataset for CAN is YouTube Face dataset [43]. It contains folders named by different people's names, which are videos frames from YouTube. An

Table 4.2.
Layer configurations for CAN (Input size can vary, the size as shown in the Table is due to input size of 128 × 128. The second Convolutional Layer is optional).

Layer	**Kernel Size**	**Stride**	**Padding**	**Input Size**	**Output Size**
Input	7	2	3	128 × 128 or 224 × 224 × 3	
Convolutional	3	1	1	128 × 128	64 × 64 × 64
Convolutional (Optional)	3	1	1	64 × 64 × 64	64 × 64 × 64
Pooling (Max)	3	2		64 × 64 × 64	32 × 32 × 64
Convolutional	3	1	1	32 × 32 × 64	32 × 32 × 64
Convolutional	3	1	1	32 × 32 × 64	32 × 32 × 64
Pooling (Average)	4	4		32 × 32 × 64	8 × 8 × 64
Fully Connection				8 × 8 × 64	1 ×1
Sigmoid				1 ×1	1 ×1

example of the dataset is shown in Figure 4.10. The dataset contains 3,425 videos of 1,595 different people. An average of 2.15 videos are available for each subject. The

shortest clip duration is 48 frames, the longest clip is 6,070 frames, and the average length of a video clip is 181.3 frames [43].

4.3.2 DATA PREPARATION

In order to train CAN, we need to prepare the face images data and the feature vectors data, since we will be using image-feature pairs for the training procedure, which will be discussed later. The data preparation has the following steps:

- Firstly a pre-trained CNN face detection and alignment model which we used in section 4.2.1 are adopted to detect and align the faces in the dataset. After this step we will have the face images with size $112 \times 96 \times 3$ and the faces are aligned such that the eyes and noses from different faces are on the same horizontal and vertical level.
- Next, the SphereFace model that we trained in section 4.2.1 for single face embedding is used here for extracting face features from the face images that are prepared in step one. After this step, we will have feature vectors with size 1024×1 that correspond to original face images.

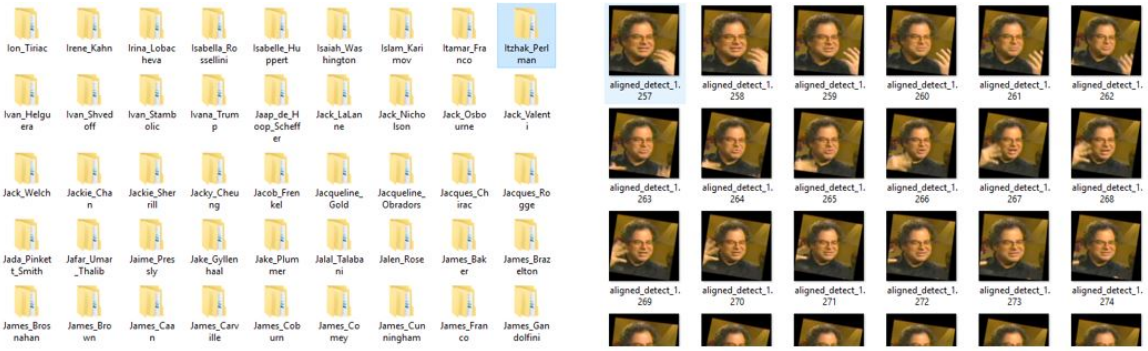


Fig. 4.10. An example of YouTube Face dataset. The image on the left shows the folders of video frames from different people. And the image on the right shows the opening up of one of the folders on the left, which are frames of videos from different person.

- Next, the face images we get in step one is converted to grayscale images with size 128 × 128 to fit with the input data dimension of the CAN.
- Finally the images are named as *'aligned_detect***.jpg'* and their corresponding feature vectors are named as *'aligned_detect***.mat'* MATLAB .mat files.

An example of the formulated face-feature pairs after the data preprocessing is shown in Figure 4.11.

4.3.3 DATA PREPROCESSING

In this step, each identity in the training data will have two *viewing perspective* for their face sets. To fulfill this, for each person's face-feature pair that we prepared in section 4.3.2, face frames from two separated videos are manually selected and split, such that each person will have two folders named *'sele1'* and *'sele2'* with each folder containing its own face-feature pairs, as illustrated in Figure 4.12.

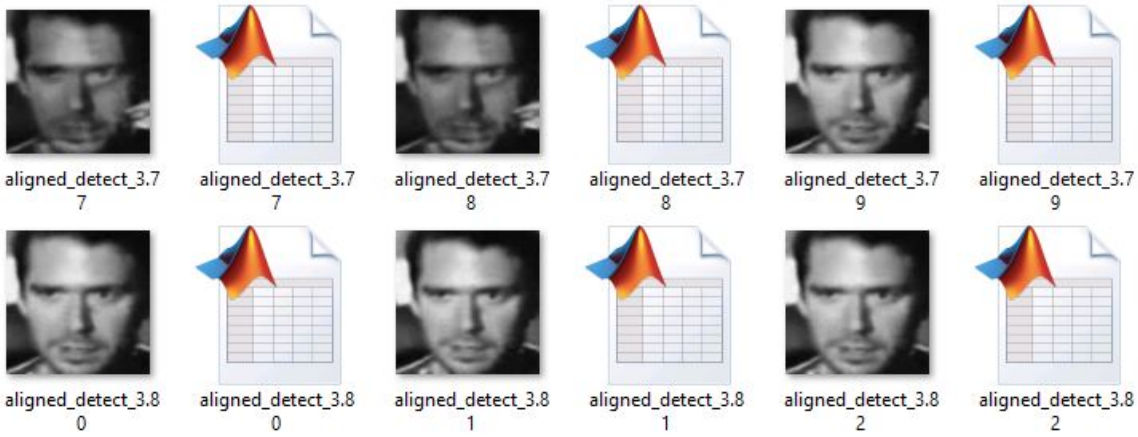


Fig. 4.11. An example illustrating the face-feature pairs after data preparation. The face images are aligned, resized and converted to 128 × 128 grayscale and the MATLAB files are storing the face images' corresponding feature vectors with size 1024 × 1.

A subset from the YouTube Face dataset is used for our current CAN training. After data preparation, the training data contains 178,460 image-feature pairs and 500 identities.

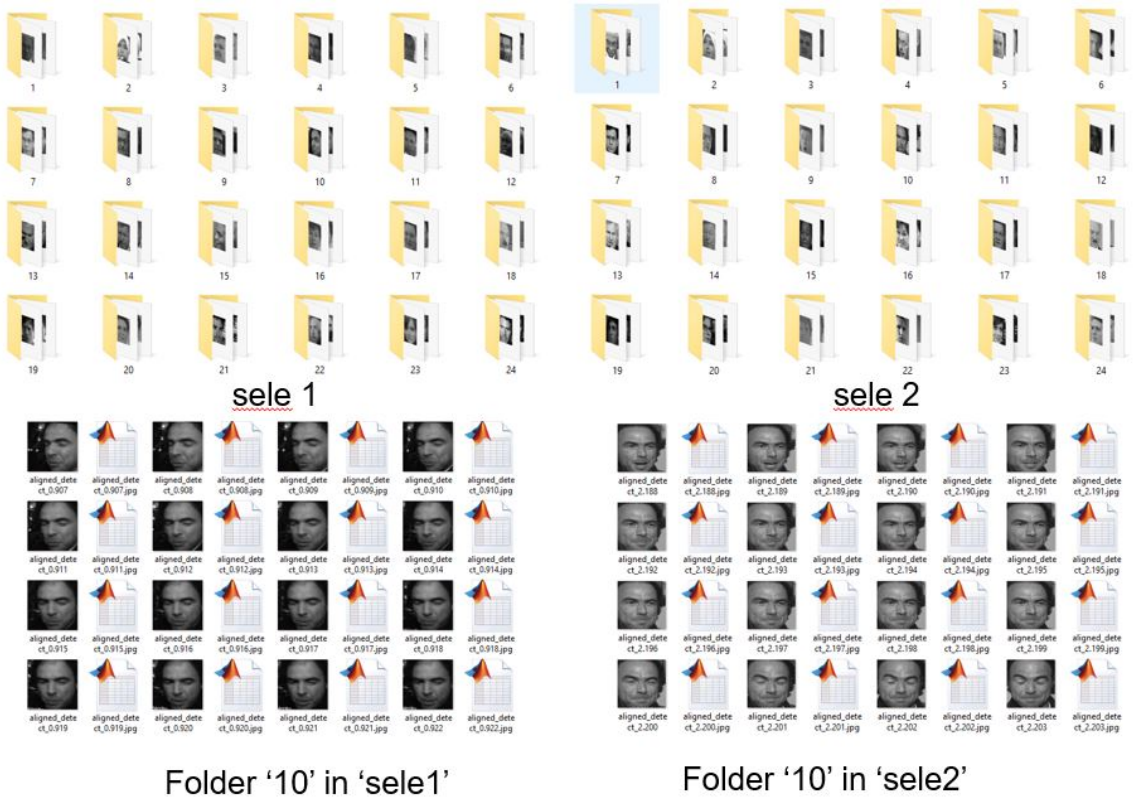


Fig. 4.12. An example illustrating the training data after data preprocessing. We see that for the identity shown here, he will have two separate folders of video frame images-feature vectors pairs in folder *'sele1'* and *'sele2'*, corresponding to two face sets from different viewing perspectives for the same person.

4.4 TRAINING PROCEDURE OF CAN

In this section a novel training procedure is presented. As discussed in section 4.1.1, previous state-of-the-art CNN for aggregating face features are either trained

with feature vectors only [26] or trained with images but inefficiently combined a feature extraction network training in the process [27]. In the theis a new training procedure combining the feature vector and the face images is presented, whose flowchart is shown in Figure 4.13

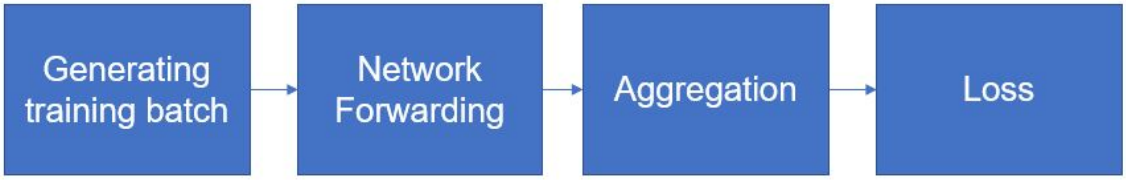


Fig. 4.13. Illustration for CAN training procedure. From left to right: The first module is training batch generation, in which small batches of face sets containing image-feature pairs are collected by the network as its input. The Second module is network forwarding, where the images in the previous step pass through CAN and the features in the previous step remain the same. After this step, the input features in the training batch and the output scalars generated by CAN is forwarded to the feature aggregation module where the feature representations for the face sets in the training batch are generated. Finally we calculate the distance between those generated face set features to compute our loss function.

4.4.1 TRAINING BATCH GENERATION

In this section we discuss in detail how the training batch is generated. The purpose for generating training batch is that, for each training epoch, three small face sets can be generated:

- **Anchor Set.** The small face set from person A
- **Positive Set.** The small face set from person A
- **Negative Set.** The small face set from person B

Here we are taking the *triplet* training strategy, meaning that the we want to make sure that not only the distance of different person's features are far off, but the distance of the same person's features are close by.

For our current training, a training batch size of 24 samples per epoch is used. And these samples spread evenly among the Anchor Set, Positive Set and the Negative Set. Meaning that each of the three sets get 8 samples of face-feature pairs per training epoch, as illustrated in Figure 4.14.

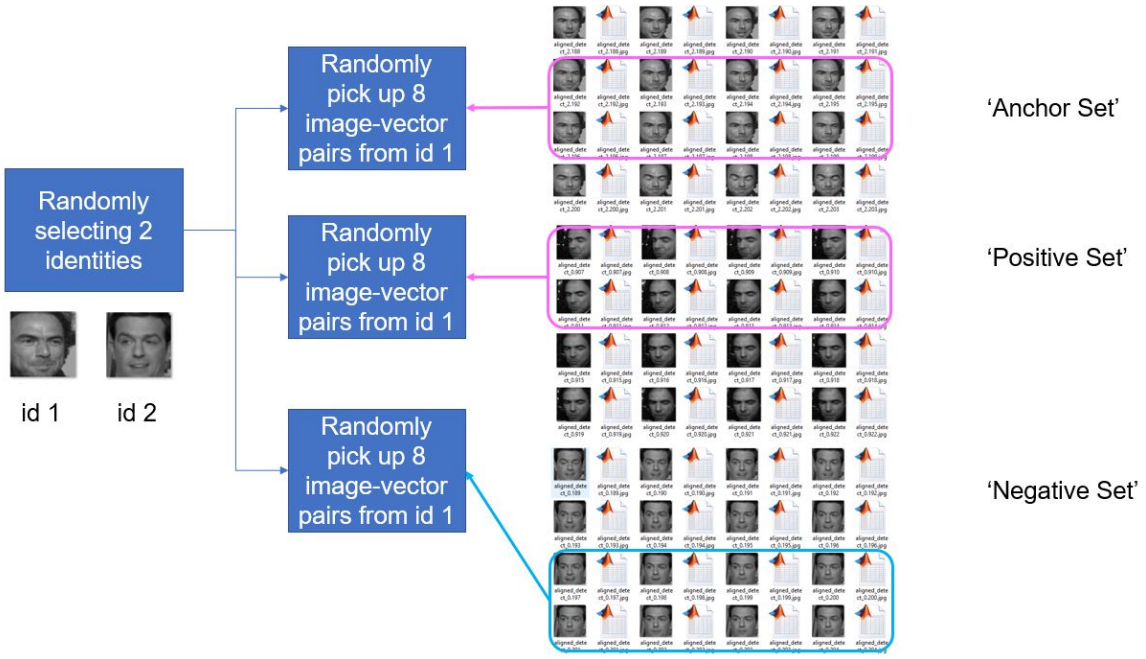


Fig. 4.14. Illustration of training batch generation for CAN training. For each training epoch, firstly 2 random identities with different numeric labels are selected from the training data. And then we randomly choose one of the identity as the Anchor Set. Then we randomly pick up 8 image-feature pairs from the *'sele1'* folder of this person. Next we randomly pick up 8 image-feature pairs from the *'sele2'* folder of this person as the Positive Set. Finally we randomly pick up 8 image-feature pairs from either 'sele1' or 'sele2' folder of another person as the Negative Set. Hence finally we will have 24 image-feature pairs corresponding to three face sets.

4.4.2 NETWORK FORWARDING AND AGGREGATION

In this section we will introduce Network Forwarding and Aggregation modules, which are steps after we acquire the training data and before we calculate loss function for each training epoch. Suppose the Anchor Set we get in section 4.4.1 is denoted as

$$\{i_{A1}, \phi_{A1}, ..., i_{A8}, \phi_{A8}\}, \tag{4.3}$$

where i represents images and ϕ represents feature vectors, and the Positive Set is denoted as

$$\{i_{P1}, \phi_{P1}, ..., i_{P8}, \phi_{P8}\}, \tag{4.4}$$

and the Negative Set as

$$\{i_{N1}, \phi_{N1}, ..., i_{N8}, \phi_{N8}\}, \tag{4.5}$$

And suppose the parameters of CAN forms a non-linear transformation from input image i to the output scalars (weights) σ, which is denoted as φ. Then we have

$$\varphi(i) = \sigma \tag{4.6}$$

During network forwarding, feature vectors remain the same while images pass through CAN, so that after one training iteration, we have the Anchor Set as

$$\{\varphi(i_{A1}), \phi_{A1}, ..., \varphi(i_{A8}), \phi_{A8}\}, \tag{4.7}$$

according to Equation 4.6, Equation 4.7 can be written as

$$\{\sigma_{A1}, \phi_{A1}, ..., \sigma_{A8}, \phi_{A8}\}. \tag{4.8}$$

Similarly, the Positive Set and the Negative Set after network forwarding can be written as

$$\{\sigma_{P1}, \phi_{P1}, ..., \sigma_{P8}, \phi_{P8}\}, \tag{4.9}$$

and

$$\{\sigma_{N1}, \phi_{N1}, ..., \sigma_{N8}, \phi_{N8}\}. \tag{4.10}$$

Hence after the step of network forwarding, we now have feature vectors and their corresponding weights. Given Equation 4.2, the aggregated feature for the Anchor Set $\mathbf{r}_A$ after passing it to the aggregation module in one training epoch is computed as

$$\mathbf{r}_A = \frac{\sum_{j=1}^{8} \sigma_{Aj}\phi_{Aj}}{\sum_{j=1}^{8} \sigma_{Aj}}. \tag{4.11}$$

And similarly, the aggregated Positive Set feature $\mathbf{r}_P$ and the aggregated Negative Set feature $\mathbf{r}_N$ after one training iteration is computed as

$$\mathbf{r}_P = \frac{\sum_{j=1}^{8} \sigma_{Pj}\phi_{Pj}}{\sum_{j=1}^{8} \sigma_{Pj}}. \tag{4.12}$$

and

$$\mathbf{r}_N = \frac{\sum_{j=1}^{8} \sigma_{Nj}\phi_{Nj}}{\sum_{j=1}^{8} \sigma_{Nj}}. \tag{4.13}$$

4.4.3 TRAINING LOSS AND SOLVER

We now have the feature representation $\mathbf{r}_A$, $\mathbf{r}_P$ and $\mathbf{r}_N$ for the Anchor, Positive and Negative Set that we randomly sampled during one training iteration. The next step is to find a loss function regarding these intermediate output and do the backward pass as we discussed in 4.1.4 and then update the CAN parameters after this training iteration. We want the distance between $\mathbf{r}_A$ and $\mathbf{r}_P$ be as small as possible while the distance between $\mathbf{r}_A$ and $\mathbf{r}_N$ be as large as possible. Hence a triplet loss function [32] here is defined on $\mathbf{r}_A$, $\mathbf{r}_P$ and $\mathbf{r}_N$: and

$$loss = \|\mathbf{r}_A - \mathbf{r}_P\|_2 - \|\mathbf{r}_A - \mathbf{r}_N\|_2 + \delta, \tag{4.14}$$

where the norms indicate Euclidean distance between vectors.

We see that since $\mathbf{r}_A$, $\mathbf{r}_P$ represents the same identity while $\mathbf{r}_N$ represents different identities, minimizing Equation 4.14 is equivalent to minimizing the distance between $\mathbf{r}_A$, $\mathbf{r}_P$ while maximizing the distance between $\mathbf{r}_A$, $\mathbf{r}_N$, since δ is a constant value. Therefore, in the training process, the loss functions regulate the parameters of the neurons such that they can gradually learn to ignore the low quality images in the

face set that prevent the distance between $\mathbf{r}_A$ and $\mathbf{r}_P$ from decreasing and $\mathbf{r}_A$ and $\mathbf{r}_N$ from increasing. And finally the neurons are trained to automatically protrude higher quality face images while repelling hard samples from face sets. In the training process, the standard backpropagation will adapt the network parameters such as to minimize the average loss as described in Equation 4.14.

In the actual training, CAN was trained on the prepared dataset with standard back propagation and a SGD solver [44].The learning rate is set as 0.001 and batch size be 24. We trained on NVIDIA® 1080Ti GPU with CUDA [45] enabled and it takes around 4 hours to finish the training process.

4.5 TESTING RESULTS OF CAN

We tested the efficiency of CAN on publicly available IJB-C dataset [46] [47]. The IJB-C dataset contains face images and videos that are captured from situations in the wild. It features a wide variety in pose, illumination and other kinds of imaging conditions, thus it is very challenging. We tested on the video frames from IJB-C, which has 500 identities with 2042 videos in total and around 11 frames per person's video. We firstly looked at the face image weights generated by CAN, and then we compared our face set recognition accuracy with reported results on IJB-C's 'compare' protocol for 1:1 *face verification* from current state of the art methods and also our own baseline method, which is averaging all the face features in a face set. In addition, we also analysis the CAN's advantage over the baseline method, which is taking the average of all the features in the set as the aggregated feature representation for the face set.

4.5.1 ILLUSTRATION FOR WEIGHTS: CAN OUTPUT

From equation 4.6 we have known that CAN forms a non-linear transformation from an input image to an output scalar, which we refer to as *weights*. Equation 4.11, 4.13 and 4.12 show that the weights directly determine how much a single face can

represent the whole face set. In other words, when gathering all the face information in the set to form an aggregated representation, faces with higher weights will present more information than faces with lower weights. Hence the output of CAN can be considered as a quality assessment for each of the faces in the set. In this section we tested CAN on specific face image set with variations of face posing, face obstacles and image sharpness to illustrate the meaning of *weights*.

Figure 4.15 illustrates the CAN output *weights* on the face image sets with various face posing (the first row to the fifth row) and with or without obstacles (the last row). We can see from row 1, 3 and 5 that, with the pose of faces in each of the frame becomes less and less frontal, the weights generated by CAN, which are shown in the Bar Chart, also decreases accordingly. And it means that CAN smartly considers that the *quality* of the video frames in video of row 1 and row 3 decreases from left to right. Now if we look at row 4, there is little variations between each of the frames in the video except for one frame, and CAN is able to detect that frame and assign smaller weight to it. And then if we look at row 2, there is very little variation for each of the frames in the video, thus the weights generated by CAN is very similar across the frames. We can also see from row 6 that the CAN is able to assign larger weights to more frontal faces and much less weights to faces with obstacles.

Figure 4.16 illustrates the CAN output *weights* on face images with different sharpness. The original face images are from the IJB-C's video frames, and the blurred images are acquired by Adobe Photoshop®. Different radius of Gaussian filter are selected to generate different levels of blurriness. And the images are forwarded to our trained CAN model to generate weights. It is encouraging to see that weights generated by CAN decrease with the increment of blurriness level of images, meaning that CAN assigns lower quality score to more blurred images, which is consistent with human cognition.

Table 4.3.
Verification Accuracy comparison of state of the art methods, our baseline methods and our proposed CAN network on IJB-C dataset.

Method	**Accuracy (%)**	**Model size**	**Running time**
EigenPEP [48]	84.8		
DeepFace-single [49]	91.4		
DeepID2+ [50]	93.2		
CNN+Baseline	94.65		
FaceNet [32]	95.12		
NAN [26]	95.52		
QAN [27]	96.17	50MB	15fps (GPU)
CAN(224, no Conv)	**96.97**	378KB	60fps (CPU)
CAN(224, Conv)	**97.17**	522KB	40fps (CPU)
CAN(128, no Conv)	**96.59**	378KB	60fps (CPU)
CAN(128, Conv)	**96.78**	522KB	40fps (CPU)

4.5.2 PERFORMANCE OF CAN

Comparing with previous work on the face set aggregation network, one big contribution of this work is the development of a small network that achieves better performance while requiring much less running time. From table 4.3 shows the comparison of accuracy and the running time of four different (very similar) version of our developed CAN network with previous state-of-the-art QAN method. We see that the model size of QAN is 132 times bigger than that of CAN while yielding worse accuracy performance. In addition, the running time for QAN is on NVIDIA® 1080 Ti GPU and for CAN on Intel® i5 4670K. The testing results show that CAN runs much faster than QAN even on a normal desktop CPU.

The accuracy of the proposed method for video face verification is shown in Table 4.3. We see from the results that QAN outperforms its previous state of the art by 0.65% and our proposed CAN outperforms QAN by 0.8%. The CAN variations does not show much performance change. In addition, it is noteworthy that CAN outperforms our baseline method by 2.32%, meaning that our smart aggregation actually works better than the naive aggregation method. An example of the failure case from using the baseline method, while non-failure from CAN is shown in Figure 4.18. We see that for face set on the left, the image on the right is much more blurred than the image on the left, in addition, the pose of the face on the right is not as frontal as the one on the left. Therefore, by forwarding the two face images to CAN for quality assessment, our network is able to smartly generate higher score for the image on the left while much lower score for the image on the right. As for the face set on the right, the images actually do not vary a lot with sharpness but varying in face pose. Hence our proposed CAN is also able to detect these variations and adaptively assign weights to the images. We notice that for this image set, from left to right the facing angle of the person is gradually decreasing, thus interestingly, weights generated by CAN is also in descending order. This is actually consistent with human cognition.

4.5.3 FAILURE CASES OF FACE SET RECOGNITION

Figure 4.17 illustrates the failure cases for our Face Set Recognition method. We can see from this figure that the baseline method fails in correctly recognizing some very apparent different identities and by using CAN as the feature aggregation method, the false recognition is reduced and these face sets be correctly identified as belonging to different persons. However, from this figure we can also see that in the current Face Set Recognition methods with CAN, recognition of some sets of faces are still very challenging due to some single face's high similarity or ambiguousness. In Figure 4.6 we have seen that CAN and single face embedding work corporately.

Therefore, in the future we can train a better single face embedding and re-train CAN with this new feature extractor to improve the performance of CAN even better.

4.6 CONCLUSION

We have presented a Cluster Aggregation Network for face set representation and recognition. It gathers all the input frames and uses an adaptive weighing schematics based on the smart assessment for face images quality to generate a set of variable weights for the input, resulting a compact representation of the face image set. This method is simple, competitive and can also be used in many scenarios such as video face recognition, face clustering and many other vision tasks.

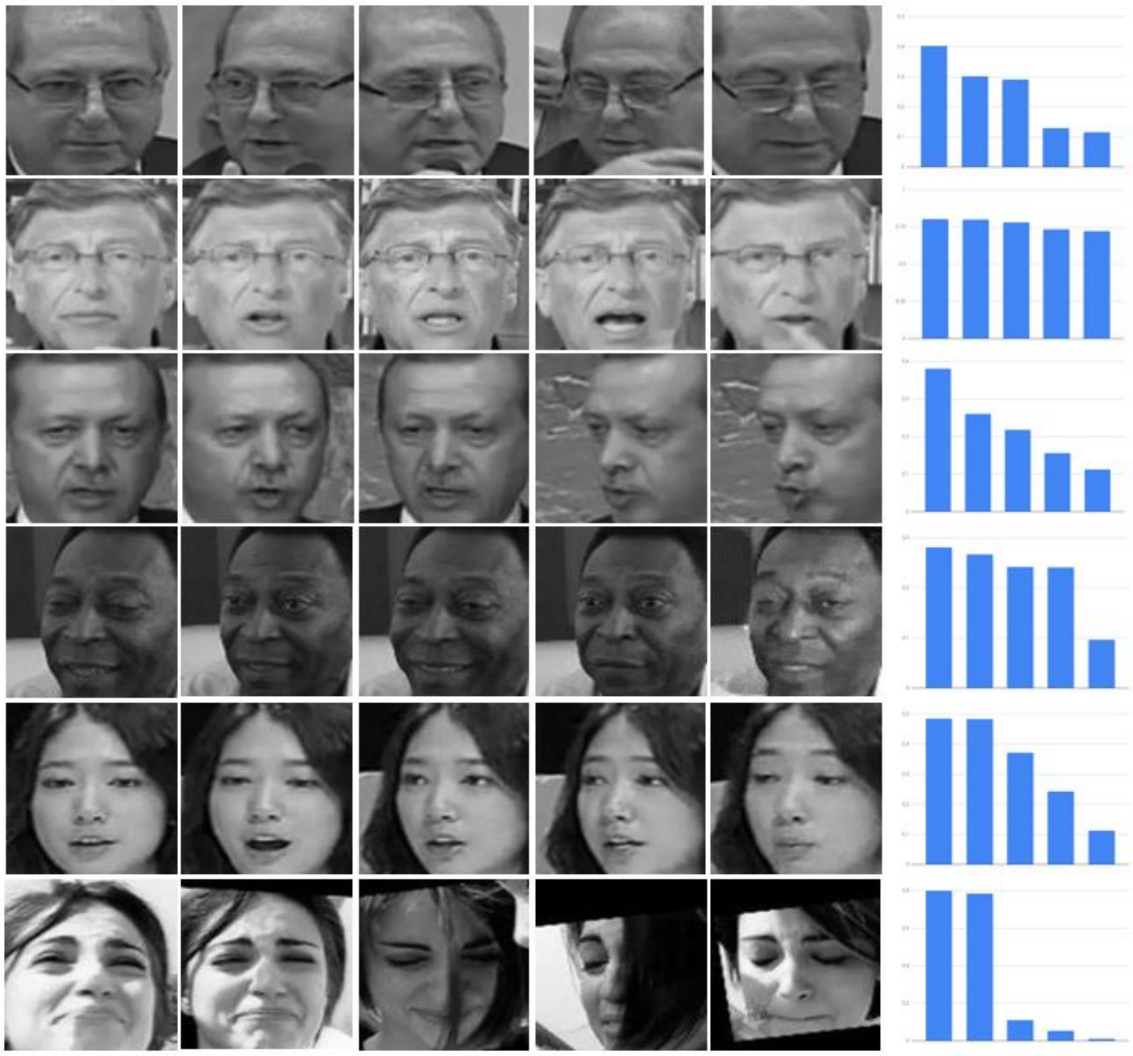


Fig. 4.15. An illustration of weights generated by CAN (The version is 128 with Conv. Layer.) for different image sets with variation of face posing and face obstacles. For each row, the images are the frames in a short face video clip. And the Bar Chart is the weights output from CAN on each of the frame image. Higher values in the Bar Chart means greater values of the weights. And for each of the rows, each bar in the Bar Chart corresponds to each image from left to right.

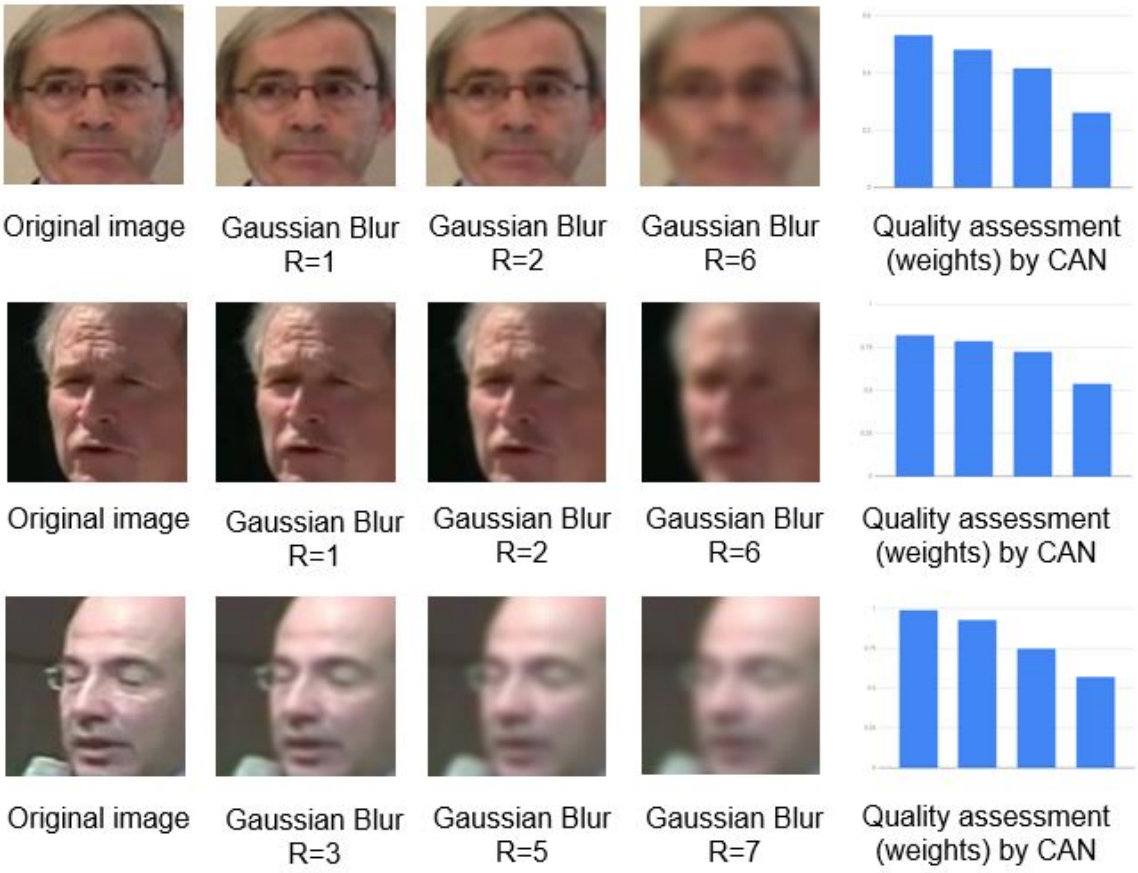


Fig. 4.16. An illustration of weights generated by CAN (The version is 224 without Conv. Layer.) that is capable of capturing the sharpness of images and assess different qualities accordingly. The CAN takes different images as input and generate weights for these images, as shown in the Bar Chart. Higher bars in the Bar Chart means larger weights and each bar in the Bar Chart corresponds to each image from left to right.

Fig. 4.17. False Face Set Recognition results. (a) Video face pairs detected by CAN as the same identity. (b) Video face pairs detected by baseline face set aggregation (See Section 4.5) as the same identity but detected by CAN as different identity.

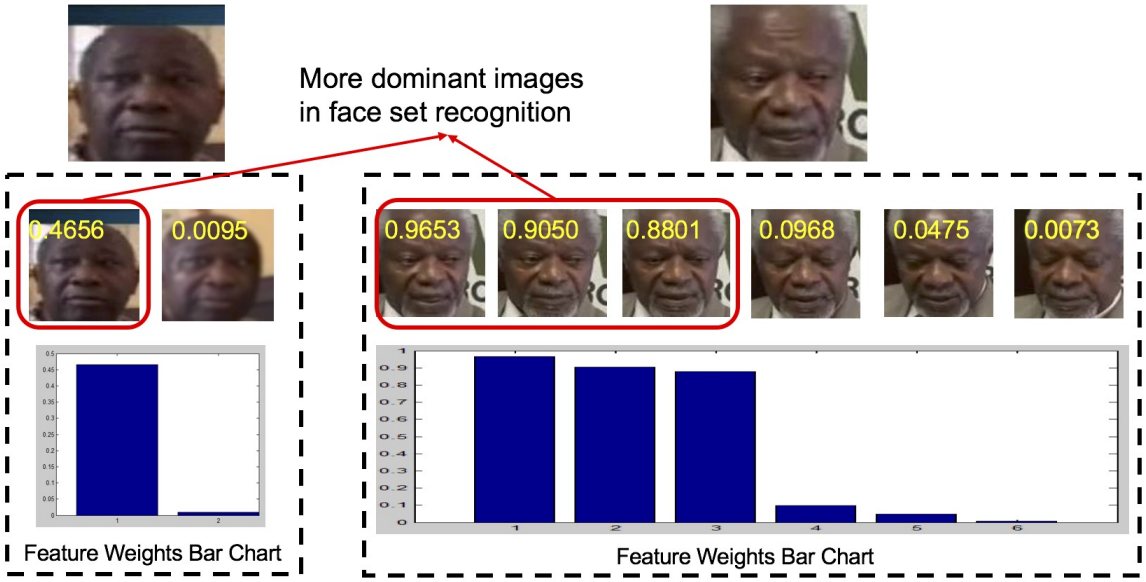


Fig. 4.18. An example of successful verification for two face sets using CAN but unsuccessful verification using the baseline. The yellow digits on top is the quality score or the feature weights generated by CAN, which has been averaged on three channels. The Bar Chart visualizes the variation of the feature weights.

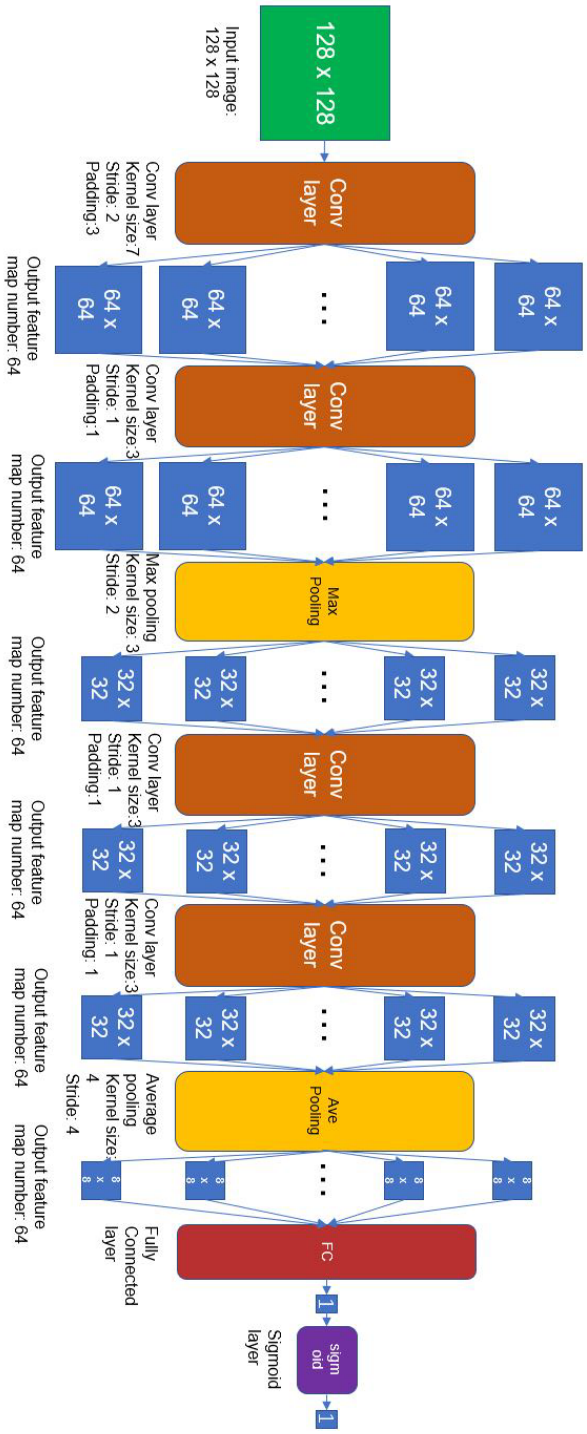


Fig. 4.19. The structure of CAN convolution neural network (rotated view). It takes grayscale image with size 128 × 128 as its input and contains four Convolutional Layers and two Pooling Layers. The Fully Connected Layer generates a single scalar for the input and the Sigmoid Layer at the very last of the network normalize the scalar to [0,1], which is the final score assigned to the input face image, or the weights assigned to the face's corresponding feature.

REFERENCES

REFERENCES

[1] J. Allebach, "Principles of digital color imaging systems - introduction," *Principles of Digital Color Imaging Systems*, 2017.

[2] C. Lee and J. P. Allebach, "The hybrid screen – improving the breed," *IEEE Transactions on Image Processing*, vol. 19, no. 2, pp. 435–450, Feb 2010.

[3] G. Y. Lin and J. P. Allebach, "Generating stochastic dispersed and periodic clustered textures using a composite hybrid screen," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3746–3758, Dec 2006.

[4] Q. Lin and J. P. Allebach, "Color fm screen design using dbs algorithm," *Proc. SPIE*, vol. 3300, pp. 353–361, 1998.

[5] G. Lin and J. P. Allebach, "Generating stochastic dispersed and periodic clustered textures using a composite hybrid screen," *IEEE Trans. Image Processing*, vol. 15, no. 12, pp. 3746–3758, 2006.

[6] ——, "Multilevel screen design using direct binary search," in *Color Imaging: Device-Independent Color, Color Hardcopy, and Applications VII, San Jose, CA, USA, January 19, 2002*, 2002, pp. 264–277. [Online]. Available: http://dx.doi.org/10.1117/12.452997

[7] P. Li and J. P. Allebach, "Tone-dependent error diffusion," *IEEE Transactions on Image Processing*, vol. 13, no. 2, FEBRUARY 2004.

[8] "Model-based digital halftoning," *IEEE Signal Processing Magazine*, vol. 20, no. 4, pp. 14–27, July 2003.

[9] F. A. Baqai and J. P. Allebach, "Computer-aided design of clustered-dot color screens based on a human visual system model," *Proceedings of the IEEE*, vol. 90, no. 1, pp. 104–122, Jan 2002.

[10] H. Ding, R. Bala, Z. Fan, R. Eschbach, C. A. Bouman, and J. P. Allebach, "Semi-automatic object geometry estimation for image personalization," vol. 7533, pp. 753 304–753 304–11, 2010.

[11] *Dzine Steps: Create, Customize, Collaborate!*, http://dzinesteps.com/.

[12] *M S International, Inc. Virtual Kitchen Designer*, https://www.msistone.com/virtual-kitchen-designer/.

[13] *Zillow Digs: Find inspiration for your home project.*, http://www.zillow.com/digs/.

[14] *Sherwin-Williams ColorSnap Visualizer*, https://www.sherwin-williams.com/visualizer/.

[15] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[16] D. C. Lee, A. Gupta, M. Hebert, and T. Kanade, "Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces," *Advances in Neural Information Processing Systems (NIPS)*, vol. 24, November 2010.

[17] C. Tai, T. Liu, J. Bagchi, F. Zhu, and J. Allebach, "Click-based interactive segmentation with graph cut," Imaging and Multimedia Analytics in a Web and Mobile World 2017*, (Part of IS&T Electronic Imaging 2017), J. Allebach, Z. Fan, and Q. Lin, Eds., San Francisco, CA, 29 January -2 February 2017 (2017).*, 2017.

[18] K. Ziga, J. Bagchi, J. Allebach, and F. Zhu, "Non-parametric texture synthesis using texture classification," Computational Imaging XIV*, (Part of IS&T Electronic Imaging 2017), C. Bouman and R. Stevenson, Eds., San Francisco, CA, 29 January -2 February 2017 (2017)*, 2017.

[19] G. Sharma, *Digital Color Imaging Handbook*. Boca Raton, FL, USA: CRC Press, Inc., 2002.

[20] B. E. Bayer, "An optimum method for two-level rendition of continuous-tone pictures,," *IEEE Int. Conf. Communications*, vol. 1, pp. 11–15, 1973.

[21] C. W. Wu, C. P. Tresser, G. R. Thompson, and M. J. Stanich, "Supercell dither masks with constrained blue noise interpolation," *IS&Ts NIP 17: Int. Conf. Digital Printing Technologies*, vol. 1, pp. 487–490, 2001.

[22] R. Bartels, "Reducing patterns in the fm part of tile-based hybrid screens," *IS&Ts 2002 PICS Conf*, pp. 241–244, 2002.

[23] S. H. Kim and J. P. Allebach, "Impact of hvs models on model-based halftoning," *IEEE Transactions on Image Processing*, vol. 11, no. 3, pp. 258–269, Mar 2002.

[24] R. Näsänen's, "Visibility of halftone dot textures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-14, no. 6, pp. 920–924, Nov 1984.

[25] D. Kacker, T. Camis, and J. P. Allebach, "Electrophotographic process embedded in direct binary search," *Proc. SPIE*, vol. 3963, pp. 468–482, 1999.

[26] J. Yang, P. Ren, D. Zhang, D. Chen, F. Wen, H. Li, and G. Hua, "Neural aggregation network for video face recognition," *Proceedings of the 32th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4362–4371, 2017.

[27] Y. Liu, Y. Junjie, and W. Ouyang, "Quality aware network for set to set recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] Z. Cui, W. Li, D. Xu, S. Shan, and X. Chen, "Fusing robust face region descriptors via multiple metric learning for face recognition in the wild," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3554–3561, 2013.

[29] H. Li, G. Hua, Z. Lin, J. Brandt, and J. Yang, "Probabilistic elastic matching for pose variant face verification," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3499–3506, 2013.

[30] H.Mendez-Vazquez, Y.Martinez-Diaz, and Z.Chai, "Volume structured ordinal features with background similarity mea- sure for video face recognition," *International Conference on Biometrics (ICB)*, 2013.

[31] L. Wolf and N. Levy, "The svm-minus similarity score for video face recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3523–3530, 2013.

[32] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering." *CoRR*, 2015.

[33] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1701–1708, 2014.

[34] R.Wang, S.Shan, X.Chen, and W.Gao, "Manifold-manifold distance with application to face recognition based on image set," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2008.

[35] ——, "Statistical computations on grassmann and stiefel manifolds for image and video-based recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2273–2286, 2011.

[36] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 640–651, 2017.

[37] J. Schmidhuber, "Deep learning in neural networks: An overview," *arXiv:1404.7828*, 2014.

[38] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural Networks for Perception (Vol. 2)*, pp. 65–93, 1992.

[39] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[40] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *arXiv:1411.7923*, 2014.

[41] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts, Amherst*, no. 07-49, October 2007.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014.

[43] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 529–534, 2011.

[44] L. Bottou, "Stochastic gradient descent tricks," *Neural Networks: Tricks of the Trade*, 2012.

[45] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[46] B. F. Klare, B. Klein, E. Taborsky, A. Blanton, J. Cheney, K. Allen, P. Grother, A. Mah, M. Burge, and A. K. Jain, "Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1931–1939, 2015.

[47] B. Maze, J. C. Adams, J. A. Duncan, N. D. Kalka, T. Miller, C. Otto, A. K. Jain, W. T. Niggel, J. Anderson, J. Cheney, and P. Grother, "Iarpa janus benchmark - c: Face dataset and protocol," *2018 International Conference on Biometrics (ICB)*, pp. 158–165, 2018.

[48] H. Li, G. Hua, X. Shen, Z. Lin, and J. Brandt, "Eigen-pep for video face recognition," *Computer Vision-ACCV*, pp. 17–33, 2014.

[49] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," *ICCV*, pp. 1701–1708, 2014.

[50] Y. Sun, X. Wang, and X. Tang, "Deeply learned face representations are sparse, selective, and robust," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2892–2900, 2015.

VITA

VITA

Tongyang Liu is currently a PhD candidate working under Dr. Qian Lin and Professor Jan P. Allebach in the School of Electrical and Computer Engineering at Purdue University. His research is focused on deep learning, computer vision, color image processing, imaging and printing. He obtained his bachelor's degree from University of Science and Technology of China (2014).