

**THREE PROBLEMS IN IMAGE ANALYSIS AND  
PROCESSING: DETERMINING OPTIMAL RESOLUTION  
FOR SCANNED DOCUMENT RASTER CONTENT, PAGE  
ORIENTATION, AND COLOR TABLE COMPRESSION**

by

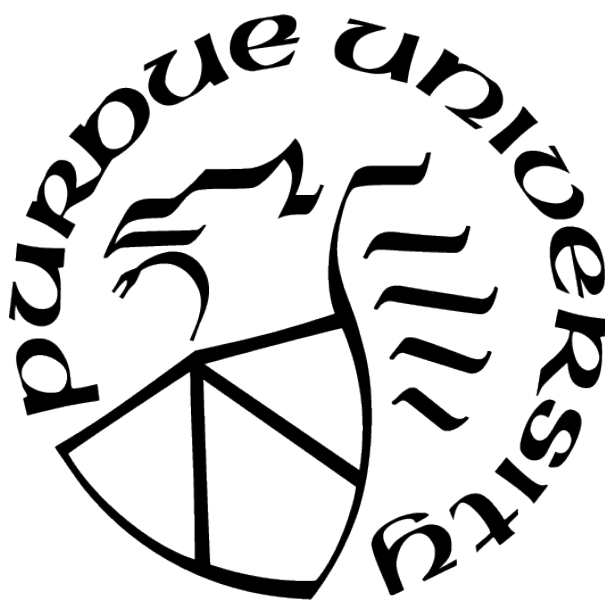
**Zhenhua Hu**

**A Dissertation**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Doctor of Philosophy**



School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL  
STATEMENT OF COMMITTEE APPROVAL**

**Dr. Jan Allebach, Chair**

School of Electrical and Computer Engineering

**Dr. Amy Reibman**

School of Electrical and Computer Engineering

**Dr. Mary Comer**

School of Electrical and Computer Engineering

**Dr. Fengqing Zhu**

School of Electrical and Computer Engineering

**Approved by:**

Dr. Vijay Raghunathan

To my parents, and my wife, Jiayue Cao.

## ACKNOWLEDGMENTS

I want to express my greatest gratitude to my major advisor Prof. Allebach, who has been giving me valuable guidance and advice on my way to pursue my PhD degree. I joined his group in the summer of 2016, and transferred to a Ph.D. student in early 2017. He has made a great impact on both my study and my personal life. I would not have been able to finish my Ph.D. study without his support. I'm also deeply grateful to HP Inc., which has been funding all my research projects. I also greatly appreciate the help of our HP partners, Dr. Mark Shaw, Terry Nelson, Peter Bauer, and Todd Harris. Their real-world experience and knowledge have benefited me a lot in doing the projects. I also want to thank Prof. Amy Reibman, Prof. Mary Comer, and Prof. Fengqing Zhu for being on my committee. Finally, I want to thank for the companion of my wife, Jiayue Cao. It's such a blessing to have her support every day.

The images used in Section 2 of this dissertation are designed by Freepik. We thank the authors macrovector, brgfx, Omelapics, vectorpouch on <http://www.freepik.com>.

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	9
LIST OF FIGURES . . . . .	10
LIST OF SYMBOLS . . . . .	14
ABBREVIATIONS . . . . .	15
ABSTRACT . . . . .	17
1 INTRODUCTION . . . . .	19
1.1 Overview . . . . .	19
1.2 Determining Optimal Resolution for Scanned Document Raster Content . . . . .	20
1.3 Determining Scanned Page Orientation . . . . .	21
1.4 Color Table Compression . . . . .	22
1.5 Summary . . . . .	23
2 DETERMINING OPTIMAL RESOLUTION FOR SCANNED DOCUMENT RASTER CONTENT . . . . .	24
2.1 Introduction . . . . .	24
2.2 Related work . . . . .	27
2.3 Methodology . . . . .	29
2.3.1 Scanner Pipeline Simulation . . . . .	29
2.3.2 Overall Algorithm . . . . .	35
2.4 Feature Vector Extraction . . . . .	35

2.4.1	Tile-STDDEV SSIM . . . . .	36
2.4.2	Sample Power Spectrum MSE . . . . .	44
2.4.3	Spatial Activity . . . . .	46
2.4.4	Edge Density . . . . .	49
2.4.5	Edge Contrast . . . . .	54
2.4.6	Support Vector Machine . . . . .	57
2.5	Experiment Results . . . . .	57
2.5.1	Data Collection . . . . .	57
2.5.2	Test Results . . . . .	62
2.5.3	Light Weight Neural Networks . . . . .	64
2.6	Conclusion . . . . .	69
2.7	Major Contributions . . . . .	70
3	DETERMINING SCANNED PAGE ORIENTATION . . . . .	72
3.1	Introduction . . . . .	72
3.2	Related work . . . . .	74
3.3	Methodology . . . . .	75
3.3.1	Overall Algorithm . . . . .	75
3.3.2	Image Binarization . . . . .	76
3.3.3	Connected Components Labeling . . . . .	78
3.3.4	Connected Components Analysis . . . . .	79

Size Limit . . . . .	80
Density Limit . . . . .	80
Transition Limit . . . . .	81
Isolate CCs . . . . .	82
Aspect Ratio . . . . .	83
CC Analysis Hierarchy . . . . .	83
CC Analysis Result . . . . .	84
3.4 Feature Vector Extraction . . . . .	86
3.4.1 Vertical Document Vector . . . . .	86
3.4.2 Horizontal Document Vector . . . . .	90
3.4.3 Zonal Density Vector . . . . .	91
3.4.4 Profile Component Run . . . . .	93
3.4.5 Support Vector Machines . . . . .	96
3.4.6 Script Detection Hierarchy . . . . .	96
3.5 Experimental Results . . . . .	97
3.5.1 Data Collection . . . . .	97
3.5.2 Test Results . . . . .	99
3.6 Low-resolution Connected Components Labeling Method . . . . .	102
3.7 Major Contributions . . . . .	104
4 COLOR TABLE COMPRESSION . . . . .	106

4.1	Introduction . . . . .	106
4.2	Methodology . . . . .	107
4.2.1	Coefficients Bit Assignment Table and Residues Bit Assignment Table	109
4.2.2	Determine the Step Size $\Delta$ . . . . .	111
4.3	Decoding Process . . . . .	113
4.4	Results . . . . .	114
4.5	Conclusion . . . . .	115
4.6	Major Contributions . . . . .	115
	REFERENCES . . . . .	117
	VITA . . . . .	125

## LIST OF TABLES

2.1	Scan resolutions and their corresponding tile sizes . . . . .	38
2.2	Tile-STDDEV SSIM values of different resolution pairs of the image shown in Figure 2.10. . . . .	41
2.3	Scan resolutions and their corresponding window sizes . . . . .	56
2.4	Scan Resolution and the Number of Sub-images . . . . .	62
2.5	Training and validation results. . . . .	63
2.6	The rank of importance for features. . . . .	63
2.7	Comparison of SVM and light-weight CNNs with different baseline networks. . .	66
2.8	Latency of light-weight CNNs on Raspberry Pi. . . . .	67
2.9	Different quantized models of MobileNetV2. . . . .	69
3.1	All scripts and the number of pages . . . . .	98
3.2	Roman script and the number of pages . . . . .	98
3.3	Training prediction accuracy of script and orientation for all scripts. . . . .	99
3.4	Training prediction accuracy of script and orientation for Roman scripts. . . .	100
3.5	Cross-validation result of script and orientation detection for all scripts . . . .	101
3.6	Cross-validation result of script and orientation detection for Roman scripts . .	102
3.7	Memory usage of lowres-label and original algorithms . . . . .	103
3.8	Training orientation detection results of lowres-label and original algorithms . .	104
4.1	Storage Needed for Compressed Files . . . . .	115

## LIST OF FIGURES

2.1	Example of two pages with different contents. We can see that picture (a) contains a plain blue area, while picture (b) has more detail. So we can see that they need different scan resolutions. . . . .	25
2.2	Size comparison of 300 dpi, 150 dpi, and 75 dpi scan resolutions. We can see that the 75 dpi image is half the size of the 150 dpi image, and 1/4 the size of the 300 dpi image. . . . .	26
2.3	Pipeline to determine the optimal scan resolution. It consists of two parts. (a) shows the way to generate all three resolution images, and (b) shows the rough pipeline to get the optimal resolution image from all three resolution images as the input. . . . .	29
2.4	75 dpi image enlarged by 400% in Nomacs and other image view software applications. The results of bilinear and nearest-neighbor interpolation show that most of them use the nearest-neighbor interpolation when zooming in. .	31
2.5	Comparison of PDF files viewed at 400%, and resized by 4× through bilinear and nearest-neighbor interpolation, all generated using Adobe Illustrator and viewed in Adobe Acrobat Reader. We can see that Adobe Acrobat Reader also uses nearest-neighbor interpolation. . . . .	32
2.6	Comparison of PDF files viewed at 400%, and resized by 4× through bilinear and nearest-neighbor interpolation, all generated using Adobe Photoshop and viewed in Adobe Acrobat Reader. We can see that Adobe Acrobat Reader also uses nearest-neighbor interpolation. . . . .	33
2.7	Comparison of PDF files viewed at 400%, and resized by 4× through bilinear, bicubic, and nearest-neighbor interpolation, all generated using Adobe Photoshop and viewed in Mac Preview. We can see that Mac Preview uses a different upsampling algorithm. . . . .	34
2.8	Training and testing pipeline of the proposed algorithm. . . . .	35
2.9	Steps to generate the STDDEV map. The above block shows the original image and the tile size of 3×3. After calculating the STDDEV of each tile, we form a new STDDEV map, which has the same size across all resolutions. . . . .	38
2.10	Original 300 dpi, 150 dpi and 75 dpi images; and their STDDEV maps. All STDDEV maps are enlarged for better view. We can see that the STDDEV maps keep most of the structural information of their original images. . . .	39
2.11	tile-STDDEV SSIM workflow. . . . .	40

2.12	Full tile-STDDEV SSIM map and the binary image of the STDDEV map. From (a) and (b) we can tell that the binary map provides a good mask to locate foreground pixels, so that background pixels will not affect the quality assessment. . . . .	41
2.13	tile-STDDEV SSIM STDDEV workflow. . . . .	42
2.14	Three example images. . . . .	43
2.15	Tile-STDDEV SSIM and Tile-STDDEV SSIM STDDEV results of example images in Figure 2.14 . . . . .	43
2.16	Changes of scanned character images from 300 dpi to 75 dpi. For a better viewing experience, the lower resolution images are resized to the same size as that of 300 dpi. . . . .	44
2.17	Workflow to calculate sample power spectrum MSE. . . . .	45
2.18	MSE of sample spectrum of example images. We can see that the values between the (300 dpi, 150 dpi) and the (300 dpi, 75 dpi) are decreasing. . . .	46
2.19	Workflow to calculate spatial activity. . . . .	47
2.20	Spatial activity of example images in Figure 2.14. We can see that the values between (300 dpi, 150 dpi) and (300 dpi, 75 dpi) are increasing. . . . .	48
2.21	Change of line drawing with decreasing resolutions. The 300 dpi image in (a) has very good details with distinct lines, while the 75 dpi image of (b) has very blurry edges. For better comparison, we enlarge the same area, the upper right corner, to be the same size. From the enlarged area, we can barely even see any edges in the 75 dpi image. . . . .	49
2.22	Workflow to calculate edge density. . . . .	50
2.23	An example of horizontal edge density. . . . .	50
2.24	An example of vertical edge density. . . . .	52
2.25	Edge density of example images in Figure 2.14. . . . .	53
2.26	Workflow to calculate edge contrast. . . . .	55
2.27	Edge contrast of example images in Figure 2.14. . . . .	56
2.28	Some of the example images in the dataset. . . . .	59
2.29	Demonstration of how sub-images are obtained. . . . .	59
2.30	Examples of blurriness and jaggedness. . . . .	60
2.31	An example of detail loss. We can clearly see three circles in the red oval in (a). But in the same area of (b), we are not very sure if they are circles. . .	61

2.32	Workflow to label optimal scan resolution. Each low-resolution (150 dpi and 75 dpi) image is resized to the same size as that of its 300 dpi reference image. The reference image and both of the resized images are opened in the same image viewing software. Finally, the optimal resolution for the input image is chosen. . . . .	62
2.33	SVM-RFE analysis result based on 5-fold cross validation. . . . .	64
2.34	Proposed CNN structure. The 300 dpi input image is passed into a baseline network with the last layer replaced by a FC layer. Then softmax outputs a probability that will decide the optimal scan resolution. . . . .	65
3.1	An example of one page in four orientations. . . . .	72
3.2	Examples of portrait and landscape pages. . . . .	73
3.3	Overall algorithm workflow. . . . .	75
3.4	Examples of gray-scale images and their binary images after applying Otsu's algorithm. (a)-(b) are the gray-scale images, and (c)-(d) are binary images generated by Otsu's algorithm . . . . .	77
3.5	8-connectivity. The middle pixel in the second row is the pixel under study. The 8-connectivity examines pixels in its West, North-West, North, and North-East pixels in connected components labeling. . . . .	78
3.6	An example of a CC with large foreground-background ratio. Here, the white pixels represent background, and the black pixels represent the foreground. We can see that the characters here are classified to be background. . . . .	81
3.7	Examples of CCs rendered with a halftone. We can see a lot of interleaved pixels in these CCs after binarization; and they cannot be identified by the size limits. . . . .	82
3.8	Examples of an isolated CC. We can see that there are no foreground pixels in one bounding box area to its left, right, upper, and lower directions. . . .	83
3.9	CC analysis hierarchy. . . . .	84
3.10	Examples of binary images and their final images after connected components analysis. (a)-(b) are the binary images after Otsu's algorithm, and (c)-(d) are final images generated after connected components analysis. . . . .	85
3.11	Examples of vertical transitions on connected components. The bounding boxes are separated into three equal zones vertically. The vertical dashed line in the middle is the imaginary line through the CC's center. And the arrows in the picture show locations of transitions. . . . .	86
3.12	Illustration of constructing the VDV from VCRs. Each row in the upper table shows VCRs of the character to the left side. The row in the bottom is the VDV, which is the the sum of all rows. . . . .	89

3.13	Examples of HCR and HDV of connected components. In the top row are three characters. Each of their bounding boxes is separated horizontally into three equal zones. And the horizontal line in the middle travels through centers of these characters. The table in the middle shows HCRs of the three characters in the top row. The last row is the HDV, which is the sum of all rows of the middle table. . . . .	91
3.14	Examples of ZDRs of connected components. Each of the character's bounding box in the top row is separated equally into three parts, and their pixel densities are listed in the table below the top row. . . . .	92
3.15	Example of a CC's four-side profiles. Here, the black pixels show the foreground, and the white pixels are the background. . . . .	94
3.16	Examples of PCR of a CC. The arrow lines in the figure show the distance from the bounding box edges to the foreground pixels in the four side profiles. . . . .	95
3.17	Script Detection Hierarchy. . . . .	97
3.18	The lowres-label algorithm workflow to obtain feature vector. . . . .	103
4.1	Compression-reconstruction process. . . . .	107
4.2	Workflow of the compression method. . . . .	108
4.3	Algorithm to count bits needed for a given DCT coefficient. . . . .	110
4.4	Process to calculate compression ratio for a given $\Delta$ . . . . .	112
4.5	Compression ratio curve with different step sizes. . . . .	113
4.6	Decoding process. . . . .	114

## LIST OF SYMBOLS

$l(x, y)$	luminance similarity of images x and y
$c(x, y)$	contrast similarity of images x and y
$s(x, y)$	structure similarity of images x and y
$\mu_x$	average of image x
$\sigma_x$	standard deviation of image x
$F(u)$	Fourier Transform
$I_b$	background intensity
$I_{avg}$	average intensity
$I_{max}$	maximum intensity
$I_{min}$	minimum intensity
$x_q$	quantized value
$x_f$	float value
$fd$	foreground pixel density
$Pl$	background pixel length from left bounding box edge
$Pr$	background pixel length from right bounding box edge
$Pt$	background pixel length from top bounding box edge
$Pb$	background pixel length from bottom bounding box edge
$\Delta$	step size

## ABBREVIATIONS

ADF	automatic document feeder
AIO	all in one
BRISQUE	blind/referenceless image spatial quality evaluator
CBAT	coefficient bit assignment table
CC	connected component
CNN	convolutional neural network
DCT	discrete cosine transform
DNN	deep neural network
DPI	dots per inch
DTFT	discrete-time Fourier transform
FC	fully-connected
FR	full reference
GGD	general Gaussian distribution
GSM	Gaussian scale mixture
HCR	horizontal component run
HDV	horizontal document vector
HVS	human visual system
IQA	image quality assessment
LSTM	long short-term memory
LUT	look-up table
LZMA	Lempel-Ziv-Markov chain algorithm
MSE	mean squared error
MTF	model transfer function
NAS	network architecture search
NIMA	Neural Image Assessment
NR	no reference
NSS	natural scene statistics
OCR	optical character recognition

PCR	profile component run
PDF	portable document format
PDV	profile document vector
PPI	pixel per inch
PSNR	peak signal-to-noise ratio
PTQ	post training quantization
QAT	quantization aware training
QE	quality estimators
RBAT	residue bit assignment table
RBF	radial basis function
ReLU	rectified linear unit
RFE	recursive feature elimination
RMS	root mean square
RR	reduced reference
SE	squeeze and excitation
SSIM	structural similarity index measure
STDDEV	standard deviation
SVM	support vector machine
VCR	vertical component run
VDV	vertical document vector
VIF	visual information fidelity
ZDV	zonal density vector
ZDR	zonal density run

# ABSTRACT

This thesis deals with three problems in image analysis and processing: determining optimal resolution for scanned document raster content, page orientation, and color table compression.

Determining optimal resolution for scanned document raster content aims to find an optimal scan resolution for different scan materials. Here the optimal scan resolution means the lowest resolution that keeps all the information of the scan materials. In this way we can save a lot of storage. In this study, the resolutions in question are 300 dpi, 150 dpi, and 75 dpi. We start with 300 dpi since this resolution would keep most scanned pages' information. 75 dpi is usually the smallest scan resolution that a printer has, and 150 dpi is the resolution in between. We developed an algorithm that extracts features and use SVM to find the optimal scan resolution. The features include tile standard deviation (STDDEV) structural similarity index measure mean (tile-STDDEV SSIM), tile STDDEV structural similarity index measure STDDEV (tile-STDDEV SSIM STDDEV), sample power spectrum MSE, and spatial activity, edge density, and edge contrast. These features can reflect the truthfulness between high-resolution images (references) and their low-resolution counterparts and the intrinsic changes from the high resolution to low resolutions. By feeding these feature into support vector machine (SVM) classifier, we can have a prediction accuracy of 93.4%.

Determining the scan page orientation can spare people from manually aligning printed pages before using a scanner. In this thesis, we propose an algorithm based on hand-crafted features and SVM. The features include vertical document vector (VDV), horizontal document vector (HDV), zonal density vector (ZDV) and profile document vector (PDV). Concatenating them together, we can have a feature vector for the document page. The feature vectors are then fed into SVM for training and predicting. This algorithm could work on multiple scripts, including Chinese, Devanagari, Japanese, Korean, Numeral, English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. In our algorithm we detect the script first, and then the orientation of the page. We also build a script detection hierarchy based on the structure similarities of different scripts. Experimental results show that the overall script accuracy is 98.2%, and the overall orientation accuracy for all scripts is 99.2%.

Color Management plays an important role in color reproduction and transformation of color information among various devices. Device profiles, such as Color look-up tables (LUTs), provide color management systems with the information necessary to convert color data between native device color spaces and device-independent color spaces. LUTs are often embedded in color documents to achieve color fidelity between different devices. The size of color tables will also increase with finer sampling of the spaces and larger bit depths. Thus, a method to compress LUTs is desirable for the purpose of conserving memory and storage, and also reducing network traffic and delay. In this dissertation, we propose a 1D color table lossless compression method based on discrete-time transformation (DCT). The compressed data consists of four files: the rounded quantized DCT coefficients for the color table, the residue table whose values are the difference of the original color tables and the initial reconstructed color tables, the coefficients bit assignment tables (CBAT) and the residue bit assignment tables (RBAT) that we proposed for quantized DCT coefficients and residue table, respectively. With these four files, we can perfectly reconstruct original 1D color tables. Experimental results show that we can achieve a compression ratio of around 3.05.

# 1. INTRODUCTION

## 1.1 Overview

Nowadays, scanners and printers are very common among people's everyday use. Although single function printers or scanners are enough for some users, most people still prefer multifunction peripheral (MFP) or all-in-one (AIO) printers, which can both scan documents and make photocopies. MFPs or AIOs can range from low-end home models to more powerful office printers. Home printers are built for light duty use. Office MFPs offer better text quality and faster speeds, with a lower cost of ink or toner. They are also equipped with automatic document feeders (ADFs), which makes it much faster to scan large quantity of pages all at once.

With the rise of the paperless office, the days of storing paper copies of documents in filing cabinets as the archival record of the document are ending. Scanning documents can free up the office space in a company by saving thousands of files on a single server. Besides, it is usually quicker to find a specific file in a computer than hunting through the dozens of file cabinets or mounds of paperwork that clutter the office. Moreover, it also provides a good way to back up the information written on ancient manuscripts. Further, rather than mailing a signed copy through USPS or FedEx, it is more commonplace to just scan and e-mail it or upload it to a cloud. All these make scanning really necessary in our everyday practice.

However, with the increasing demand for scanning, some problems need to be addressed. First, if using an ADF, current scan routines will scan all documents to the same resolution. If we use a high resolution like 300 dpi, the resulting images will occupy more memory than needed. Besides, it requires a lot of work to align documents to the right orientation before feeding them into an ADF. Also, there is very limited memory in a color printer; but the color tables used to do color conversions between color spaces may need a lot of memory. We also need to compress the color tables.

## 1.2 Determining Optimal Resolution for Scanned Document Raster Content

In current scanners, modern scan routines require a predefined scan resolution, whether it is customer-selected or a default value in the settings. When the scanning process begins, the resolution cannot be changed. This results in all scanned pages, no matter how much their contents may vary, having output images of the same size. For example, if one page has an image area of plain color and another image has many details, the two output images would have the same size after the scanning process. However, we can clearly see that fine-detailed images need a larger scan resolution than a plain page. If we scan both pages with a low resolution like 75 dpi, the scanned plain images would have all the information but the fine-detailed images will experience a lot of detail loss. However, if we scan them both at a high resolution like 300 dpi, the plain images would consume much more memory than needed. As a result, the best strategy for us would be to scan pages with different resolutions according to their contents, so that we can still maintain good image quality while lowering the total storage used.

In order to tackle this problem, different scan resolutions for different contents must be chosen. In this study, the resolutions in question are 300 dpi, 150 dpi, and 75 dpi. We start with 300 dpi since this resolution would keep most scanned pages' information. Undoubtedly, higher scan resolutions would keep more details and yield higher-quality images. The quality of scanned images degrades as the scan resolution decreases, which may cause blurriness. And this is quite obvious in scanned text documents, since blurriness can cause a document to be unreadable.

We developed an algorithm to automatically select the optimal scan resolution based on document contents. The algorithm is based on some hand-crafted features and the support vector machine (SVM) classifier. The features can be classified into two categories: full-reference (FR) features and no-reference (NR) features. The FR features, which include tile standard deviation (STDDEV) structural similarity index measure mean (tile-STDDEV SSIM), tile STDDEV structural similarity index measure STDDEV (tile-STDDEV SSIM STDDEV), sample power spectrum MSE, and spatial activity, reflect the truthfulness between high-resolution images (references) and their low-resolution counterparts. The NR

features exploit the changes of images from the high resolution to lower resolutions. The features include transition density, edge density, and edge contrast. Combining the features together, we can get a feature vector for each page. By feeding these feature vectors into a support vector machine (SVM) classifier, we can have a prediction accuracy of 93.4%.

### 1.3 Determining Scanned Page Orientation

With the trend of the paperless office, lots of documents are being scanned every day. Also, the need to preserve old ancient books or other materials requires a lot of scanning. When scanning, a page could be in one of the 4 orientations: 0, 90, 180, and 270 degrees. When there are large quantity of pages to scan, it is difficult or even impossible to know all pages' orientations while scanning. Manually aligning them could be tedious and very time-consuming. So it would great if the scanner could automatically detect their orientations. Moreover, it would be best if the scanner could detect the orientation of pages in various language scripts.

To determine one page's orientation, we developed an algorithm to find features of text characters, and predict its script and orientation using a support vector machine (SVM). The algorithm starts by turning a scanned color image into a grayscale image, and then partitions it into 16 sub-images with equal width and height. Then, Otsu's method [1] is applied to each sub-image to get 16 binary sub-image. Assembling them together, we can get a big binary image. Then we do connected components (CC) labeling using 8-neighborhood, followed by CC analysis to get text characters and reject non-text CCs. The features are calculated on text CCs only. The features include vertical document vector (VDV), horizontal document vector (HDV), zonal density vector (ZDV) and profile document vector (PDV). These features can reflect different information of a connected component (CC). Concatenating them together, we can have a feature vector for the document page. The feature vectors are then fed into an SVM for training and prediction. This algorithm can work on multiple scripts, including Chinese, Devanagari, Japanese, Korean, Numeral, English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. In our algorithm we propose to first detect the script, and then the orientation of the page. We also build a script

detection hierarchy based on the structural similarities of different scripts. Experimental results show that the overall script accuracy is *98.2%*, and the overall orientation accuracy for all scripts is *99.2%*.

#### 1.4 Color Table Compression

Color Management plays an important role in color reproduction and transformation of color information among various devices. Device profiles, such as Color look-up tables (LUTs), provide color management systems with the information necessary to convert color data between native device color spaces and device-independent color spaces. LUTs are often embedded in color documents to achieve color fidelity between different devices, which increases the total size of these documents. The size of color tables will also increase with finer sampling of the spaces and larger bit depths. Thus, a method to compress LUTs is desirable for the purpose of conserving memory and storage, and also reducing network traffic and delay.

Our work is based on the work of Chuohao Tang et al. [2] on lossy compression of 3D and 4D color tables through the discrete cosine transform (DCT). Their algorithm used a specific lossy method, followed by a general lossless method to compress the color tables. The specific lossy method was tailored to the specific characteristics of the color tables to be compressed. The general method was generic, and could be used to compress a variety of different types of data streams.

While their work does a good job in compressing 3D and 4D LUTs, they didn't try to do a lossless compression. So here we propose a lossless compression method on 1D LUTs. The compressed data consists of four files: quantized DCT coefficients from color tables, quantized DCT Coefficients Bit Assignment Tables (CBAT) , a residue table whose values are the difference between the original color tables and the initial reconstructed color tables, and the residue table bit assignment tables (RBAT). With these four files, we can perfectly reconstruct original 1D color tables. Experimental results show that we can achieve a maximum compression ratio of 3.05.

## 1.5 Summary

This dissertation deals with three problems in image analysis and processing: determining optimal resolution for scanned document raster content, page orientation, and color table compression. These are three important topics in modern MFPs, with the first two aiming at scanners, and the last one focusing on printers. Determining the optimal scan resolution can help reduce the total memory used to store all scanned images while keeping all the information. Determining the scanned page orientation automatically can free the manpower that would be needed to align all the pages before scanning. Further, since color tables are vastly used in printers to do conversion between color spaces, compressing them can save a lot of storage.

The dissertation is organized as follows:

- Chapter 2 proposes an algorithm to determine the optimal scan resolution based on the document's raster content.
- Chapter 3 develops a method to determine the orientations of scanned pages of various scripts.
- Chapter 4 introduces a method of lossless compression of color tables.
- Chapter 5 talks about my major contributions in the work described in this dissertation.

## 2. DETERMINING OPTIMAL RESOLUTION FOR SCANNED DOCUMENT RASTER CONTENT

### 2.1 Introduction

Modern scan routines require a predefined scan resolution, whether it is customer-selected or a default in the scanner's settings. When the scanning process begins, the resolution cannot be changed. This results in all scanned pages, no matter how much their contents may vary, having output images of the same size. For example, if one page has an image area of plain color and another image has many details, as shown in Fig. 2.1, the two images will have the same size after the scanning process. However, we can clearly see in Fig. 2.1 that picture (b) needs a larger scan resolution than picture (a). If we scan both pages with a low resolution like 75 dpi, picture (a) will have all the information but picture (b) will lose some details. However, if we scan them both at a high resolution like 300 dpi, picture (b) will have all the details, but we also need more space to store picture (a). As a result, the best strategy for us would be to scan images with different resolutions according to their contents, so that we can still maintain a good image quality while lowering the total used storage.

In order to tackle this problem, different scan resolutions for different contents must be chosen. Undoubtedly, higher scan resolutions would keep more details and yield scanned images with higher quality. The quality of scanned images degrades as the scan resolution decreases, which may cause blurriness and jaggedness. And this is quite obvious in scanned text documents, since blurriness can cause a document to be unreadable.

Scan resolution is measured in DPI or dpi, which is short for dots per inch. In printing, it is used to measure the number of dots that can be placed in a line whose length is 1 inch. Usually, the bigger the dpi is, the more details can be printed on the page [3]. However, monitors don't have dots, but do have digital pixels. As a result, some may prefer to use PPI or ppi, short for pixels per inch, when referring to image resolutions. But the use of DPI or PPI is often equivalent to each other. If we use dpi in scanning, one 'dot' actually equals to one digital image pixel [4] in a monitor. In this thesis, we choose to use dpi when referring to scan resolutions.



(a) flat image



(b) cat image

**Figure 2.1.** Example of two pages with different contents. We can see that picture (a) contains a plain blue area, while picture (b) has more detail. So we can see that they need different scan resolutions.

When choosing different scan resolutions like 300 dpi, 150 dpi, and 75 dpi, the resulting scanned images will have different sizes. Suppose we want to scan a page of size 8.5 by 11 inches at 300 dpi, the result digital image would have a size of 2550 ( $8.5 \times 300$ ) by 3300 ( $11 \times 300$ ) pixels. If the scan resolution is 150 dpi, the image size would be 1275 ( $8.5 \times 150$ ) by 1650 ( $11 \times 150$ ) pixels, half of the size of the 300 dpi image. And the image size using a scan resolution of 75 dpi would be 637 ( $8.5 \times 75$ ) by 825 ( $11 \times 75$ ), half the size of the 150 dpi image, and 1/4 the size of the 300 dpi image. The size differences are be seen in Figure 2.2. As a result, for raw images using the same bit depth, the 150 dpi image occupies 1/4 of the memory used to store its 300 dpi counterpart, while the 75 dpi image requires only 1/16 of 300 dpi image's memory. As we can see from Figure 2.2, the big size difference makes it very difficult to compare the image quality among different scan resolutions directly. Further, the zooming habits of people when viewing large or small images on screens make the problem more complicated.



**Figure 2.2.** Size comparison of 300 dpi, 150 dpi, and 75 dpi scan resolutions. We can see that the 75 dpi image is half the size of the 150 dpi image, and  $1/4$  the size of the 300 dpi image.

We propose a traditional machine learning based method to find the optimal lowest scan resolution for each scan document. We collected a large number of documents, and scanned them into 300 dpi, which are used as benchmarks for good image quality. The images are stored as 'TIFF' without any compression. Then we use an area-based down-sampling method to get images of 150 and 75 dpi. To compare the image quality, we open images of all resolutions in the same image-reading software, and view them at 100% to determine the lowest acceptable scan resolution. With all these data, we extracted some hand-crafted features and fed them into a support vector machine (SVM) to do training and testing. The features are: tile standard deviation (STDDEV) structural similarity index measure mean (tile-STDDEV SSIM), tile STDDEV structural similarity index measure STDDEV

(tile-STDDEV SSIM STDDEV), sample power spectrum mean squared error (MSE), spatial activity, transition density, edge density, and edge contrast. Combining the features together, we can get a feature vector for each page. By feeding these feature vectors into a support vector machine (SVM) classifier, we can have a prediction accuracy of 93.4%.

## 2.2 Related work

Finding the optimal scan resolution is to find the smallest scan resolution with acceptable image quality. Image quality assessment (IQA) is a widely-researched topic [5][6]. Typically, there are three categories of quality estimators (QEs): full reference (FR) QEs, reduced reference (RR) QEs, and no reference (NR) QEs. FR QEs exploit the “perfect” reference image and examine the truthfulness that a new image has with the reference image. Maybe the simplest measure is the mean squared error (MSE) or peak signal-to-noise ratio (PSNR). But its nature of heavily relying on pixel correspondences cannot reflect a human’s subjective view. Other QEs, like structural similarity index (SSIM) [7][8], multi-scale structural similarity [9], visual information fidelity (VIF) [10], most apparent distortion (MAD) [11], are better to reflect a human’s perceptual assessment. With the rise of deep learning research, perceptual features, or feature maps of deep convolutional neural networks [12] are widely used and are considered to resemble the human visual system (HVS) much better. But they all require a reference image of the same resolution, size and good pixel correspondence. There are also some FR QEs that don’t require the same size of the test and reference images. The Multiscale Image Quality Estimator (MIQE) [13] was developed to assess images of different resolutions from various viewing devices like laptops, televisions, and smart phones. It is based on the wavelet transform and mutual information to assess the quality of a distorted image with a lower resolution compared with the reference image with a higher resolution. A cross spatial resolution image QE [14] is proposed to estimate the quality of interpolated high-resolution images compared to their low-resolution counterparts. The QE is based on a natural scene statistics (NSS) framework, which contains frequency energy falloff, dominant orientation, and spatial continuity statistics.

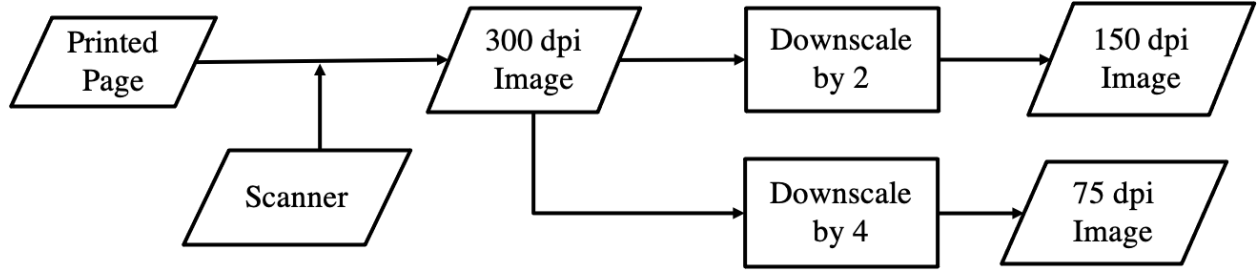
Reduced reference (RR) approaches are used if the reference image is available, but is too costly to get. Most RR QEs are based on NSSs models [15]. For example, the “divisive normalization” QE [16] uses the Gaussian scale mixture (GSM) model on the wavelet coefficients of the reference image and the image to be tested. Also, the “quality-aware image” QE [17] exploits information in a general Gaussian distribution (GGD) model. But since some of the images under study in our work like line drawings are not natural scenes, RR QEs are not suitable for our study.

When there is not any information from a reference image, IQA must be processed on the test image only. This is referred to as the no reference (NR) IQA. NR QEs exploit properties of test images directly, like the blocking from JPEGs [18][19], ringing from JPEG 2000 [20][21] and blurring [22][23]. Some QEs exploit the NSS properties like BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) [24]. There are also some training based NR IQAs that learn the human judgements on a large database of human opinion scores [10]. Some features are specifically designed to assess text-related image quality. These features include stroke line width [25], edge raggedness [26][27], small speckles [28][29], sharpness [30][31], or features based on deep neural networks [32]. Although our problem has reference images, they cannot be used directly as in FR QEs. As a result, we can use some NR QEs to help assess the image quality.

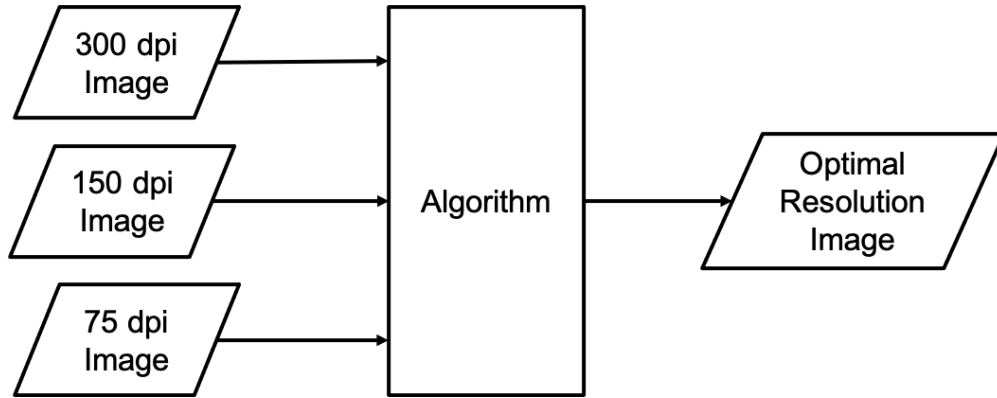
In this thesis, we propose some QEs that can help perform IQA of images at different resolutions. Although the reference images cannot be used directly as in FR IQA, our QEs can use their information to help assess qualities changes among different scan resolutions. We also explore NR QEs since they can reflect image qualities without reference images. Further, the image resolutions in consideration are 300 dpi, 150 dpi, and 75 dpi. We chose these resolutions because after some research and investigation, we found that 300 dpi can preserve image qualities that are good enough for most scanned materials. Besides, most scanners from the high-end to low-end all support these resolutions. Tests on various kinds of images show that these QEs can achieve a prediction accuracy at 93.4% through SVM.

## 2.3 Methodology

### 2.3.1 Scanner Pipeline Simulation



(a) Scanner pipeline to generate all resolution images



(b) Algorithm pipeline

**Figure 2.3.** Pipeline to determine the optimal scan resolution. It consists of two parts. (a) shows the way to generate all three resolution images, and (b) shows the rough pipeline to get the optimal resolution image from all three resolution images as the input.

Figure 2.3 shows the overall scanner pipeline for this problem. In reality, to save the scanning time, the scanner only scans the printed page into 300 dpi, and then we do downscaling to get 150 and 75 dpi images. Then, the three images will be the input to the algorithm to get the optimal resolution image.

When doing downscaling, after carefully studying the scanning process, we chose to use the area-based downscaling method or box sampling method. It subsamples an image using pixel area relations. That is, if we want to downscale a 300 dpi image to the size of 150 dpi, we would use a  $2 \times 2$  box and calculate the box average as the value of the corresponding

output pixel. When downscaling the image to 75 dpi, the average is calculated in a  $4 \times 4$  window. This corresponds well to the scanning process [33].

Since the images are finally viewed by people on a computer, we also need to simulate the real viewing situation after the digital images are generated. Ideally, people would choose to use an image viewing software application, or a PDF (Portable Document Format) reader if the scanned result is a PDF file. For small images, people tend to enlarge them for better viewing. Since we consider 300 dpi images to be the “perfect” reference, and the 150 and 75 dpi results are smaller than them, we need to resize 75 dpi and 150 dpi images to the same size as that of the 300 dpi file. There are many image resizing algorithms, like nearest-neighbor interpolation, bilinear interpolation, trilinear interpolation. Their results can be very different. So the upsampling methods used in these software applications need to be studied before we resize the low-resolution images. To solve this problem, we investigated several commonly-used image and pdf viewing software applications and found what upsampling method they use.



(a) 75 dpi image viewed at 400% in Nomacs [34]



(b) 75 dpi upscaled by 4 $\times$  through bilinear interpolation



(c) 75 dpi upscaled by 4 $\times$  through nearest-neighbor interpolation



(d) 75 dpi image viewed at 400% in Windows Photo Viewer



(e) 75 dpi image viewed at 400% in PhotoShop



(f) 75 dpi image viewed at 400% in IrfanViewer [35]

**Figure 2.4.** 75 dpi image enlarged by 400% in Nomacs and other image view software applications. The results of bilinear and nearest-neighbor interpolation show that most of them use the nearest-neighbor interpolation when zooming in.

Figure 2.4 shows an image viewed in some common image view software like Nomacs [34], Windows Photo Viewer, Photoshop, and IrfanViewer [35]. The image is 75 dpi and all are zoomed in by 400%. We choose 75 dpi over 150 dpi for better view and comparison purposes. The original image is captured through area-based downsampling from a scanned 300 dpi image, and saved as an uncompressed 'TIFF' file. We can see that except for IrfanViewer, the other software's results all look very similar to each other. For better comparison, we also use code to resize the image by 400% using bilinear interpolation and nearest-neighbor interpolation. The code-generated images are saved in uncompressed 'TIFF', and viewed in Nomacs at 100%. Figures 2.4(b) and 2.4(c) are the results of bilinear interpolation and

nearest-neighbor interpolation, respectively. We can see that most image view software applications are using nearest-neighbor interpolation while zooming in images. IrfanViewer, however, actually uses Lanczos' method [36] followed by a filter to generate a result better than bilinear and nearest-neighbor interpolation.

Besides image files, it is also common for people to save scanned digital files as PDF files. So we also do the research on what upsampling algorithm PDF viewers are using. Compared with the many image viewing software used, there are mostly two major PDF viewers: Adobe Acrobat reader, and Preview on Mac. In order to best simulate the real situation, we generate the PDF files from 75 dpi images (as we don't scan in 75 dpi directly) using two major software applications: Adobe Illustrator and Adobe Photoshop using their default settings.

Figure 2.5 shows the results of PDF files zoomed in by 400%, bilinear-upscaled by  $4\times$ , and nearest-neighbor upscaled by  $4\times$ , all viewed in Adobe Acrobat Reader. The PDF files are all generated using Adobe Illustrator from the 75 dpi image in Figure 2.4. Comparing Figure 2.5(a) with Figure 2.5(c), we can see that the default zooming-in algorithm used in Adobe Acrobat Reader is also nearest-neighbor interpolation.



(a) 75 dpi image viewed at 400% in Adobe Acrobat Reader      (b) 75 dpi upscaled by  $4\times$  through bilinear interpolation      (c) 75 dpi upscaled by  $4\times$  through nearest-neighbor interpolation

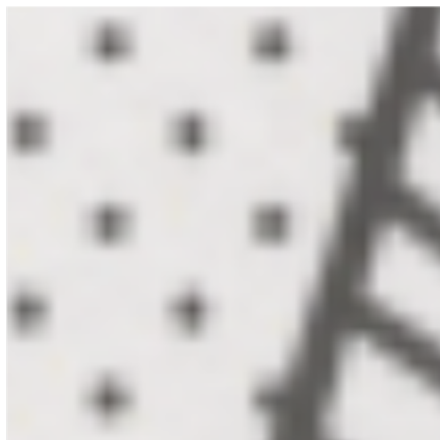
**Figure 2.5.** Comparison of PDF files viewed at 400%, and resized by  $4\times$  through bilinear and nearest-neighbor interpolation, all generated using Adobe Illustrator and viewed in Adobe Acrobat Reader. We can see that Adobe Acrobat Reader also uses nearest-neighbor interpolation.



(a) 75 dpi image viewed at 400% in Adobe Acrobat Reader      (b) 75 dpi upscaled by 4× through bilinear interpolation      (c) 75 dpi upscaled by 4× through nearest-neighbor interpolation

**Figure 2.6.** Comparison of PDF files viewed at 400%, and resized by 4× through bilinear and nearest-neighbor interpolation, all generated using Adobe Photoshop and viewed in Adobe Acrobat Reader. We can see that Adobe Acrobat Reader also uses nearest-neighbor interpolation.

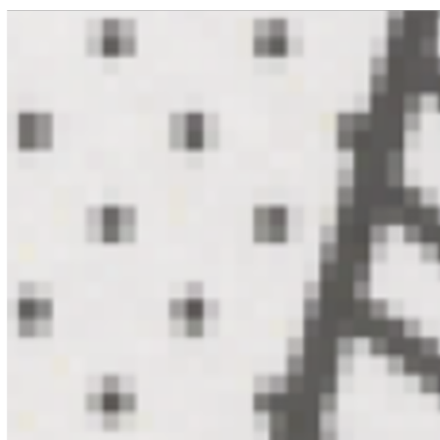
Figure 2.6 shows the results of PDF files zoomed in by 400%, bilinear-upsampled by 4×, and nearest-neighbor upscaled by 4×, all viewed in Adobe Acrobat Reader. Unlike Figure 2.5, the PDF files here are all generated using Adobe Photoshop from the 75 dpi image in Figure 2.4. Comparing Figure 2.6(a) with Figure 2.6(c) we can see that the default zooming-in algorithm used in Adobe Acrobat Reader is also the nearest-neighbor interpolation method.



(a) 75 dpi image viewed at 400% in Mac Preview



(b) 75 dpi upscaled by 4 $\times$  through bilinear interpolation



(c) 75 dpi upscaled by 4 $\times$  through nearest-neighbor interpolation



(d) 75 dpi upscaled by 4 $\times$  through bicubic interpolation

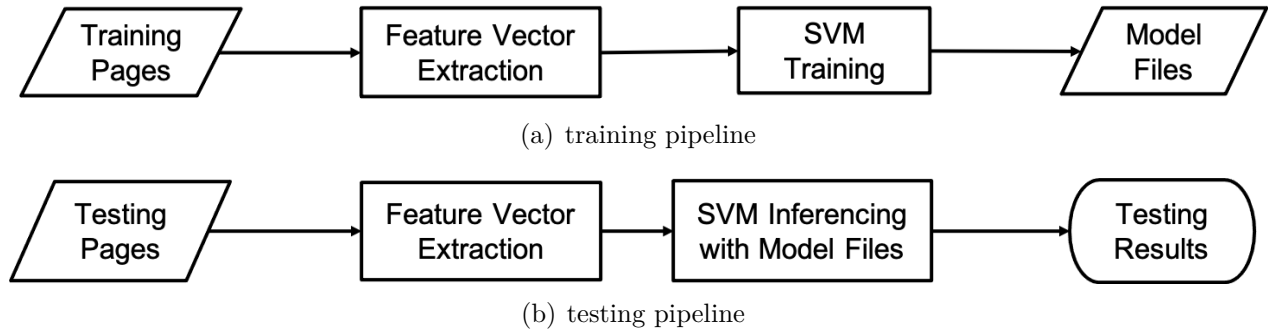
**Figure 2.7.** Comparison of PDF files viewed at 400%, and resized by 4 $\times$  through bilinear, bicubic, and nearest-neighbor interpolation, all generated using Adobe Photoshop and viewed in Mac Preview. We can see that Mac Preview uses a different upsampling algorithm.

Figure 2.7 shows the results of PDF files zoomed in by 400%, bilinear-upsampled by 4 $\times$ , bicubic-upsampled by 4 $\times$ , and nearest-neighbor upsampled by 4 $\times$ , all viewed using Mac Preview. The PDF files here are all generated using Adobe Photoshop from the 75 dpi image in Figure 2.4. We don't test the PDF files generated from Adobe Illustrator since they appear the same from Figure 2.5 and Figure 2.6. Comparing Figure 2.7(a) with

Figure 2.7(c), Figure 2.7(b), and Figure 2.7(d), we can see that the default zooming-in algorithm used in Mac Preview is not bilinear, bicubic, or nearest-neighbor interpolation. It is more like bilinear interpolation followed by low-pass filtering.

From Figure 2.4 to Figure 2.7, we can see that although Mac Preview uses a different up-sampling method, most of the image and PDF viewing software applications' default method when zooming in is nearest-neighbor interpolation. As a result, we will use nearest-neighbor interpolation when we need to resize low-resolution images to their 300 dpi counterparts.

### 2.3.2 Overall Algorithm



**Figure 2.8.** Training and testing pipeline of the proposed algorithm.

The algorithm workflow is shown in Fig. 2.8. In the dataset we have training and testing pages. We extract feature vectors from training pages, and feed the feature vector into the SVM to get model files. In the testing phase, the testing images would go through the same feature vector extraction process. The feature vectors are then passed into the SVM classifier. Using the trained model files, we can finally get the desired scan resolution for the testing pages.

## 2.4 Feature Vector Extraction

This section mainly talks about the features we choose, and the detailed process to get them. These features can be classified into two categories: no-reference and full-reference. The no-reference features mainly exploit the intrinsic features of the scanned images themselves. They include: edge density and edge contrast. By comparing the values at all

resolutions (300 dpi, 150 dpi, and 75 dpi), we can track the changes of these features, which can reflect the image quality changes. For the full-reference features, although the reference images (300 dpi) are much bigger than images at other resolutions, we can still use some method, like partitioning images into the same number of windows and using one value to represent each window, or resizing the low-resolution images into the same size as that of the 300 dpi images. The full-reference features include tile-STDDEV SSIM, tile-STDDEV SSIM STDDEV, sample power spectrum mean squared error (MSE), and spatial activity. These features are then concatenated into a single feature vector, from which we will use SVM to do training and testing.

#### 2.4.1 Tile-STDDEV SSIM

SSIM (structural similarity index) [7] is the probably the most successful FR QE, and is widely used in estimating the quality of images. It is composed of three elements over small patches of two images  $x$ ,  $y$ : a luminance similarity term  $l(x, y)$ , a contrast similarity term  $c(x, y)$ , and a structure similarity term  $s(x, y)$ . Their computations are [7]:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \quad (2.1)$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad (2.2)$$

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \quad (2.3)$$

Here  $\mu_x$ ,  $\mu_y$  are averages of images  $x$  and  $y$ , respectively.  $\sigma_x$ ,  $\sigma_y$  are standard deviations of  $x$  and  $y$ , respectively. All these statistics are computed within a local window at the center pixel. Thus, the elements computed in Equations 2.1 - 2.3 and the final SSIM computed in Equation 2.4 are maps of the same size as the images  $x$  and  $y$ . Here,  $\sigma_{xy}$  is the covariance of

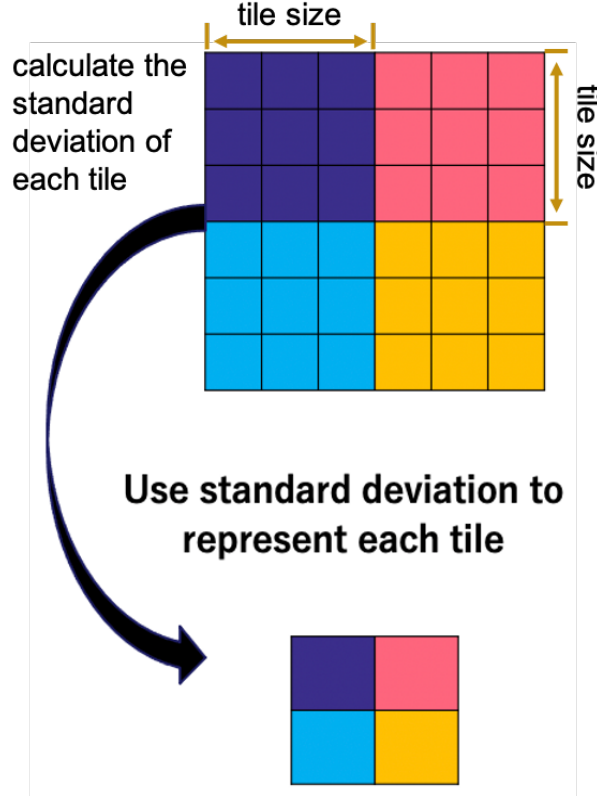
$x$  and  $y$ . And  $c_1$ ,  $c_2$  and  $c_3$  are variables to stabilize the division with a weak denominator. The formula to calculate the SSIM value is

$$SSIM(x, y) = l(x, y) \cdot c(x, y) \cdot s(x, y) \quad (2.4)$$

SSIM is very powerful in predicting the structural similarity of a test image to a reference image. After some research and investigation, we find that the majority of images have good perceptual quality at 300 dpi. So we can use 300 dpi images as the reference to assess qualities of lower resolution (150 and 75 dpi) images. But images of different resolutions have different sizes, which cannot be assessed using SSIM directly. As a result, we need to do some pre-processing steps before SSIM can be applied.

We noticed that for images of different resolutions, the ratio of their sizes corresponds to the ratio of their resolutions. For example, the 300 dpi image's width/height is twice the width/height of the 150 dpi images, the same as 300/150. The 300 dpi image's width/height is four times the width/height of the 75 dpi images, which is the ratio of 300 over 75. As a result, we can generate images of the same size by utilizing these ratios. Then we can apply the SSIM to assess quality changes across different resolutions.

We propose a QE called tile-STDDEV SSIM. According to the size ratios, we partition different resolution images into the same number of non-overlapping tiles. In each tile, we calculate the standard deviation (STDDEV) of it. And then we combine all the values to form a STDDEV map, as is shown in Figure 2.9. We choose STDDEV to represent each tile because low resolution images (150 and 75 dpi) can be approximated by area-based downsampling from high resolution (300 dpi) images. The number of gray levels decreases in this process, which can result in a decrease of the activity in the corresponding tiles. At the end of the preprocessing, since all images, whatever their resolutions are, have the same number of tiles, their STDDEV maps must be the same size.



**Figure 2.9.** Steps to generate the STDDEV map. The above block shows the original image and the tile size of  $3 \times 3$ . After calculating the STDDEV of each tile, we form a new STDDEV map, which has the same size across all resolutions.

**Table 2.1.** Scan resolutions and their corresponding tile sizes

Resolution (dpi)	Tile size (pixel number)
300	12
150	6
75	3

The image resolutions and their corresponding tile sizes can be found in Table 2.1. The tile sizes are proportional to their resolutions so that the images have the same number of tiles regardless of their resolutions.



(a) 300 dpi image



(b) 150 dpi image



(c) 75 dpi image



(d) 300 dpi STD-  
DEV map



(e) 150 dpi STD-  
DEV map

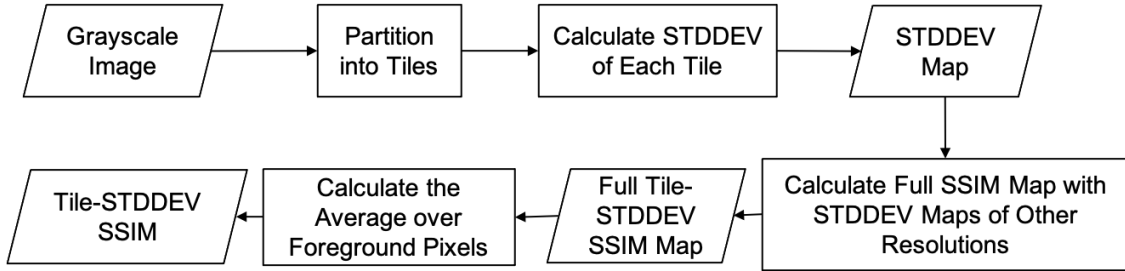


(f) 75 dpi STD-  
DEV map

**Figure 2.10.** Original 300 dpi, 150 dpi and 75 dpi images; and their STDDEV maps. All STDDEV maps are enlarged for better view. We can see that the STDDEV maps keep most of the structural information of their original images.

Figure 2.10 shows the original 300 dpi, 150 dpi, and 75 dpi images; and their STDDEV maps, respectively. We can see that although the original images are of different sizes, their STDDEV maps have the same size, and keep most of the structural information.

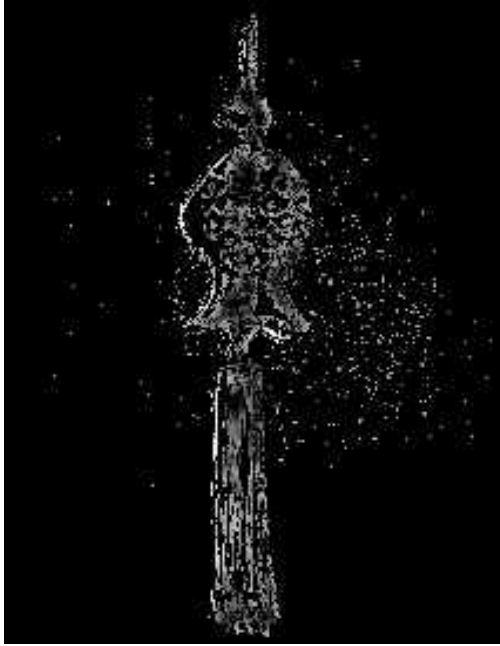
Figure 2.11 shows the workflow of tile-STDDEV SSIM. We have a grayscale image as the input. If an image is a color image, we will turn it to grayscale before passing it into the workflow. Then, the input image are separated into non-overlapping tiles according to their resolutions (Table 2.1), so that the number of tiles will be the same across all resolutions. In each tile, we calculate its STDDEV value. In this way, we have STDDEV maps of the same size. Then, we can calculate the SSIM values between low-resolution STDDEV map and the high resolution STDDEV map, namely the (300 dpi, 150 dpi) and (300 dpi, 75 dpi) image pairs. Note that we are computing the SSIM between the STDDEV maps of the two images, not the pixel gray values of the two images.



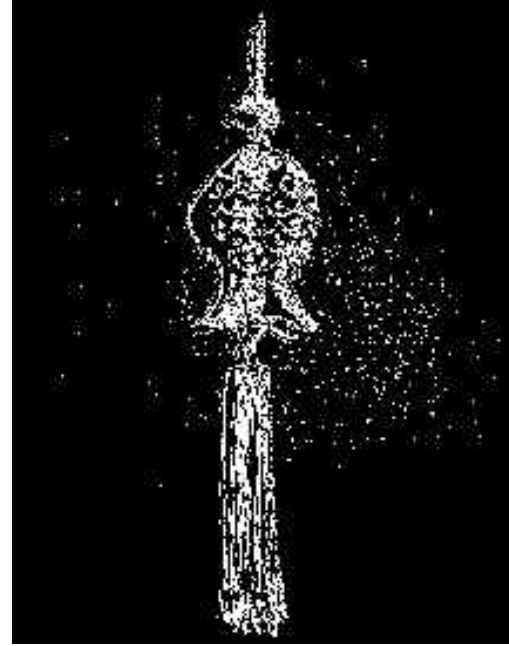
**Figure 2.11.** tile-STDDEV SSIM workflow.

On the STDDEV maps, the SSIM values are calculated on  $7 \times 7$  image patches with the stride equal to 1 pixel to get the full SSIM map. Traditionally, we use its average to assess the structural similarities of two images. But when people look at an image, we hypothesize that they are mainly focused on the foreground areas. The background areas, like the large white area in a document image, or the plain flat areas in a natural scene image, usually attract much less attention. As a result, it would not be reasonable to calculate the average over all these foreground areas.

In order to find the foreground pixels, we binarize the 300 dpi STDDEV map through the Otsu's method [1], as is shown in Figure 2.12. By averaging over foreground pixels only, we can have the tile-STDDEV SSIM. With the decreasing scan resolution, we should expect a continuous drop in tile-STDDEV SSIM values from the (300 dpi, 150 dpi) to the (300 dpi, 75 dpi) pairs.



(a) Full tile STDDEV SSIM map between 300 and 150 dpi images



(b) Binary STDDEV image mask

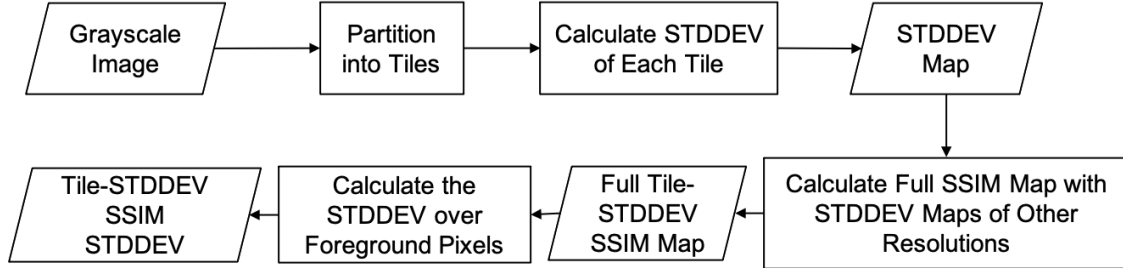
**Figure 2.12.** Full tile-STDDEV SSIM map and the binary image of the STDDEV map. From (a) and (b) we can tell that the binary map provides a good mask to locate foreground pixels, so that background pixels will not affect the quality assessment.

**Table 2.2.** Tile-STDDEV SSIM values of different resolution pairs of the image shown in Figure 2.10.

Resolution pair	Tile-STDDEV SSIM
(300 dpi, 150 dpi)	0.71
(300 dpi, 75 dpi)	0.19

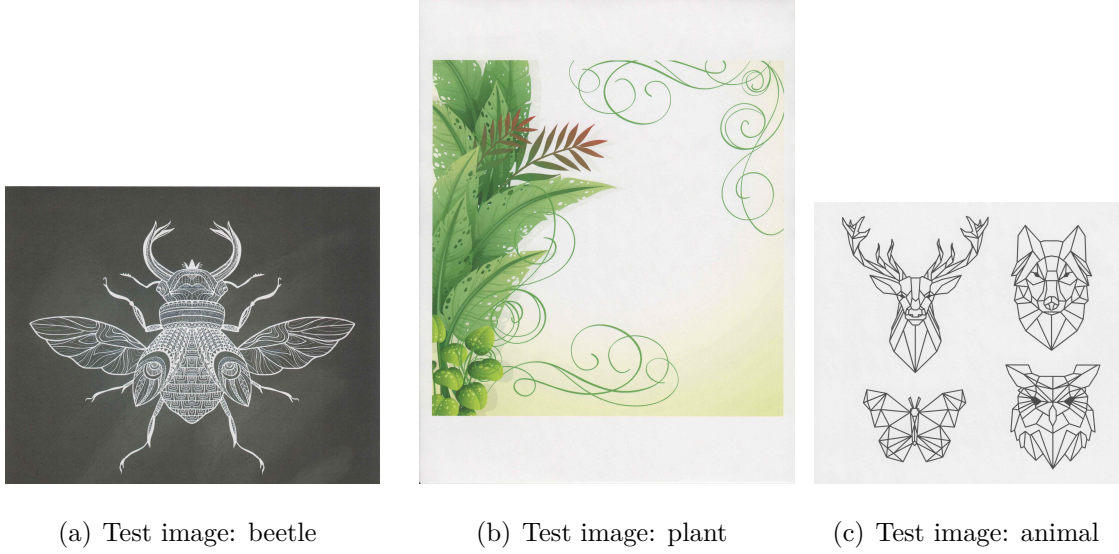
Table 2.2 shows the tile-STDDEV SSIM values calculated over the foreground pixels only. From it we can see that the value decreases a lot from (300 dpi, 150 dpi) pair to (300 dpi, 75 dpi) pair.

In addition to tile-STDDEV SSIM which is the averaged value over the foreground SSIM of the full-SSIM maps, we can also use the STDDEV of the foreground SSIM values in the full-SSIM maps as a feature. The workflow is shown in Figure 2.13,

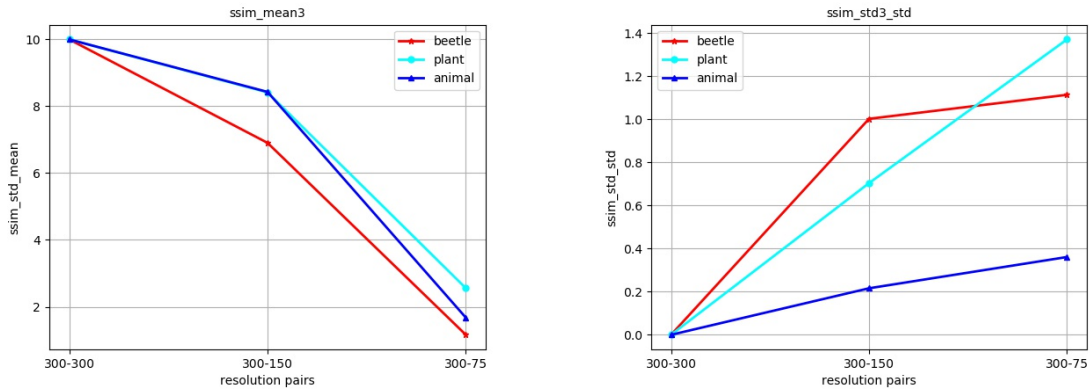


**Figure 2.13.** tile-STDDEV SSIM STDDEV workflow.

As is shown in Figure 2.13, the workflow is very similar to tile-STDDEV SSIM (shown in Figure 2.11). We also start with grayscale images, and get the full tile-STDDEV SSIM map from the STDDEV maps. The difference is that, instead of the average, we calculate the STDDEV of the foreground pixel values of the full tile-STDDEV SSIM map. Here, we call it the tile-STDDEV SSIM STDDEV. As the scan resolution goes down, image patches with high variations will see a lot of decrease in variation values, thus resulting in a decrease in tile-STDDEV SSIM values. In the mean time, the smooth areas will not change much. They can maintain relatively high values across all resolutions. As a result, as the scan resolution goes down, the tile-STDDEV SSIM STDDEV of image foreground patches will tend to increase. This can be used as an indicator of the relation between scan resolution and image quality.



**Figure 2.14.** Three example images.



(a) Tile-STDDEV SSIM of example images in Figure 2.14. We can see that the values between (300 dpi, 150 dpi) and (300 dpi, 75 dpi) are decreasing.

(b) Tile-STDDEV SSIM STDDEV of example images in Figure 2.14. We can see that the values between (300 dpi, 150 dpi) and (300 dpi, 75 dpi) are increasing.

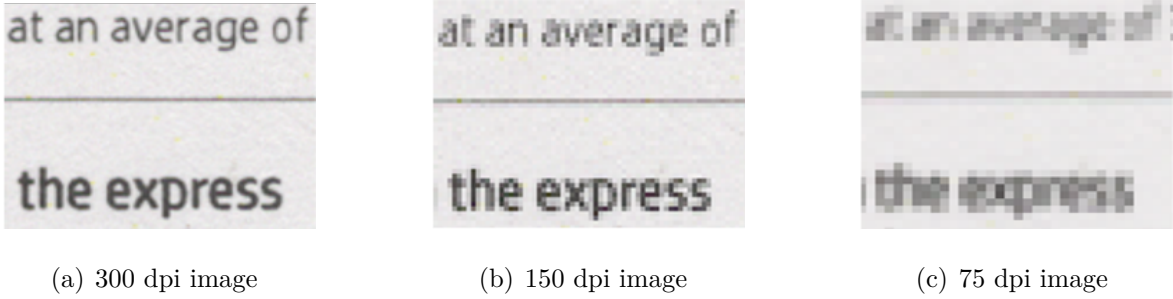
**Figure 2.15.** Tile-STDDEV SSIM and Tile-STDDEV SSIM STDDEV results of example images in Figure 2.14

Figure 2.14 shows three example images. Figure 2.15 shows the values of Tile-STDDEV SSIM and Tile-STDDEV SSIM STDDEV under different resolution pairs. We can clearly see that for all three images, the Tile-STDDEV SSIM values are decreasing, and Tile-STDDEV

SSIM values are increasing. Also, the different changes among them may indicate that they correspond to different optimal scan resolutions.

#### 2.4.2 Sample Power Spectrum MSE

With the decrease of scan resolution, one can clearly see the loss of high frequency details, such as the blurring of edges. As we can see from Figure 2.16, the characters are clearly distinguishable when scanned in 300 dpi. Although we can identify the words in 150 dpi images, the small-sized words in the top are a little bit blurry. The situation is even worse in the 75 dpi images, as the characters seem to have merged together, and it becomes impossible to tell the words.



**Figure 2.16.** Changes of scanned character images from 300 dpi to 75 dpi. For a better viewing experience, the lower resolution images are resized to the same size as that of 300 dpi.

The loss of frequency information can be reflected in the frequency domain using discrete-time Fourier transform (DTFT) [37]. The equation to calculate 1D DTFT is

$$F(u) = 1/L \sum_{i=0}^{L-1} S[i] e^{-j2\pi(ui/L)} \quad (2.5)$$

where  $L$  is the length of the input signal  $S$ , and  $i$  is the index of  $S$ .  $u$  is the index of the output 1D Fourier Transform  $F$ .

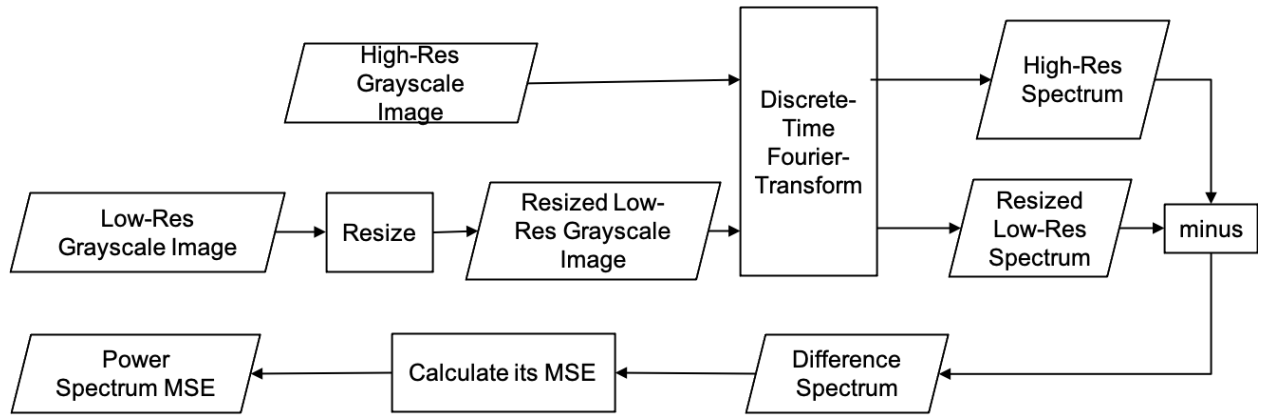
Similarly, the way to calculate the 2D DTFT is

$$F(k, l) = (1/(MN)) \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I[m, n] e^{-j2\pi(km/M + ln/N)} \quad (2.6)$$

where  $M$  and  $N$  are width and height of the input image  $I$ , respectively,  $m$  and  $n$  are the horizontal and vertical indices of  $I$ , respectively, and  $k$  and  $l$  are the indices of the horizontal and vertical indices of the output Fourier Transform  $F$ .

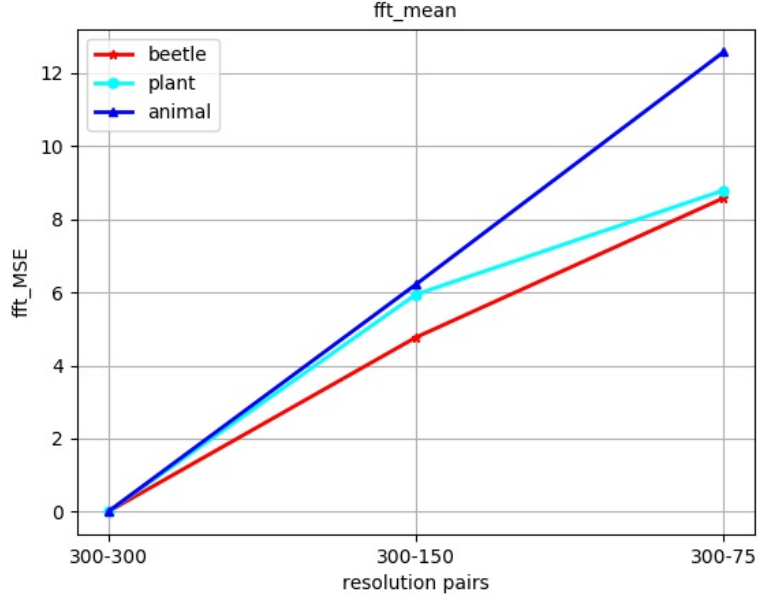
To simulate power spectrum changes, we plotted the power spectrum changes in Figure ?? . Here, to better visualize the power spectrum changes, we use a simulated 1D signal instead of 2D images. The simulated signal is  $\text{sinc}(8 \times x)$ , with  $x$  in range of  $-8, 8$ . The sampling interval is 0.125. The transforms are calculated using Equation 2.5. As introduced in Section 2.3.1, we use the area-based downsampling to get the low resolution signals, and use nearest neighbor interpolation to resize the signals to their original length.

We can consider using the sample power spectrum MSE to indicate the image quality changes. As one can expect, the value will increase with the decrease in scan resolutions.



**Figure 2.17.** Workflow to calculate sample power spectrum MSE.

The workflow to calculate the sample power spectrum MSE is shown in Figure 2.17. The images are converted to grayscale before passing into the workflow. Before calculating the discrete-time Fourier-transform, we need to resize the low-resolution images (150 dpi and 75 dpi) to the same size as that of 300 dpi reference images. This again is to simulate the true viewing situation.

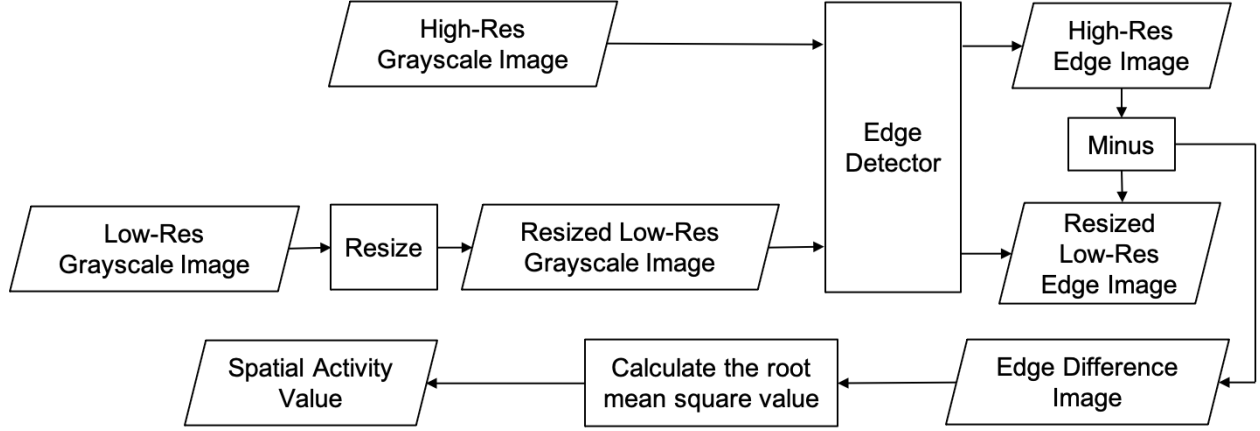


**Figure 2.18.** MSE of sample spectrum of example images. We can see that the values between the (300 dpi, 150 dpi) and the (300 dpi, 75 dpi) are decreasing.

Figure 2.18 shows the mean of sample spectrum under different resolution pairs for example images in Figure 2.14. We can clearly see that the MSE of the sample spectra increase from the (300 dpi, 150 dpi) pair to the (300 dpi, 75 dpi) pairs.

### 2.4.3 Spatial Activity

In the paper [38], the spatial activity is defined as the root mean square (RMS) difference between the edge maps of two images. In the paper, the authors use the Sobel edge detector [39] to get the edge map.



**Figure 2.19.** Workflow to calculate spatial activity.

The workflow to calculate the spatial activity is shown in Figure 2.19. Here, the color images are turned into grayscale before passed into the workflow. Since this is a FR feature, we also need to resize the low-resolution image to the size of the high-resolution image. The Sobel edge detector is also applied here to get the edge map. The detailed process are as follows:

$$E = \sqrt{(S * I)^2 + (S^T * I)^2} \quad (2.7)$$

where  $I$  is the input grayscale image, and  $E$  is the output edge map.  $*$  denotes the cross-correlation operation.  $S$  is the horizontal Sobel filter, which is

$$\begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix} \quad (2.8)$$

And  $S^T$  is the transpose of  $S$ , which is

$$\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} \quad (2.9)$$

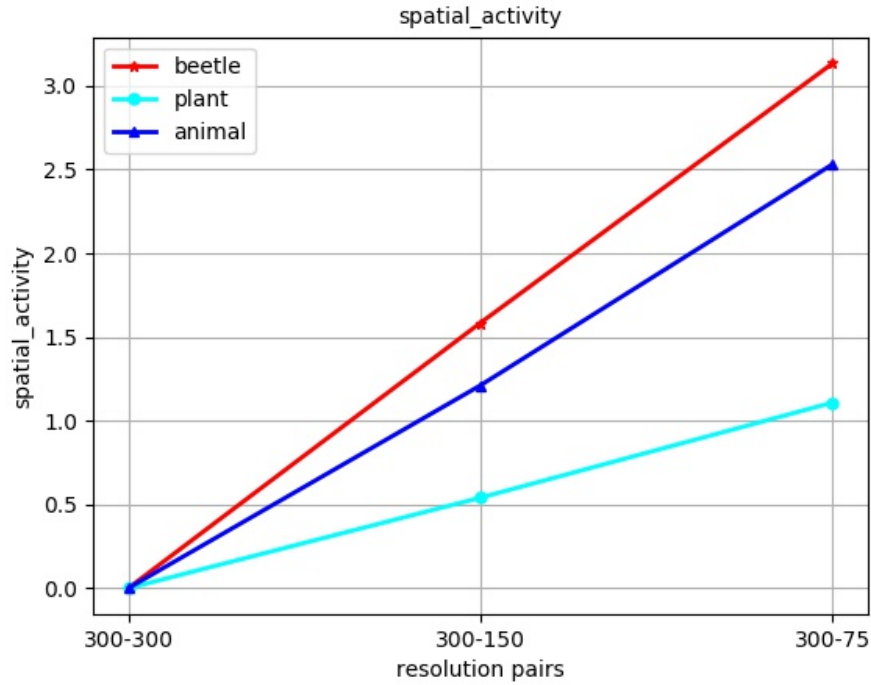
After we get the two edge maps  $E1$  and  $E2$ , then we calculate the difference  $D$  as

$$D = E1 - E2 \quad (2.10)$$

The final spatial activity value  $SA$  is calculated as

$$SA = \sqrt{1/(MN) \sum_{i,j} (D[i,j])^2} \quad (2.11)$$

where  $M$  and  $N$  are image width and height of edge maps, respectively, and  $i$  and  $j$  are horizontal and vertical indices of  $D$ , respectively.

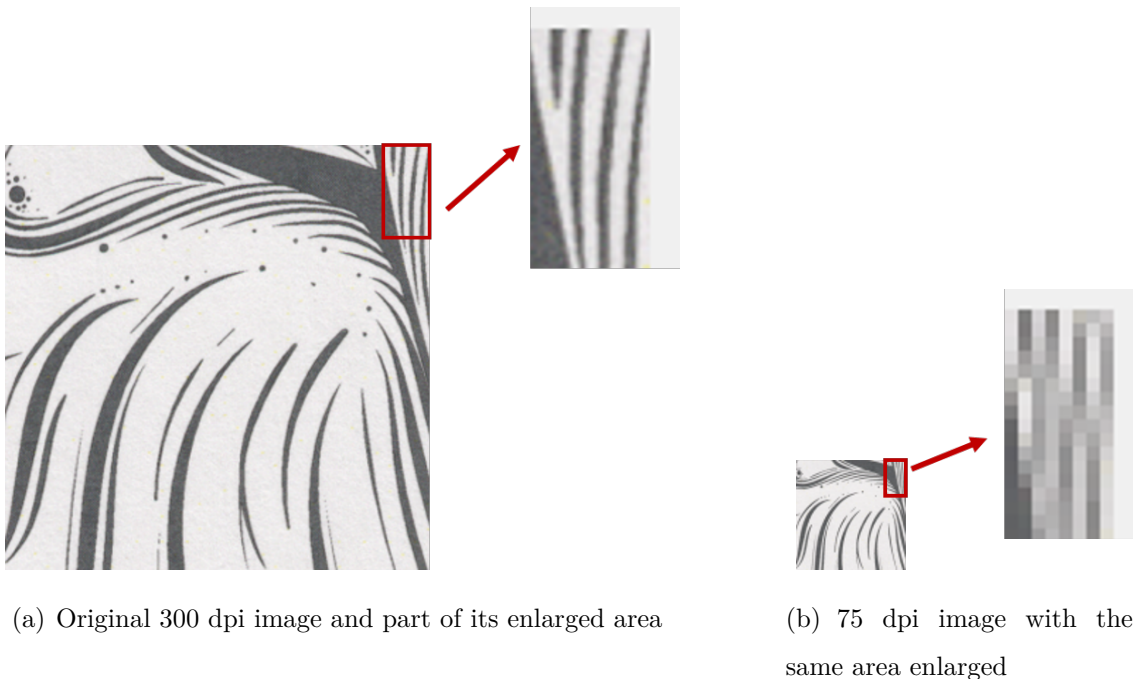


**Figure 2.20.** Spatial activity of example images in Figure 2.14. We can see that the values between (300 dpi, 150 dpi) and (300 dpi, 75 dpi) are increasing.

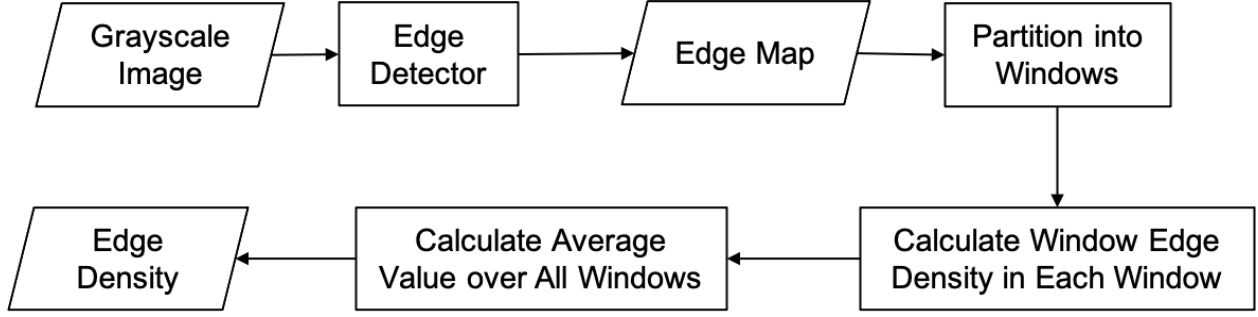
Figure 2.20 shows the spatial activity values for different resolution pairs of example images in Figure 2.14. We can clearly see the increase of spatial activity values from the (300 dpi, 150 dpi) to the (300 dpi, 75 dpi) resolution pair.

#### 2.4.4 Edge Density

As the scan resolution decreases, we can clearly see that the images become more and more blurry. Some sharp edges may even disappear. This is much more obvious with document images or line drawings, as their edges are more distinct and maybe very close to each other. As the resolution decreases, two edges are likely to merge into one edge or become visibly indistinguishable. All of this would cause some of the characters to become unidentifiable, and drawings to become blurry, as shown in Figure 2.21. With this motivation in mind, we define a new NR QE that can take both the foreground and background pixels into consideration, and thus better reflect the local pixel changes.

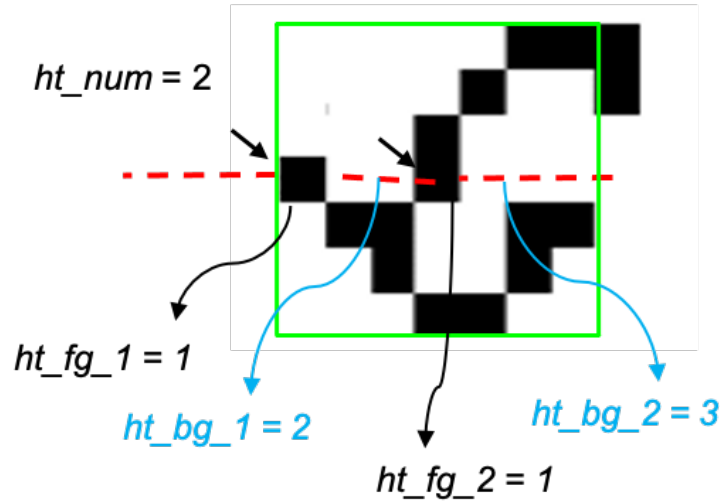


**Figure 2.21.** Change of line drawing with decreasing resolutions. The 300 dpi image in (a) has very good details with distinct lines, while the 75 dpi image of (b) has very blurry edges. For better comparison, we enlarge the same area, the upper right corner, to be the same size. From the enlarged area, we can barely even see any edges in the 75 dpi image.



**Figure 2.22.** Workflow to calculate edge density.

Figure 2.22 shows the workflow to calculate edge density. Color images are converted to grayscale before being passed into the workflow. Then we apply the Canny edge detector [40] to get the edge map, on which we calculate the edge density. In order to be able to capture the changes in detail, we partition the edge map into small windows. In order to assure the pixel correspondences and window numbers from different scan resolutions, we choose different window sizes according to their resolutions. The window sizes are the same as in Table 2.1. Then edge densities are calculated in each window, and we take the average value as the edge density of the image.



**Figure 2.23.** An example of horizontal edge density.

The edge density is comprised of two parts: horizontal edge density and vertical edge density. Figure 2.23 shows an example of how we compute edge density. The process is as follows:

(1) Imagine a horizontal line (red line) passing through the center of the small window (the green box).

(2) Count the number of horizontal transitions along the horizontal line. A transition is defined as sudden change from *background* to *foreground*. In this case, the number of horizontal transitions  $ht\_num$  is 2.

(3) Following the horizontal line, count the number of consecutive foreground pixels, and then number of consecutive background pixels, and then the number of consecutive foreground pixels, and so on. In this case, the numbers are 1, 2, 1, 3, denoted in the figure as  $ht\_fg\_1, ht\_bg\_1, ht\_fg\_2, ht\_bg\_2$ , respectively. In Figure 2.23, the number of consecutive foreground and background pixels are color-coded as black and blue, respectively. Note that the smallest number of foreground or background pixels is 1, since the counting process is based on pixels, and one pixel belongs either to the foreground or the background.

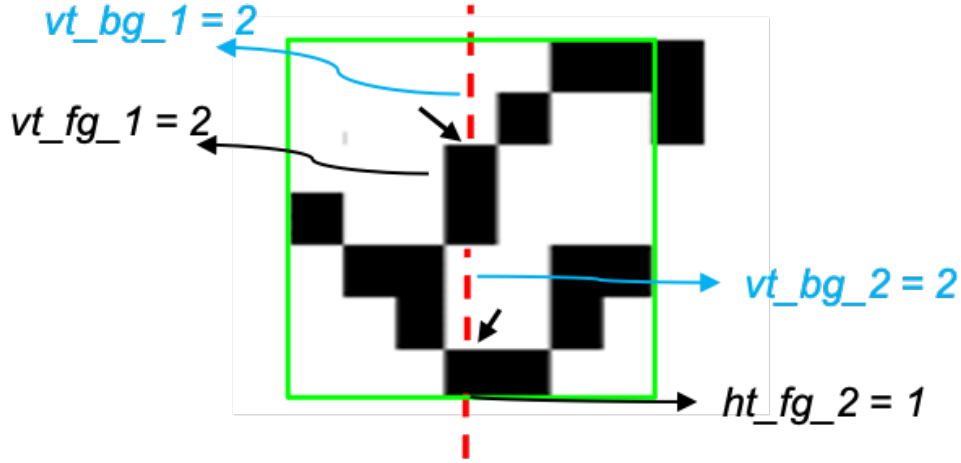
(4) The horizontal edge density  $ht\_density$  is calculated using the formula

$$ht\_density = ht\_num \cdot \left( \sum_i^I \mathbb{1}(ht\_fg\_i > 0) \frac{1}{ht\_fg\_i} + \sum_j^J \mathbb{1}(ht\_bg\_j > 0) \frac{1}{ht\_bg\_j} \right) \quad (2.12)$$

where  $\mathbb{1}$  is the indicator function.  $I$  is the total number of sequences of foreground pixels, and  $J$  is the total number of sequences of background pixels.  $ht\_fg\_i$  is the number of consecutive foreground pixels in the  $i$ -th sequence of foreground pixels, and  $ht\_bg\_j$  is the number of consecutive background pixels in the  $j$ -th sequence of foreground pixels. Their values are all larger than 0. In Figure 2.23, the horizontal edge density is  $2 \cdot (1/1 + 1/2 + 1/1 + 1/3) = 5.67$ .

The reason why we sum the reciprocals of consecutive foreground/background pixel numbers are that it can better reflect the merging of foreground/background line segments. Clearly, image quality degrades most when foreground and background pixels are interleaved by one pixel. Namely, one foreground pixel is followed by one background pixel, and

then followed by one background pixel, and so on. In this situation, simply downsampling the image from 300 dpi to 150 dpi would make the resultant image lose a lot of information. With large pixel numbers, like a 10-pixel-width foreground line segment followed by a 10-pixel-width background segment, the image quality would not be affected much even when downsampled to 75 dpi. As a result, we use the reciprocals of line segments to reflect drastic changes in the number of foreground and background pixels.



**Figure 2.24.** An example of vertical edge density.

The vertical density is calculated in a similar way, as shown in Figure 2.24. We draw a vertical line through the window center, and count the number of consecutive foreground and background pixels. The equation to calculate the vertical edge density  $vt\_density$  is

$$vt\_density = vt\_num \cdot \left( \sum_i^I \mathbb{1}(vt\_fg\_i > 0) \frac{1}{vt\_fg\_i} + \sum_j^J \mathbb{1}(vt\_bg\_j > 0) \frac{1}{vt\_bg\_j} \right) \quad (2.13)$$

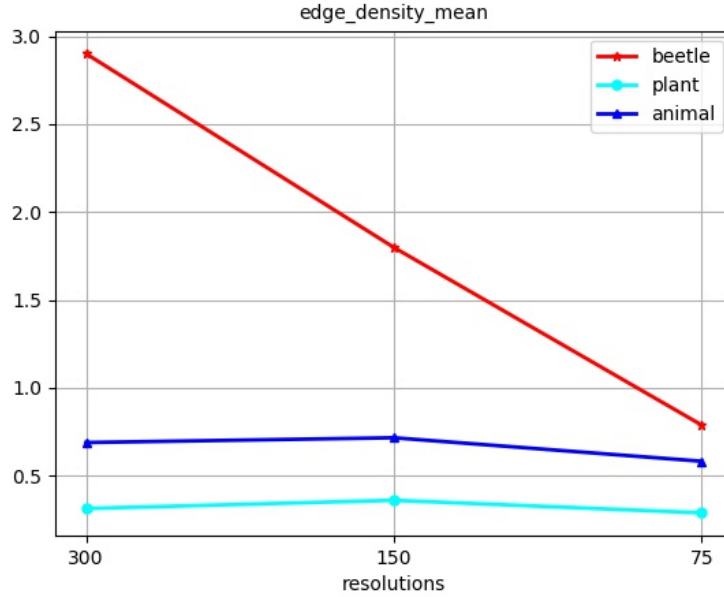
where  $\mathbb{1}$  is the indicator function.  $I$  is the total number of sequences of foreground pixels, and  $J$  is the total number of sequences of background pixels.  $vt\_num$  is the number of vertical transitions.  $vt\_fg\_i$  is the number of consecutive foreground pixels in the  $i$ -th sequence of foreground pixels along the vertical line, and  $vt\_bg\_j$  is the number of consecutive

background pixels in the  $j$ -th sequence of foreground pixels. Their values are all larger than 0. In Figure 2.24, the horizontal edge density is  $2 \cdot (1/2 + 1/2 + 1/2 + 1/1) = 5$ .

Clearly, the direction that would achieve the maximum edge density value is the most vulnerable to suffer from changes in scan resolution. So the final edge density  $edge\_density$  is

$$edge\_density = \max(vt\_density, ht\_density) \quad (2.14)$$

Accordingly, the edge density for the window shown in Figure 2.23 and Figure 2.24 is  $\max(5.67, 5) = 5.67$



**Figure 2.25.** Edge density of example images in Figure 2.14.

Figure 2.25 shows the edge density mean of the example images in Figure 2.14. We can see that the value generally decreases with the decrease in scan resolution. The values in the 'beetle' image drop the most, while the changes in the other two images are very subtle, and even increase slightly in going from 150 dpi to 75 dpi. These can indicate different optimal scan resolutions for the example images.

### 2.4.5 Edge Contrast

As shown in Figure 2.21, blurry edges are the most apparent quality degradation when the scan resolution goes down from 300 dpi to 75 dpi. The separated edges merged together. Some areas with high frequency alternation between black and white become gray. Although the edge density can reflect the merging of edges, they cannot represent the changes of pixel values around edges. Those pixel value changes around edges are important since they reflect the contrast, which is a very important QE. As a result, we propose a QE based on Michelson contrast [41].

Contrast is the difference in color or intensity that makes an object distinguishable from its surrounding background. The human visual system (HVS) is more sensitive to contrast than absolute intensity; and this enables us to see a world similarly despite the big intensity changes from day to night. There are many ways to define contrast [42], but some of the most commonly used are Weber contrast [43], Root mean square (RMS) contrast [44], and Michelson contrast.

Weber contrast is based on the Weber's law, in which the amount of noticeable change in intensity is proportional to the value of the intensity. It is defined as

$$c\_weber = \frac{I - I_b}{I_b} \quad (2.15)$$

where  $c\_weber$  is the Weber contrast.  $I$  and  $I_b$  are the intensity of the foreground and background, respectively. Weber's contrast is mostly used in cases where the foreground's intensity is very similar to the background, or the background is very uniform [45].

RMS contrast is defined as the STDDEV of pixel intensities:

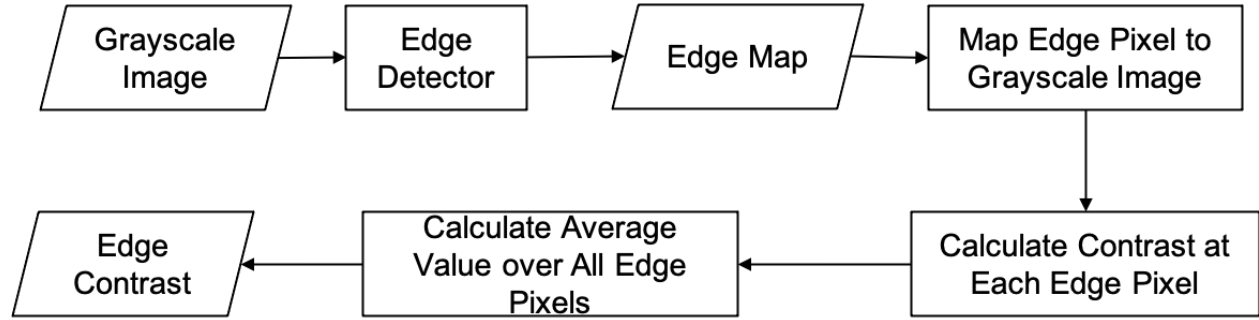
$$c\_rms = \sqrt{1/(MN) \sum_{i,j} (I_{ij} - I_{avg})^2} \quad (2.16)$$

where  $c\_rms$  is the RMS contrast.  $M$  and  $N$  are the width and height of the region of interest in the input image, respectively, and  $i$  and  $j$  are horizontal and vertical indices, respectively.  $I_{ij}$  is the  $ij$ -th element of the image, and  $I_{avg}$  is the average intensity of the region of interest in the image.

Michelson contrast is the contrast used in to calculate an imaging system's modulation transfer function (MTF) [46], in which the value is calculated as the ratio of output contrast over twice the input intensity midpoint. Michelson contrast is defined as

$$c\_michelson = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} \quad (2.17)$$

where  $c\_michelson$  is the Michelson contrast.  $I_{max}$  and  $I_{min}$  are the maximum and minimum intensity values, respectively, of the areas in question. Since it is easier to calculate than the RMS contrast, and lowering scan resolution would simultaneously affect the maximum and minimum intensity values, we choose Michelson contrast to calculate the contrast changes from 300 dpi to 150 and 75 dpi images.

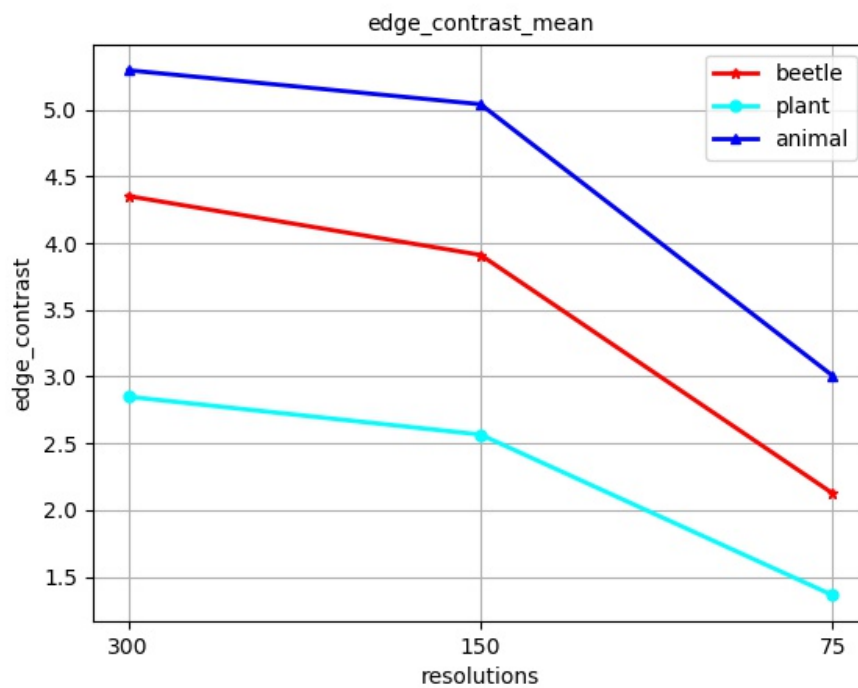


**Figure 2.26.** Workflow to calculate edge contrast.

Figure 2.26 shows the workflow to calculate edge contrast. Since it is a NR QE, we only need one input image. Color images are converted to grayscale before being fed into the pipeline. We use the Canny edge detector [40] to detect edges since it can result in 1-pixel-width edge maps. After getting the edge pixels, we map them back to the grayscale image. Specifically we will use the coordinates of edge pixels in the edge map to identify the edge pixels in the grayscale image, and to calculate the Michelson contrast in a small window. The window sizes are based on the scan resolutions (shown in Table 2.3); so that the calculated contrast values correspond to roughly the same area, independent of the resolution of the image.

**Table 2.3.** Scan resolutions and their corresponding window sizes

Resolution (dpi)	Window width/height (pixel number)
300	13
150	7
75	3



**Figure 2.27.** Edge contrast of example images in Figure 2.14.

Figure 2.27 shows how edge contrast changes with respect to different resolutions of the example images in Figure 2.14. As expected, we can see that the values decrease from 300 dpi to 75 dpi. Also, at any fixed resolution, the edge contrast decreases as the image becomes less binary and more continuous-tone.

### 2.4.6 Support Vector Machine

After the feature vector is obtained, we employ a support vector machine (SVM) to classify each page according to its desired resolution.

The SVM is useful for data classification. It maps the data into a higher dimensional space, and finds a linear separating hyperplane with the maximal margin in it [47]. The SVM is a binary classifier. When used for multi-class classification, there are three methods: one-against-all, one-against-one, and directed acyclic graph (DAG). In the one-against-all method, a  $k$ -class classifier has  $k$  models. The  $i$ -th SVM is trained with all examples in the  $i$ -th class with positive labels, and all other examples with negative labels [48]. But this method may be biased by a big size difference between the numbers of positive and negative labels. The one-against-one method requires an SVM model between every two classes. So a  $k$ -class classifier needs  $k(k-1)/2$  models. A test sample will be fed into all models and each model will output a result. The final result is the one with the maximum number of votes [49]. The DAG has the same training process as the one-against-one method. But in the testing phase, it uses a rooted binary directed acyclic graph. A  $k$ -class classifier has  $k(k-1)/2$  internal node classifiers and  $k$  leaf classifiers. A test sample will need to pass all classifiers to get the prediction result [50]. Research has shown that one-against-one and DAG are more suitable for practical use [51].

In this project, we choose the one-against-one method. We use the LIBSVM [52] and radial basis function (RBF) kernel to do the training and prediction.

## 2.5 Experiment Results

### 2.5.1 Data Collection

We collect our own data for training and testing. The collection process follows the routine below.

(1) PDF files are collected from the internet, and printed using an HP LaserJet 500 color MFP M575.

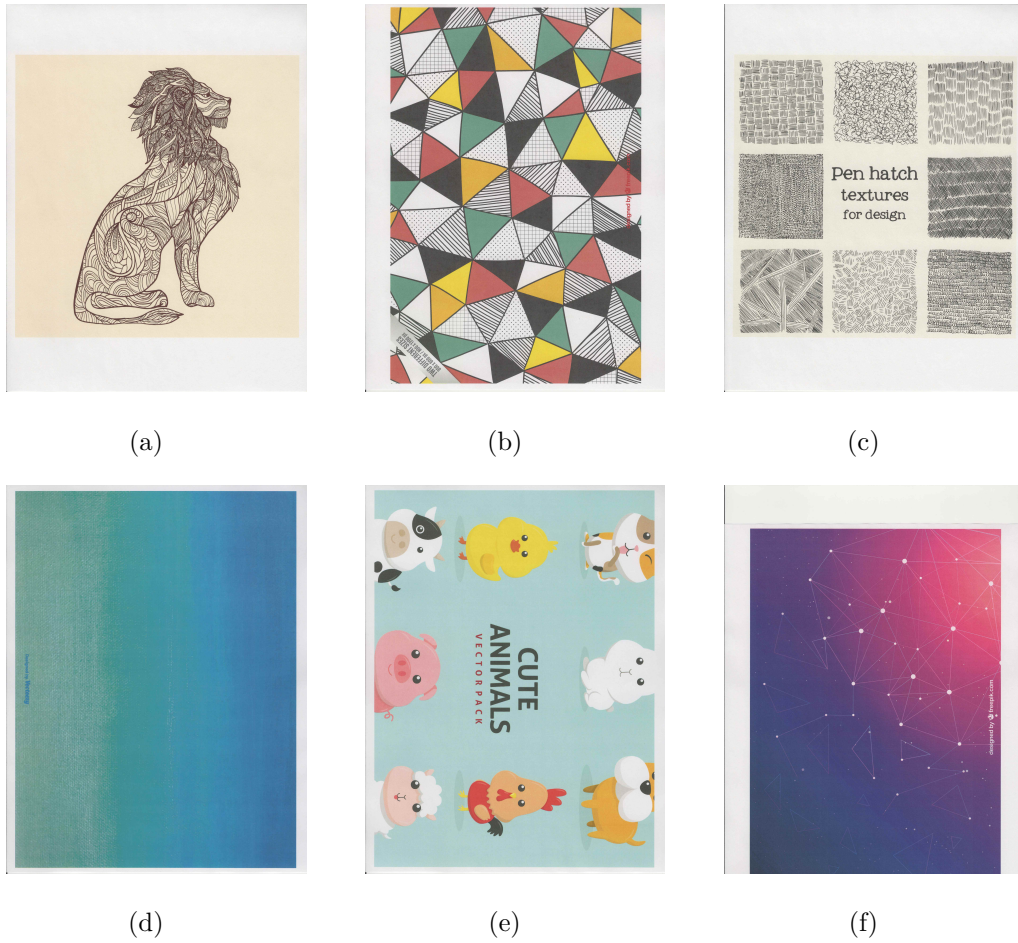
(2) Pages are scanned into 300 dpi digital images using HP Officejet Enterprise Color Flow MFP X585 with the HP-internal *w\_scan* routine. *W\_scan* is chosen instead of regular scanning because it doesn't include post-processing steps like edge enhancement and JPEG compression, which may unnecessarily affect our judgement.

(3) In order to augment the data set, the 300 dpi images are partitioned into  $300$  pixel  $\times 300$  pixel sub-images. Duplicated sub-images are removed. Each sub-image is treated separately as an individual image. In this dissertation we don't consider stitching the sub-images together.

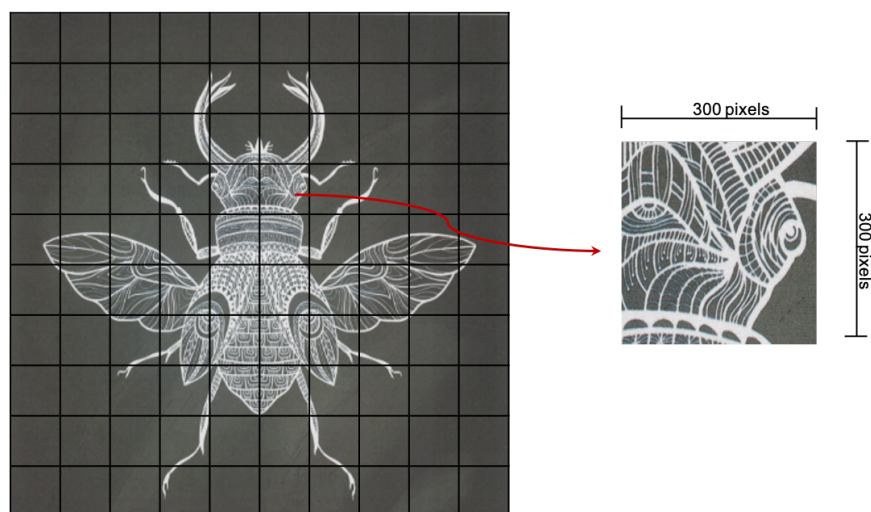
(4) Images of other resolutions are achieved from 300 dpi sub-images using area-based downsampling as introduced in Section 2.3.1.

We also include image rotation for data augmentation. We do not use flipping because most of our features are flipping-invariant.

In this dissertation, we mainly focus on line drawings and cartoon images, as shown in Figure 2.28.



**Figure 2.28.** Some of the example images in the dataset.

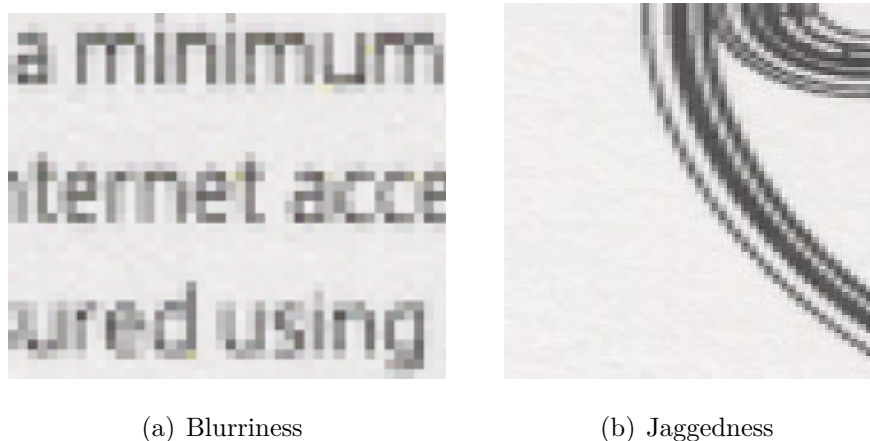


**Figure 2.29.** Demonstration of how sub-images are obtained.

Figure 2.29 shows how we get the sub-images. The big image on the left is the scanned whole image. We then partition it into small sub-images of size 300 dpi  $\times$  300 dpi, as shown using the black grid. The right part is an enlarged sub-image.

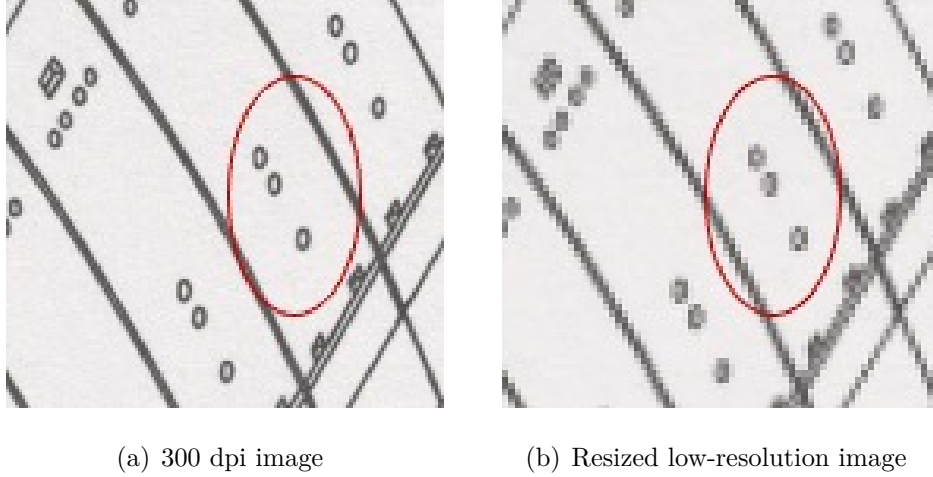
*Optimal scan resolution* is quite an ambiguous term. Although we explain it as the lowest scan resolution with acceptable image quality, we still need to make it clear what the acceptable image quality is. In the scope of this thesis, we have defined two criteria for acceptable image quality:

(1) For 75 dpi and 150 dpi images, we resize them to the size of their 300 dpi counterpart. Since blurriness and jaggedness are two most obvious degradations in our scope (Figure 2.30). The blurriness or jaggedness is either imperceptible, or perceptible but not annoying in the resized image. Clearly, this is a NR criterion, as we don't need a reference image.



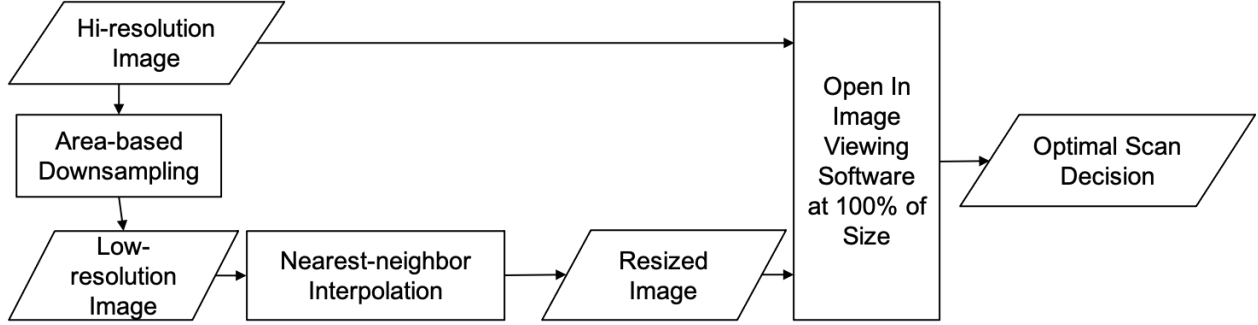
**Figure 2.30.** Examples of blurriness and jaggedness.

(2) If we resize the low-resolution image to the size of the 300 dpi reference, the resized image must contain all the details that the reference image contains. For example, if there is a circle in the 300 dpi image, there must be also a circle (not a square, or not sure if it's a circle or square) in the resized image, as is shown in Figure 2.31.



**Figure 2.31.** An example of detail loss. We can clearly see three circles in the red oval in (a). But in the same area of (b), we are not very sure if they are circles.

Figure 2.32 shows the workflow of how we label the optimal scan resolution for an image. The high-resolution image is the 300 dpi reference image, and the low-resolution images are 150 dpi and 75 dpi images. As is stated in Section 2.3.1, 300 dpi images are achieved through scanning, and low-resolution images are captured using the area-based downsampling method. The low-resolution (150 dpi and 75 dpi) images are resized by nearest-neighbor interpolation to the same size as that of their 300 dpi reference image. All images, the reference image, and the two resized images are opened in the same image viewing software like Nomacs, at 100% of image size. The viewer then looks at the three images side by side, and picks the optimal scan resolution based on the criteria mentioned above. All sub-images are viewed on a 14-inch HP Elitebook 840 G3 laptop. The display resolution is 1366 pixels  $\times$  768 pixels. The subject was seated straight in front of the laptop, slightly below the eye height. The distance between the laptop and the subject's eyes is around three times the height of the laptop screen. The subject is asked to wear glasses or contact lenses if needed. For the work reported in this dissertation, there was only one subject; and that was the author.



**Figure 2.32.** Workflow to label optimal scan resolution. Each low-resolution (150 dpi and 75 dpi) image is resized to the same size as that of its 300 dpi reference image. The reference image and both of the resized images are opened in the same image viewing software. Finally, the optimal resolution for the input image is chosen.

The resolutions and their corresponding number of scanned sub-images collected are shown in Table 3.1. For each resolution, we have 537 sub-images and the data set has in total 1,611 sub-images. These sub-images are achieved from 45 scanned whole pages.

**Table 2.4.** Scan Resolution and the Number of Sub-images

Scan Resolution	Sub-Image#
300 dpi	537
150 dpi	537
75 dpi	537
<b>Overall</b>	<b>1,611</b>

### 2.5.2 Test Results

After getting the features, we use SVM to help train and test the accuracy of our model. In practice, we use LIBSVM [52] with radial basis function (RBF) as the kernel, to help the training and testing process. LIBSVM uses grid search and is very efficient in finding the optimal parameters. In order to get a more compelling result, we use 5-fold cross-validation to test our model. Precisely, we split the training data equally into five folders. For each experiment, we pick four of them as the training set to get the SVM model, and the remaining

one folder as the testing set to get the testing accuracy. We repeat this process four times, each time with a different folder as the testing set. The final accuracy is the average of all five experiments. We also test the training accuracy by running the model on the training data.

Table 2.5 shows the training and cross-validation results. We see that we achieve a training accuracy of 95.2%, and validation accuracy of 93.4% with a STDDEV of 1.2%.

**Table 2.5.** Training and validation results.

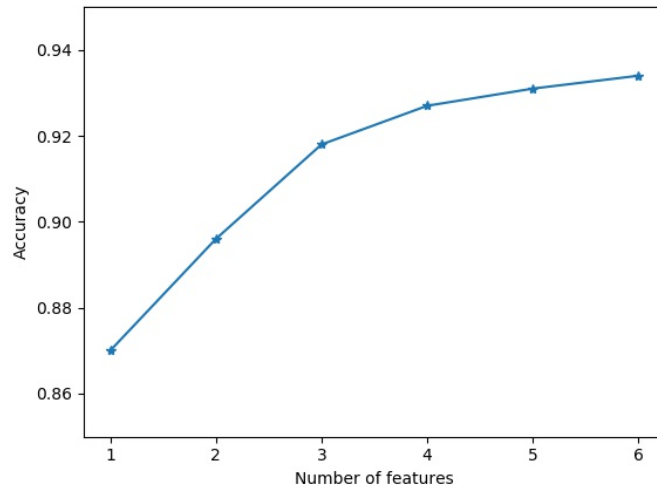
	<b>Accuracy</b>	<b>STDDEV</b>
Training	95.2%	0.3%
Cross-validation	93.4%	1.2%

We conduct Support Vector Machine Recursive Feature Elimination (SVM-RFE) [53] to rank the importance of the features we defined. We start by removing only one feature of the feature vector, and examine the accuracy of the model. The feature that results in the least accuracy loss will be removed. This process will continue until we only have one feature left. Table 2.6 shows the rank of importance of our features. From the table, we can see that edge contrast is the most important, followed by tile-STDDEV SSIM, sample power spectrum MSE, and tile-STDDEV SSIM STDDEV. Spatial activity and edge density are the two least important features.

**Table 2.6.** The rank of importance for features.

<b>Rank of Importance</b>	<b>Feature</b>
<b>1</b>	edge contrast
<b>2</b>	tile-STDDEV SSIM
<b>3</b>	sample power spectrum MSE
<b>4</b>	tile-STDDEV SSIM STDDEV
<b>5</b>	spatial activity
<b>6</b>	edge density

Figure 2.33 shows the accuracy changes with the number of features. The features are added in the order of importance. That is, '1' means edge contrast only with the accuracy of 87%; '2' means edge contrast and tile-STDDEV SSIM with the accuracy of 89.6%; '3' means edge contrast, tile-STDDEV SSIM and sample power spectrum MSE, with the accuracy of '91.8%'; and '6' means all features together with an accuracy of '93.4%'.



**Figure 2.33.** SVM-RFE analysis result based on 5-fold cross validation.

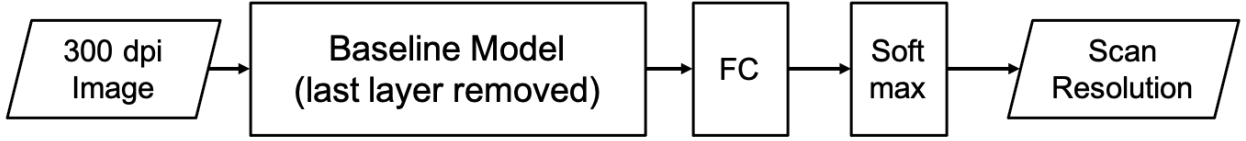
### 2.5.3 Light Weight Neural Networks

Recently, convolutional neural networks (CNN) or deep neural networks (DNN) have seen a rapid development in computer vision areas like classification [54], object detection [55][56], image or video super-resolution [57], and image segmentation [58]. NIMA (Neural Image Assessment) [59] is one CNN proposed to predict the distribution of human ratings on image quality. The proposed method uses pretrained VGG16 [60], Inception-v2 [61], or MobileNet [62] as the base network. The last layer is replaced by a fully-connected (FC) layer which indicates the final rating.

Inspired by NIMA, we also tried different base CNNs on our task. But as we only have three resolutions, the output of the final FC layer has three nodes, followed by the softmax function in Equation 2.18, as shown in Figure 2.34.

$$\delta(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, \quad i = 1, 2, \dots, K \quad (2.18)$$

where  $\delta(z_i)$  is the probability of the value  $z_i$ , and  $i$  is the index of the input values.



**Figure 2.34.** Proposed CNN structure. The 300 dpi input image is passed into a baseline network with the last layer replaced by a FC layer. Then softmax outputs a probability that will decide the optimal scan resolution.

Since our algorithm is targeted for use in multifunction peripheral (MFP) devices, the performance in terms of required computation and model size must be put in high priority. We choose to use some light-weight CNNs. Specifically, we use MobileNetV2 [63], MobileNetV3 [64], and GhostNet [65] as the baseline network, with their weights pretrained using ImageNet [66] and the last layer replaced by a FC layer.

MobileNetV2 came up with the structure “Inverted Residuals with Linear Bottleneck”, where shortcut connections are between the bottleneck layers, and the intermediate expansion layer uses lightweight depthwise convolutions [62] to reduce the computations. Besides, the ReLU (rectified linear unit) functions at the end of the bottlenecks are removed. This can help preserve information in the data for small networks. MobileNetV3 is achieved through NAS (network architecture search) and the NetAdapt algorithm. They also add SE (squeeze and excitation) [67] modules in the structure that use a hard-swish function to be the activation function at some layers. Two models: MobileNetV3-Large and MobileNetV3-Small were released, which correspond to high and low resource use cases, respectively. Since we are most concerned with the speed, we chose to test the MobileNetV3-Small model. Ghostnet

applies a series of cheap linear transformations instead of traditional convolution operations. This can reduce the computation cost while maintaining good classification accuracy.

As we don't have enough data to train the CNN models, we use the pre-trained models trained on ImageNet [66], and fine-tuned the baseline models using our dataset. In the training process, we train each model for 100 epochs and choose the model parameters corresponding to the best validation accuracy. The results are shown in Table 2.7. For better comparison, we also show the SVM results in the table.

**Table 2.7.** Comparison of SVM and light-weight CNNs with different baseline networks.

	<b>Latency/s</b>	<b>Model Size /Mb</b>	<b>Accuracy /%</b>
SVM	0.92	0.16	93.4
MobileNetV2	0.02	8.9	92.2
MobileNetV3-large	0.02	17.0	91.6
MobileNetV3-small	0.01	6.2	92.2
GhostNet	0.02	15.8	91.9

The latency is tested on an HP laptop Elitebook 840 G3 with a CPU: Intel Core i7-8700 @3.20GHz  $\times$  12. The CNNs are written using highly-optimized PyTorch, while the SVM code is written in Python. We cannot compare the CNNs directly with the SVM algorithm since they are written in different programming frameworks and the SVM code is not optimized. But from Table 2.7 we can see that in the scope of light-weight CNNs, MobileNetV3-small has the smallest latency, while other baseline networks have similar speed.

For all CNN models, the model size is the actual memory used in a computer. For the purpose of saving memory, only the state\_dict is saved. The state\_dict is a Python dictionary object that maps each layer to its parameter tensor. The saved model only contains state\_dict from convolutional layers, linear layers, and registered buffers like batchnorm's running\_mean. From the table, we can see that SVM has the smallest model size, and MobileNetV3-large occupies the most memory, followed by GhostNet, which is 15.8 Mb.

MobileNetV2 and MobileNetV3-small occupies much less memories, with 8.9 Mb and 6.2 Mb, respectively.

The performance is further tested using Raspberry Pi 4 Model B @1.5 GHz  $\times$  4, as this is much closer to the computation power in current MFPs. As is shown in Table 2.7, we only test the results of CNNs.

**Table 2.8.** Latency of light-weight CNNs on Raspberry Pi.

	<b>Latency/s</b>
MobileNetV2	1.91
MobileNetV3-large	3.32
MobileNetV3-small	3.29
GhostNet	2.08

From Table 2.8 we can see that the smallest latency is *1.91s* from MobileNetV2. This is actually too slow to scan a single image. As a result, we need to do model quantization [68] to achieve better performance.

Typically, the weights in DNNs or CNNs are 32-bit floating numbers (FP32). If we can convert them to 8-bit integer, we can both reduce the memory and increase the inference speed by  $4\times$ . In PyTorch, there are two types of quantization: quantization to unsigned 8-bit integers (UINT8) or signed 8-bit integers (INT8). The former is mainly for activation functions, and the latter is mainly for weights.

When quantizing the activation function, batches of data are fed through network and the resulting distributions of the different activations are calculated. These distributions are then used to determine how specifically the different activations should be quantized at the inference time.

The formula is

$$x_q = round \left( (x_f - min_{x_{fs}}) \cdot \left( \frac{2^n - 1}{max_{x_{fs}} - min_{x_{fs}}} \right) \right) \quad (2.19)$$

where  $x_q$  is the quantized value.  $x_f$  is the input floating point value.  $\min_{x_{fs}}$  and  $\max_{x_{fs}}$  denote the minimum and maximum values of all the floating values from the same layer or all layers.  $n$  is the number of bits used for quantization. Usually  $n$  is 8. *round* denotes the rounding to the nearest integer operation.

Quantization to INT8 is mainly for weights. The equation is

$$x_q = \text{round} \left( x_{fs} \cdot \left( \frac{2^n - 1}{2 \cdot \max_{|x_{fs}|}} \right) \right) \quad (2.20)$$

where  $|x_{fs}|$  is the absolute value of all floating point weights.

There are three types of model quantization: dynamic quantization (DQ), post training quantization (PTQ), and quantization aware training (QAT).

In DQ, the weights are quantized ahead of time, but the activation functions are quantized during inference. This is for situations where inference time is dominated by loading weights rather than matrix computations, like LSTM (long short-term memory) and Transformer with small batch sizes.

PTQ is for situations where the memory and computation power is limited. It actually quantizes the model weights and activation functions after the model is trained. For weights, it can globally quantize all weights, or quantize all weights on a per-channel basis using Equation 2.20. Usually the second method can achieve better accuracy than the first one, although both would experience an accuracy loss.

QAT [69] goes another round of training after the model is trained using FP32. In QAT training's back-propagation step, the weights are stored in FP32; so that they can be easily nudged by small amounts. But the forward propagation simulates quantized inference by implementing the quantization as follows:

- (1) Weights are quantized before they are convolved with the input
- (2) Activations are quantized at points where they would be during inference.

Since there is more training involved, and the training process is 'aware' that the model would be quantized in inference time, QAT usually achieves better accuracy than DQ and PTQ.

**Table 2.9.** Different quantized models of MobileNetV2.

MobileNetV2 Model	Latency /s	Model Size /Mb	Accuracy /%
FP32	1.91	8.89	92.2
PTQ with global quantization	0.13	2.34	55.5
PTQ with per-channel quantization	0.13	2.65	88.2
QAT	0.13	2.65	93.7

Table 2.9 shows the result using different quantization methods for MobileNetV2 model. Here, we choose MobileNetV2 over other CNN models because of its high accuracy and low latency. The latency is tested using Raspberry Pi 4 Model B @1.5 GHz  $\times$  4. We didn't use DQ since it does not apply for our situation here. We tested the PTQ with both global and per-channel quantization. We can see that in Table 2.9, The inference time is reduced a lot, from 1.91s per image to 0.13s per image. Moreover, the sizes of quantized models are much smaller than that of the FP32 model. PTQ with global quantization occupies the least storage, with only 2.34 Mb. But the accuracy is only 55.5%. PTQ with per-channel quantization achieves much better accuracy at 88.2%, although model size is a little bit bigger (2.65 Mb) than PTQ with global quantization. QAT has the same model size as that of PTQ with per-channel quantization. With more training, it gets an accuracy at 93.7%.

## 2.6 Conclusion

Current scanning mechanisms scan all pages into the same resolution. If the scan resolution is 300 dpi or higher, the resulting images may waste some memory. If the scan resolution is 75 dpi, many scanned materials, like fine drawings, may lose a lot of details. Moreover, a scan resolution between 75 and 300 dpi, like 150 dpi, may make part of images very blurry, while part of images occupy more memory than needed.

To tackle this problem, we first simulate the real viewing scenarios when people enlarge images on their computers, and find out that most image viewing software uses nearest-

neighbor interpolation when zooming in an image. Then, we define the criteria based on which to judge an image’s quality. That is, when resized to the same size as that of 300 dpi, the blurriness or jaggedness is either imperceptible, or perceptible but not annoying in the resized image. Also, the resized image should keep all details that its 300 dpi reference image has. The resolution that meets both criteria is called the optimal scan resolution.

In this section, we propose a traditional machine learning-based method. We first extract some FR features, like tile-STDDEV SSIM, tile-STDDEV SSIM STDDEV, sample power spectrum MSE, and spatial activity to reflect how truthful a low-resolution image is to its 300 dpi counterpart. We also find some NR features, like edge density and edge contrast. These features can reflect the view quality at different resolutions. Concatenating these features and feeding them into an SVM, we can have a prediction accuracy at 93.4%.

We also test some light-weight CNN models like MobileNetV2, MobileNetV3-large, MobileNetV3-small, and GhostNet, and their prediction accuracies are 92.2%, 91.6%, 92.2%, and 91.9%, respectively. These results show that our SVM model has the best result.

Also, in order to solve the problem that CNN models consume a lot of memory and computation power, we quantize the original FP32 models to integers. The quantization is tested on MobileNetV2 model. Results show that QAT quantization can reduce the memory and increase the inference speed.

## 2.7 Major Contributions

My major contributions for this project are as follows:

- Generate the dataset to compare images at different scan resolutions, and define ground truth optimal resolution for each tile image.
- To the best of our knowledge, this is the first study that feeds both FR and NR QEs to an SVM to estimate the quality of different resolution images.
- To the best of our knowledge, this is the first study that uses different window size-based edge density as the QE to estimate the quality of different resolution images.
- Introduce a brand new NR QE: edge density.

- Introduce two brand new FR QEs based on SSIM: tile-STDDEV SSIM and tile-STDDEV SSIM STDDEV.

### 3. DETERMINING SCANNED PAGE ORIENTATION

#### 3.1 Introduction

Recently, with the trend of the paperless office, lots of documents are being scanned every day. Also, the need to preserve old ancient books or other materials requires a lot of scanning. Moreover, correct orientation is very important before further processing, like optical character recognition (OCR). When scanning, the page can be in one of the four orientations: 0, 90, 180, and 270 degrees. Fig. 3.1 shows one page in all four orientations mentioned above. Sometimes it is difficult or even impossible to know all pages' orientations while scanning. Manually aligning all pages can be tedious and very time-consuming. As a result, it would be great if the scanner can automatically detect the pages' orientations and align them accordingly during the scanning process. Since the page can be in portrait or landscape mode, as shown in Fig. 3.2, the scanner should be able to work well in either of the two modes. Further, since this demand is worldwide, it would be best if the scanner can detect the orientation of pages in various language scripts, such as Roman, Chinese, Devanagari, Japanese, and Korean. Furthermore, for some documents like annual reports which contain a lot of numbers, we also need the scanner to detect their orientations correctly.

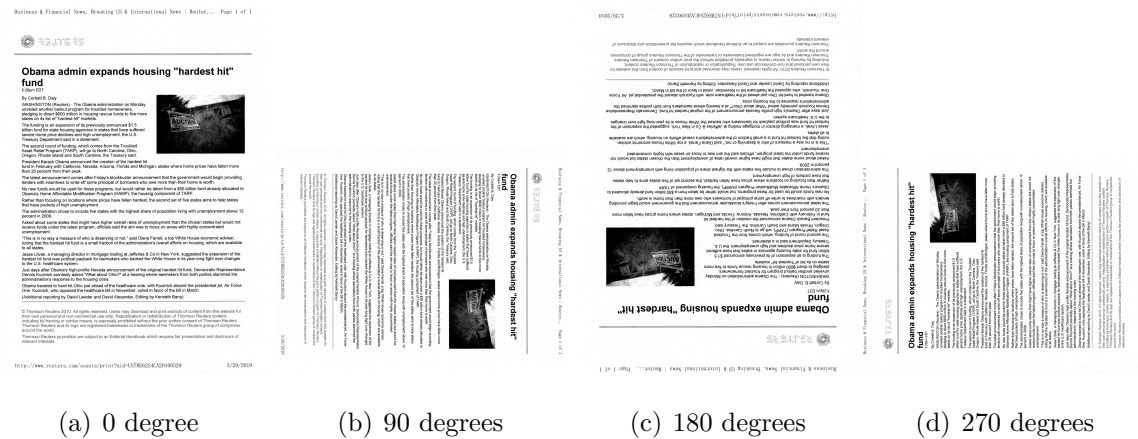
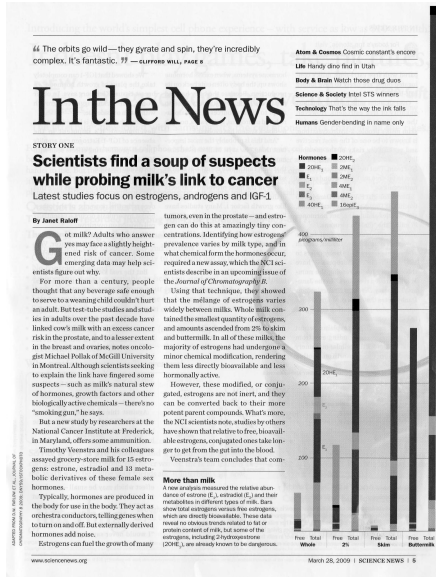


Figure 3.1. An example of one page in four orientations.



### 3.2 Related work

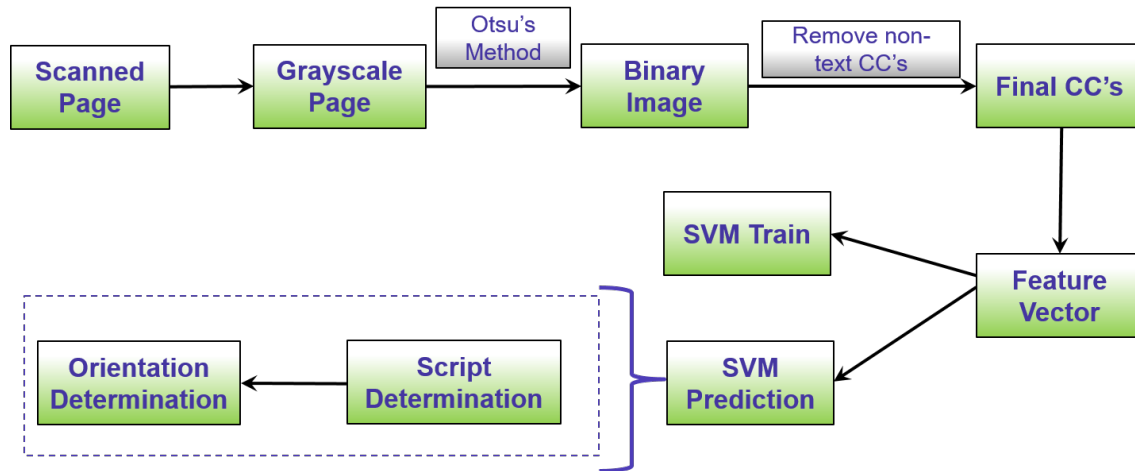
Several methods have been proposed to detect the orientation of document pages. Guo et al. [70] used three vertical component runs to get a 96-dimensional feature vector, which was then fed into an SVM to determine the text pages' Up/Down orientations. van Beusekom et al. [71] proposed a model to find the text line, the number of ascenders and descenders to determine one page's orientation. They tested their model on Batin script and achieved an overall accuracy of 98.8% on all orientations. They also tested the method on Japanese script but only got an accuracy of 85.7%. Roy et al. [72] proposed an algorithm to detect one page's up/down orientation based on text-asymmetry ratios computed from strip-based projection files. Their algorithm also needs to find the ascenders and descenders based on text lines. But for some pages written in traditional Chinese, where text characters are written vertically, these text line based methods might not work well. And also for some Asian scripts like Chinese and Japanese, there are no obvious ascenders or descenders. Fan et al. [73] invented a method that utilized the OCR method to find characters of 'i' or 'T' to determine the page's orientation. Ghosh et al. [74] proposed a method to identify the script and orientation of 11 official scripts in India. Their feature vector is composed of a reservoir area, a white hole area, and horizontal and vertical white-black transitions. They also designed a complex hierarchy to detect the script. Their method achieved fairly high accuracies; but it is specially designed for Indian languages. Rashid et al. [75] proposed to use a convolutional neural network (CNN) to identify scripts. They achieved above 95% recognition accuracy at the connected component level on datasets of Greek-Latin, Arabic-Latin, and Antiqua-Fraktur documents. Their CNN contains two convolutional layers with four and eight feature maps, followed by two sub-sampling layers. For the three script pairs, they achieved an accuracy of 98.40%, 95.61%, and 96.61%, respectively. But they did not try detecting the orientation.

Lu et al. [76] proposed an algorithm to detect document images' up/down orientation through document vectorization. They also used the same feature vector to detect page script. The idea is to find a feature vector vertically on a text character, and then convert it into a vector. Orientation and script detection of a document image are determined based on

distances between the detected document vector and pre-constructed vector templates. The algorithm was tested on Chinese, Roman, Arabic, and Chinese scripts and achieved good results. Based on their work, Jain et al. [77] developed an algorithm that can detect the four orientations mentioned above and multiple scripts using an artificial neural network. In their algorithm, they constructed features from the horizontal direction as well as the vertical direction. They also introduced foreground pixel densities as part of the feature vector. Their method achieved fairly good accuracies on Chinese, Japanese, and Korean, but performed rather badly on Devanagari, Numeral, and some Roman scripts. Our work is mainly based on Jain’s algorithm. While keeping their features, we also introduce features from the side background areas in a text character’s bounding box. Moreover, we develop new methods to rule out non-text connected components that might affect detection accuracy.

### 3.3 Methodology

#### 3.3.1 Overall Algorithm



**Figure 3.3.** Overall algorithm workflow.

The algorithm workflow is shown in Fig. 3.3. A document page is first scanned into a color image, and then transformed into a grayscale image. Then Otsu’s binarization is applied to get a binary image. After connected components (CCs) labeling, we apply methods to remove non-text CCs. Finally, a feature vector is extracted and fed into an SVM to do

training and prediction. After training, we save all the parameters into a self-defined header file. In prediction, the algorithm determines the image's script first, and then the orientation based on these header files.

### **3.3.2 Image Binarization**

The image is binarized based on Otsu's method [1]. We choose this method because it is fast and effective with document images. The algorithm assumes that the image contains two classes of pixels following a bi-modal histogram (foreground pixels and background pixels). It computes the histogram and probabilities of each intensity level, steps through all possible thresholds from the lowest to highest intensity level, and calculates each threshold's intra-class variance. The optimum threshold separating the two classes is the one with minimal intra-class variance. In this case, because the sum of pairwise squared distances is constant, their inter-class variance is maximal.

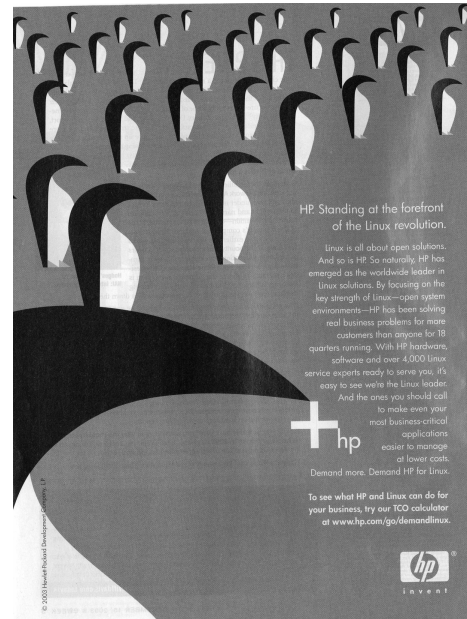
In order to get better local details, the image is equally separated into 16 ( $4 \times 4$ ) sub-images and Otsu's method is applied to each sub-image independently. The foreground and background areas are determined based on the assumption that the background area always constitutes a larger portion than the foreground area.

6

craft legacies of skill, self-discipline, and professional pride.<sup>13</sup> Whatever their historical origins—and there is a major debate over this—the districts as they had come to function by the end of the 1980s were in fact new social constructions.<sup>14</sup>

The distinctive elements in the configuration of the industrial districts are quite different from the socio-economic relationships between the old small-scale firms and their workers and the large firms whose dependent subcontractors they had been. Often the distinctive district configuration appeared in the wake of the break-up of a large firm or firms in the region.<sup>15</sup> Yet despite differences, all districts display similarities along three dimensions. First, within the districts there is a *division of labor* among firms, which promotes high levels of flexibility and productivity. Because firms within the districts often specialize in a one phase of the production process and through their subcontracting networks aggregate orders from several other local firms, they are able to invest in new capital equipment and amortize rapidly these investments. Flexible relations among local firms are not mirrored in workplace practices within them. Instead, because of the specialization in phases of production by district firms, work is often organized in highly specialized and narrow tasks, conducted by long-term and highly skilled employees. This, too, enhances the productivity of district-based firms.

A second feature of the districts is a distinctive *milieu* that includes the local institutional infrastructure (i.e., local banks, trade associations, training institutes and collaborative research and development facilities) as well as more “cultural” attributes and practices (i.e., craft traditions, “trust” among firms and between workers and managers, class mobility, etc.). A final feature underlying



(a)

(b)

6

craft legacies of skill, self-discipline, and professional pride.<sup>13</sup> Whatever their historical origins—and there is a major debate over this—the districts as they had come to function by the end of the 1980s were in fact new social constructions.<sup>14</sup>

The distinctive elements in the configuration of the industrial districts are quite different from the socio-economic relationships between the old small-scale firms and their workers and the large firms whose dependent subcontractors they had been. Often the distinctive district configuration appeared in the wake of the break-up of a large firm or firms in the region.<sup>15</sup> Yet despite differences, all districts display similarities along three dimensions. First, within the districts there is a *division of labor* among firms, which promotes high levels of flexibility and productivity. Because firms within the districts often specialize in a one phase of the production process and through their subcontracting networks aggregate orders from several other local firms, they are able to invest in new capital equipment and amortize rapidly these investments. Flexible relations among local firms are not mirrored in workplace practices within them. Instead, because of the specialization in phases of production by district firms, work is often organized in highly specialized and narrow tasks, conducted by long-term and highly skilled employees. This, too, enhances the productivity of district-based firms.

A second feature of the districts is a distinctive *milieu* that includes the local institutional infrastructure (i.e., local banks, trade associations, training institutes and collaborative research and development facilities) as well as more “cultural” attributes and practices (i.e., craft traditions, “trust” among firms and between workers and managers, class mobility, etc.). A final feature underlying



(c)

(d)

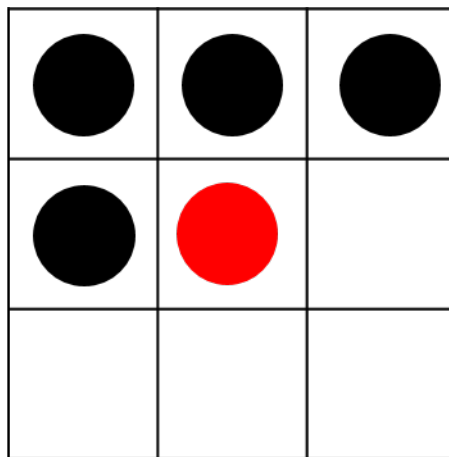
**Figure 3.4.** Examples of gray-scale images and their binary images after applying Otsu’s algorithm. (a)-(b) are the gray-scale images, and (c)-(d) are binary images generated by Otsu’s algorithm

Fig. 3.4 shows two grayscale images and their binary images after applying Otsu’s algorithm. We can see that the algorithm can identify the texts whether they are inverted or not. For an image that contains only text characters, like (a), its binary output also contains only text characters. But for images like (b), which contains large portion of pictures and drawings, its binary image contains a lot of non-text CCs. These non-text CCs would have a bad effect on the document’s feature vector, and may lead to unknown results. So we need to get rid of them in the connected components analysis part.

### 3.3.3 Connected Components Labeling

After obtaining the binary image, our next step is to apply connected components labeling to the whole image. Its input is a binary image and the output is a symbolic image in which the label assigned to each pixel is an integer uniquely identifying the connected component to which that pixel belongs [78]. Connectivity is usually determined by 4-connected neighborhood or 8-connected neighborhood [79]. In order to better deal with Numeral images whose pixels might only be connected in the diagonal or anti-diagonal direction, we chose 8-connected neighborhood to do connected components labeling.

Fig. 3.5 shows an example of 8-connectivity. For the center pixel examined (color red), its West, North-West, North, and North-East pixels are considered as its neighbors.



**Figure 3.5.** 8-connectivity. The middle pixel in the second row is the pixel under study. The 8-connectivity examines pixels in its West, North-West, North, and North-East pixels in connected components labeling.

Based on the number of passes an algorithm goes through the image, connected components algorithms can be categorized into one-pass and two-pass algorithms. The one-pass algorithm goes through the image only once. It is based on graph traversal methods in graph theory. Once the first pixel of a CC is found, all pixels of that CC are labeled before going to the next CC [80], [81]. This method always requires a linked-list to implement. The two-pass algorithm iterates through an image twice [82]: the first pass is to assign temporary labels and record equivalences; the second pass replaces each temporary label by the smallest label of its equivalence class. Since we want an algorithm that scans the image in raster order, we choose the two-pass connected components labeling method and use a stack to store equivalent labels. The process is as follows [83]:

First pass:

- (1) Iterate through each pixel by row, then by column;
- (2) If the pixel is not the background, check its neighbor pixels. If there are no foreground pixels in them, give the pixel examined a unique label and continue. If there are at least one foreground pixels in them, find the neighbor with the smallest label, assign that label to the pixel, and store the equivalence between the neighboring pixels.

Second pass:

- (1) Iterate through each pixel by row, then by column;
- (2) Relabel the pixel with the lowest neighboring label, if it is foreground.

After the second pass, we will have an image map with all CCs with different labels.

### 3.3.4 Connected Components Analysis

Since our feature vector is extracted from text characters, in this part we will remove non-text CCs and keep text CCs.

A document page may contain pictures, tables, or other non-text areas. Normally, some of them will stay as foreground pixels after Otsu's algorithm. In this work, several constraints are introduced to remove non-text areas, based on their characteristics that are distinct from texts.

## Size Limit

Since the sizes of text characters are usually in a small range, we can remove very large or small CCs by setting limits on their width and height. Explicitly, the size limits are (all parameters are determined empirically):

(a) Lower limits. Among width and height, one must be larger than  $0.01 \times \text{resolution}$  and the other must be larger than  $0.03 \times \text{resolution}$ . We set two thresholds because some text CCs have different lengths or widths.

(b) Upper limits. We set hard and dynamic limits on maximum size of CCs. For hard limits, one CC's width should not be larger than  $0.45555 \times \text{image width}$ . Similarly, its height should not be larger than  $0.45555 \times \text{image height}$ . For dynamic limits, a CC's size must not exceed  $2.775 \times \text{average of (width + height)}$ .

## Density Limit

The density of a CC, defined as the ratio of *foreground pixel number / bounding box area*, should be less than 1. This rules out blocks which have only foreground pixels, which are unlikely to be text characters.

Also, as to some CC's which have a very large ratio of *foreground / bounding box area*, we reverse them since their foreground and background pixels are likely to be inverted. But in order to speed the algorithm, this method is only applied when there are not enough CC's for feature vector extraction. Here enough means at least 20 text CC's. And the threshold to apply this method is empirically set to be 0.725. Note that after doing the contrast reversal, we need re-do connected components labeling to the resulting image.



**Figure 3.6.** An example of a CC with large foreground-background ratio. Here, the white pixels represent background, and the black pixels represent the foreground. We can see that the characters here are classified to be background.

### Transition Limit

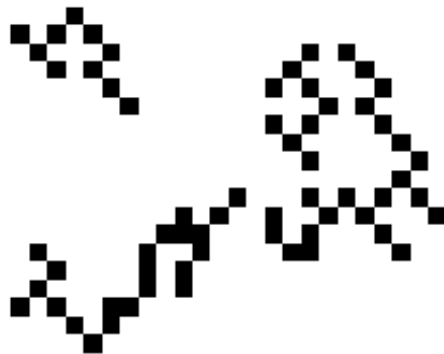
Normally, a text character in any script should contain both foreground and background pixels. Usually, the number of foreground-background transitions is under a certain threshold. So we can employ this to rule out non-text CCs. This can also be used to eliminate text CCs which have multiple characters bonded together when the scanning resolution is low or the page has low quality.

For a connected component, imagine that there is a line passing through it vertically or horizontally. We call it a transition if the line encounters a foreground pixel from background pixels.

In this work, we count the number of transitions through the center of a CC both horizontally and vertically. Empirically, the limit is set to be eight in both directions. And a CC is to be removed if the transition number in either direction exceeds the limit.

Besides, in document images where text characters are written on a large picture, like a magazine cover, the picture is often rendered with halftone patterns. As a result, when we apply Otsu's method, we can often find CCs with a halftone pattern, and these CCs often meet the size limits, as shown in Fig. 3.7. Although applying a low-pass filter to the whole image might be the easiest to remove them, it can also blur the edges [84], making some text

characters bound together when the scanning resolution is low. So a new method is required to remove these CCs.



**Figure 3.7.** Examples of CCs rendered with a halftone. We can see a lot of interleaved pixels in these CCs after binarization; and they cannot be identified by the size limits.

Considering the fact that in CCs rendered with a halftone pattern, foreground and background pixels are always interleaved, a notion of horizontal or vertical transition density is advocated. The way to calculate horizontal transition density is: first horizontally count the total number of transitions along all rows in the bounding box, then divide this number by bounding box height. The vertical transition density is calculated similarly, except that the transitions are counted along the vertical direction, and the denominator is the bounding box width.

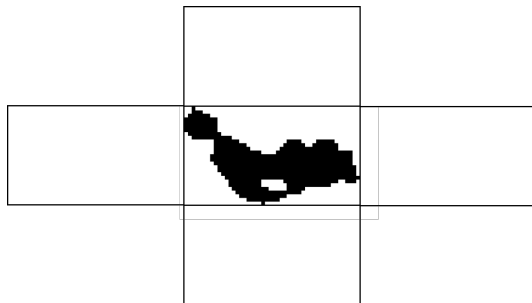
Empirically, the thresholds are set to be  $1.5$  in both directions. Any CCs with either transition density larger than  $1.5$  will be removed.

Also, considering the fact that some halftone-rendered CCs have only one pixel at each bounding box edge, while text characters have multiple foreground pixels on at least one edge, we can also eliminate these CCs.

## Isolate CCs

Since text characters always occur along a text line or in a paragraph, CCs which appear isolated can be considered as non-text and will be removed. Here, we call a CC “isolated” if no other foreground pixels exist within a bounding box area from the top, bottom, left and

right of the connected components. Fig. 3.8 shows an example of an isolated CC. Each of the four boxes to the left, right, bottom and top has the same width and height as those of the center bounding box. We can see that it is isolated as there are no foreground pixels in the four surrounding boxes.



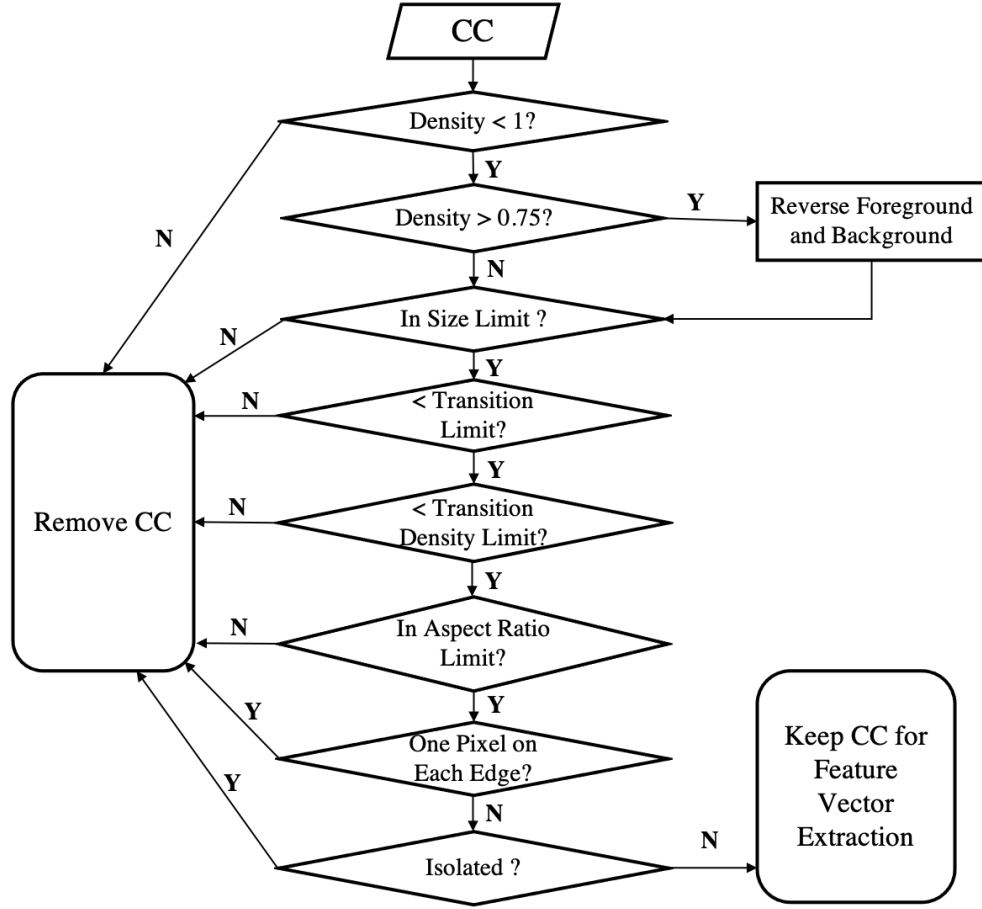
**Figure 3.8.** Examples of an isolated CC. We can see that there are no foreground pixels in one bounding box area to its left, right, upper, and lower directions.

## Aspect Ratio

Since text characters' aspect ratios are within a certain range, we set the ratio to be within the range of  $\frac{1}{6}$  to 6 to rule out non-text connected components.

## CC Analysis Hierarchy

Fig. 3.9 shows the hierarchy for CC analysis. For a CC, we first check if it has a density less than one, and will reverse the foreground and background if the density value is larger than 0.75. Then we check if it satisfies the size limit. If the answer is 'no', we remove it, otherwise we test it with transition limit. If it passes, the CC is then tested with the aspect ratio limit. If it is in the range of the smallest and largest aspect ratios, we check if it only has one pixel on every edge of its bounding box, followed by testing if the transition density is less than the threshold. Finally, we test if it is isolated. A CC is called text CC if it passes all the tests, and it can be used for feature vector extraction.



**Figure 3.9.** CC analysis hierarchy.

## CC Analysis Result

Fig. 3.10 shows two binary images after Otsu's algorithm and their final images after connected component analysis (CCA). From (a) and (c), we can see that CCA keeps almost all the text characters in the binary document. But in cases like (b), where the image contains a lot of non-text CCs, CCA removes most of them while keeping most of the text characters; and these will be enough to do feature vector extraction.

craft legacies of skill, self-discipline, and professional pride.<sup>13</sup> Whatever their historical origins—and there is a major debate over this—the districts as they had come to function by the end of the 1980s were in fact new social constructions.<sup>14</sup>

The distinctive elements in the configuration of the industrial districts are quite different from the socio-economic relationships between the old small-scale firms and their workers and the large firms whose dependent subcontractors they had been. Often the distinctive district configuration appeared in the wake of the break-up of a large firm or firms in the region.<sup>15</sup> Yet despite differences, all districts display similarities along three dimensions. First, within the districts there is a *division of labor* among firms, which promotes high levels of flexibility and productivity. Because firms within the districts often specialize in a one phase of the production process and through their subcontracting networks aggregate orders from several other local firms, they are able to invest in new capital equipment and amortize rapidly these investments. Flexible relations among local firms are not mirrored in workplace practices within them. Instead, because of the specialization in phases of production by district firms, work is often organized in highly specialized and narrow tasks, conducted by long-term and highly skilled employees. This, too, enhances the productivity of district-based firms.

A second feature of the districts is a distinctive *milieu* that includes the local institutional infrastructure (i.e., local banks, trade associations, training institutes and collaborative research and development facilities) as well as more “cultural” attributes and practices (i.e., craft traditions, “trust” among firms and between workers and managers, class mobility, etc.). A final feature underlying



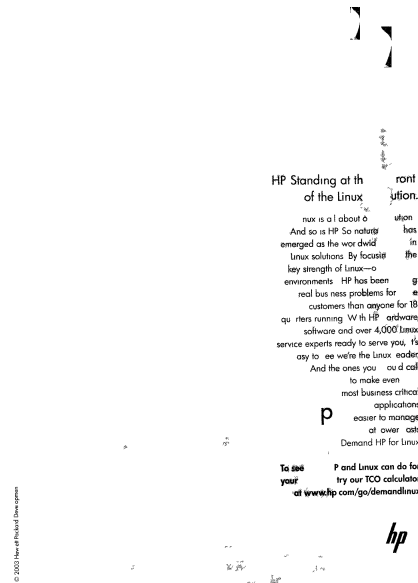
(a)

(b)

craft legacies of skill, self-discipline, and professional pride.<sup>13</sup> Whatever their historical origins—and there is a major debate over this—the districts as they had come to function by the end of the 1980s were in fact new social constructions.<sup>14</sup>

The distinctive elements in the configuration of the industrial districts are quite different from the socio-economic relationships between the old small-scale firms and their workers and the large firms whose dependent subcontractors they had been. Often the distinctive district configuration appeared in the wake of the break-up of a large firm or firms in the region.<sup>15</sup> Yet despite differences, all districts display similarities along three dimensions. First, within the districts there is a *division of labor* among firms, which promotes high levels of flexibility and productivity. Because firms within the districts often specialize in a one phase of the production process and through their subcontracting networks aggregate orders from several other local firms, they are able to invest in new capital equipment and amortize rapidly these investments. Flexible relations among local firms are not mirrored in workplace practices within them. Instead, because of the specialization in phases of production by district firms, work is often organized in highly specialized and narrow tasks, conducted by long-term and highly skilled employees. This, too, enhances the productivity of district-based firms.

A second feature of the districts is a distinctive *milieu* that includes the local institutional infrastructure (i.e., local banks, trade associations, training institutes and collaborative research and development facilities) as well as more “cultural” attributes and practices (i.e., craft traditions, “trust” among firms and between workers and managers, class mobility, etc.). A final feature underlying



(c)

(d)

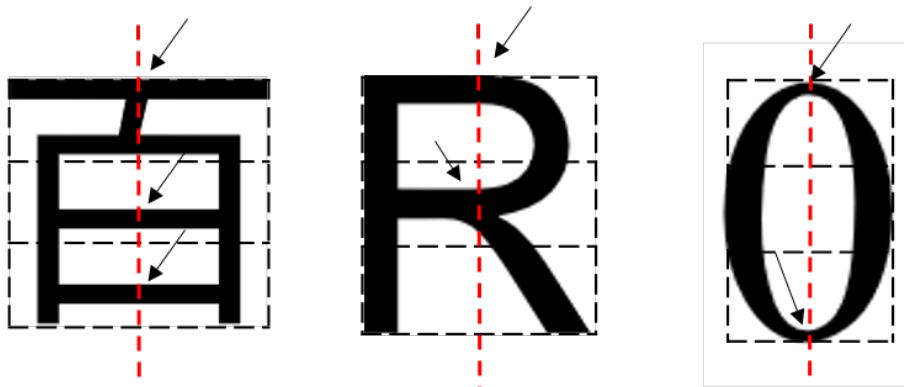
**Figure 3.10.** Examples of binary images and their final images after connected components analysis. (a)-(b) are the binary images after Otsu’s algorithm, and (c)-(d) are final images generated after connected components analysis.

### 3.4 Feature Vector Extraction

After the text CCs are found, our next step is to extract the feature vector representing the document image. The feature vector is a concatenation of four vectors: vertical document vector (VDV), horizontal document vector (HDV), zonal document vector (ZDV), and profile document vector (PDV). These four vectors are achieved by averaging four features of every text CC: vertical component run (VCR), horizontal component run (HCR), zonal document run (ZDR) [77], and profile component run (PCR), respectively. In this section, we will talk about how to get these features.

#### 3.4.1 Vertical Document Vector

The method to calculate vertical component run (VCR) is first described by Lu et al. [76] to detect script like Arabic, Chinese, Korean and Roman. The idea is to separate the bounding box into three equal areas, called top zone, middle zone, and bottom zone. Then, we assume that there is a virtual scanning line passing through the center of a CC. We define a transition as the event in which the line passes to a foreground pixel from background pixel. Finally, we record the number of transitions and where the transitions appear, as shown in Fig. 3.11.



**Figure 3.11.** Examples of vertical transitions on connected components. The bounding boxes are separated into three equal zones vertically. The vertical dashed line in the middle is the imaginary line through the CC's center. And the arrows in the picture show locations of transitions.

If the line starts from a foreground pixel on the edge of bounding box, we still record it as a transition. So from Fig. 3.11 We can see that the first Chinese character has in total three vertical transitions, one in top zone, one in middle zone, and one in bottom zone; the number 0 has in total two vertical transitions, one in top zone and one in bottom zone; the character R has in total two vertical transitions, one in top zone and one in middle zone.

In practice, we assume that the number of transitions in a CC cannot exceed eight. The upper limit 8 is set because most scripts under study have no more than 8 vertical transitions. If indeed a character of more than eight vertical transitions is found, it is statistically insignificant. The extra vertical transitions can be ignored without affecting our results [76]. The steps to get a CC's VCR are:

(1) Identify its bounding box, and vertically divide it into three equidistant zones: the top zone  $T$  on the top, the middle zone  $M$  in the middle, and the bottom zone  $B$  at the bottom.

(2) Find the horizontal center of the CC, and draw a vertical line through it.

(3) We have eight elements,  $N_1, N_2, \dots, N_8$ , to record the total number of transitions in a CC. If the number is  $n$ . Set the index  $N_n$  to be 1, set the others to 0.

(4) Have another eight elements,  $T_1, T_2, \dots, T_8$ , to record the occurrences of transitions in the top zone. For example, if only the 1<sup>st</sup> transition is in the upper part, we put  $T_1$  be 1, the others to 0; if both the 1<sup>st</sup> and 2<sup>nd</sup> transitions are in the upper part, we put  $T_1$  and  $T_2$  be 1, the others to 0; if all  $n$  transitions occur in the top zone, we put  $T_1, \dots, T_n$  to be 1, the others to 0; if there are no transitions in the top zone, we keep  $T_1, T_2, \dots, T_8$  to be 0.

(5) Form eight more elements  $M_1, M_2, \dots, M_8$  for the middle zone, and  $B_1, B_2, \dots, B_8$  for the bottom zone. Repeat step (4) for these two zones.

(6) Combining all elements in (3) to (5) we will have a VCR that has 32 dimensions:

$$VCR = [N_1, N_2, \dots, N_8, T_1, T_2, \dots, T_8, M_1, M_2, \dots, M_8, B_1, B_2, \dots, B_8] \quad (3.1)$$

And we have  $N_i = 1$  if

$$i = \sum_{j=1}^8 T_j + \sum_{j=1}^8 M_j + \sum_{j=1}^8 B_j \quad (3.2)$$

Referring to the three examples mentioned in Fig. 3.11, for the first Chinese character, its VCR is [00100000 10000000 01000000 00100000]. Because it has in total three vertical transitions, we set the third element of the first eight elements the value 1. The top zone has its first transition, so we put 1 at the beginning of the second eight elements. The middle zone has the second transition, so we put the value 1 at the second place of the third eight elements. And the bottom zone has its third transition, so the third place of the last eight elements has the value 1, and others 0. Similarly, for the number 0, its VCR is [01000000 10000000 01000000 00000000], since it has in total two vertical transitions and they appear in the upper and middle zones, respectively, while no transitions appear in the bottom zone. For the character R, its VCR is [01000000 10000000 00000000 01000000], since it has in total two vertical transitions and they appear in the upper and bottom zones, respectively. And there are no transitions in the middle zone.

Normally, a document contains a large number of text characters. Each of them has its own VCR. Next we calculate the sum of them as a feature vector for the whole page. The feature vector is called vertical document vector (VDV); and the formula to get it is as follows:

$$VDV = \sum_{i=1}^N VCR_i \quad (3.3)$$

where  $N$  is the total number of CCs in the document under study.  $VCR_i$  is the VCR of  $i$ -th CC.

	$N_1 - N_8$								$T_1 - T_8$								$M_1 - M_8$								$B_1 - B_8$									
E	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
N	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
G	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
I	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
N	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
E	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
E	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
R	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
<div><div></div><div><math>\Sigma</math></div><div></div></div>																																		
VDV	3	2	3	0	0	0	0	0	6	0	0	0	0	0	0	0	0	2	4	0	0	0	0	0	0	0	0	1	3	0	0	0	0	

**Figure 3.12.** Illustration of constructing the VDV from VCRs. Each row in the upper table shows VCRs of the character to the left side. The row in the bottom is the VDV, which is the the sum of all rows.

Fig. 3.12 illustrates an example to get the VDV from VCRs. Suppose a document has only one word “ENGINEER”; and each character is correctly segmented. Each character is converted into a VCR, which corresponds to one row in the upper table. So in total we have eight VCRs. The last row in Fig. 3.12 is the VDV of the document, which equals to the sum of all VCRs in the upper table.

Before using the VDV for training an SVM, we need to normalize it to reduce the effect of document length. The normalized value  $VDVnorm$  is calculated using the following equation [76]:

$$VDVnorm_i = \begin{cases} VDV_i \cdot 100 / numCC & \text{if } i \leq 8 \\ VDV_i \cdot 100 / numVertTransitions & \text{if } i > 8 \end{cases} \quad (3.4)$$

where

$$numCC = \sum_{i=1}^8 VDV_i \quad (3.5)$$

and

$$numVertTransitions = \sum_{i=9}^{32} VDV_i \quad (3.6)$$

Here,  $VDVnorm_i$  is the  $i$ -th element in the normalized vector  $VDVnorm$ .  $VDV_i$  is the  $i$ -th element in the vector  $VDV$ .  $numCC$  is the total number of CCs in the document under study, and  $numVertTransitions$  is the total number of vertical transitions in the document. Through the above equations, we normalize the VDV to the range of  $[0, 100]$ .

### 3.4.2 Horizontal Document Vector

Based on the idea of VCR and VDV, the idea of horizontal component run (HCR) and horizontal document vector (HDV) are introduced to detect document pages in 90 or 270 degree orientations. The steps to get the HCR of a CC are:

- (1) Identify the its bounding box, and horizontally divide it into three equidistant zones: the left zone  $L$ , the center zone  $C$  in the middle, and the right zone  $R$ .
- (2) Find the center of the CC, and draw a horizontal line through it.
- (3) We have eight elements,  $H_1, H_2, \dots, H_8$ , to record the total number of transitions in a CC. If the number is  $n$ , we set the  $H_n$  index to be 1, others to 0.
- (4) Have another eight elements  $L_1, L_2, \dots, L_8$ , to record the occurrences of transitions in the left zone. Then, form eight more elements  $C_1, C_2, \dots, C_8$  for the center zone, and  $R_1, R_2, \dots, R_8$  for the right zone, to record the occurrences of transitions in these two zones, respectively.
- (5) Combine all elements in (3) to (4) to get an HCR that has 32 dimensions:

$$VCR = [H_1, H_2, \dots, H_8, L_1, L_2, \dots, L_8, C_1, C_2, \dots, C_8, R_1, R_2, \dots, R_8] \quad (3.7)$$

And we have  $H_i = 1$  if

$$i = \sum_{j=1}^8 L_j + \sum_{j=1}^8 C_j + \sum_{j=1}^8 R_j \quad (3.8)$$

Fig. 3.13 shows examples of getting an HCR and HDV. On the top row are three characters with three horizontal zones and center horizontal lines shown on them. We can see that the first Chinese character has in total three horizontal transitions, one in the top left,

one in the center zone, and one in the right zone; the character p has in total two vertical transitions, one in the left zone and one in the right zone; the number 7 has in total one vertical transition in the center zone. Their HCRs are listed in the table in the middle. Summing them up, we get the HDV at the bottom.

	$H_1 - H_8$								$L_1 - L_8$								$C_1 - C_8$								$R_1 - R_8$							
山	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
p	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\Sigma$																															
HDV	1	1	1	0	0	0	0	0	2	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0

**Figure 3.13.** Examples of HCR and HDV of connected components. In the top row are three characters. Each of their bounding boxes is separated horizontally into three equal zones. And the horizontal line in the middle travels through centers of these characters. The table in the middle shows HCRs of the three characters in the top row. The last row is the HDV, which is the sum of all rows of the middle table.

Similar to what we did to get VDV, we calculate the sum of all HCRs of a document as a feature vector for the whole page, and call it the horizontal document vector (HDV). We also need to normalize the HDV to get rid of the effect of document length like Equations 3.4 - 3.6.

### 3.4.3 Zonal Density Vector

We also introduce the zonal density vector (ZDV) as part of the feature vector. The ZDV is obtained via the zonal density run (ZDR), which reflects the distribution of foreground pixel densities. The steps to get the ZDR are:

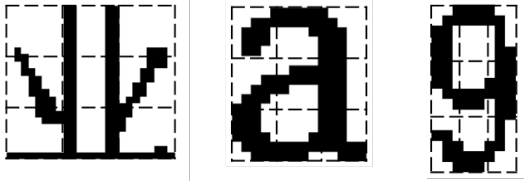
- (1) Divide a CC's bounding box area equally into 9 ( $3 \times 3$ ) small areas.

(2) For each small area, calculate its foreground pixel density ( $fd$ ) following this equation:

$$fd = \frac{numForePixel}{numAreaPixel} \cdot 100 \quad (3.9)$$

where  $numForePixel$  refers to the number of foreground pixels in a certain area, and  $numAreaPixel$  is the number of pixels in the area. Here we multiply by 100 to put the value to the range of  $[0, 100]$ . The small areas are iterated first row-wise, and then column-wise. Their  $fd$ s are denoted as  $fd_1, fd_2, \dots, fd_9$ .

(3) Concatenate all values calculated in (2); and we will get a ZDR with 9 elements.

									
	fd <sub>1</sub> – fd <sub>9</sub>								
业	2	49	6	27	49	27	26	56	24
a	46	46	44	36	42	56	70	40	66
9	62	37	48	54	37	84	41	39	35

**Figure 3.14.** Examples of ZDRs of connected components. Each of the character's bounding box in the top row is separated equally into three parts, and their pixel densities are listed in the table below the top row.

Fig. 3.14 shows ZDR examples of a Chinese character, the letter 'a' and the number '9'.

The ZDV for the document is formed by summing the ZDRs obtained from all CCs in the document. Since each document has a different number of text characters, we also need to normalize the ZDV using following equation:

$$ZDV_{norm} = \sum_{i=1}^N ZDR_i / N \quad (3.10)$$

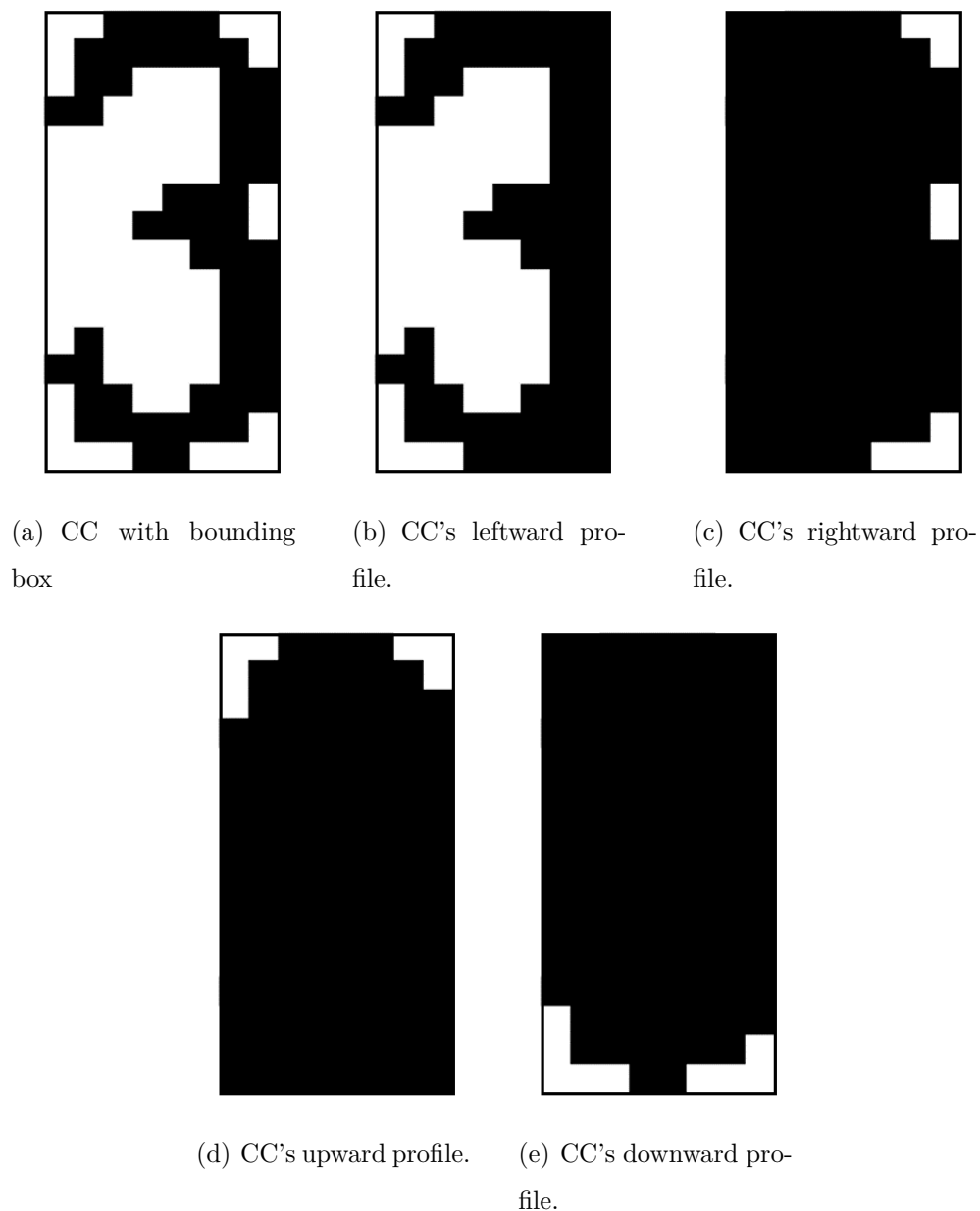
Here,  $N$  is the total number of CCs in the document, and  $ZDR_i$  is the ZDR of the  $i$ -th CC.

Notice that  $ZDV_{norm}$  is also in range  $[0, 100]$ .

### 3.4.4 Profile Component Run

So far the features found from a single CC are mainly focused on the foreground pixels in its bounding box. The background areas surrounding the foreground pixels, however, also contain important information of the character [85]. And this information is very important for relatively simple-structured scripts like Numeral. As a result, the profile component run (PCR) of a CC and profile document vector (PDV) of a document are utilized to better detect the document's script and orientation.

The four side profiles are the four surrounding background areas in a CC's bounding box, namely, the leftward, upward, rightward, and downward background areas, as shown in Fig. 3.15. While black pixels represent the foreground pixels, the four-side profiles are the white pixel areas seen from the left, right, up, and down directions until they hit a foreground pixel.



**Figure 3.15.** Example of a CC's four-side profiles. Here, the black pixels show the foreground, and the white pixels are the background.

The PCR is obtained from the four side profiles. The steps to get the PCR from a CC are:

- (1) Identify the CC's bounding box.

(2) On the left edge, identify 5 points on each edge of the bounding box, namely, the two vertices, the middle point, and two points whose distance from the nearest vertex is  $1/6$  of the length of the left edge.

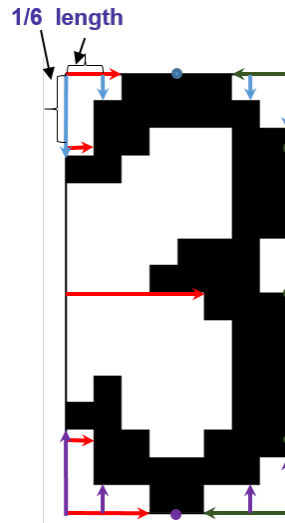
(3) For each point found in (2), count the number of consecutive background pixels before hitting a foreground pixel along the line that is perpendicular to the left edge.

(4) For each number obtained in (3), multiply it by 100, and then divide it by the bounding box's width, to remove the effect of the CC's bounding box size. Denote them as  $Pl_1, Pl_2, \dots, Pl_5$ .

(5) Repeat steps (2) - (4) for other three edge. Note that the denominator for the right edge is also the bounding box's width, but for the top and bottom edges, the denominator is the bounding box's height. So we get  $Pr_1, Pr_2, \dots, Pr_5$  for the right edge,  $Pt_1, Pt_2, \dots, Pt_5$  for the top edge, and  $Pb_1, Pb_2, \dots, Pb_5$  for the bottom edge.

(6) Combining all the values obtained in (4) and (5), we will get a PCR for the CC, which is

$$PCR = [Pl_1, Pl_2, \dots, Pl_5, Pr_1, Pr_2, \dots, Pr_5, Pt_1, Pt_2, \dots, Pt_5, Pb_1, Pb_2, \dots, Pb_5] \quad (3.11)$$



**Figure 3.16.** Examples of PCR of a CC. The arrow lines in the figure show the distance from the bounding box edges to the foreground pixels in the four side profiles.

The PDV is achieved by summing all PCRs of the document. In order to reduce the effect of document length, we also need to normalize it

$$PDV_{norm} = \sum_{i=1}^N PCR_i / N \quad (3.12)$$

Here,  $N$  is the total number of CCs in the document, and  $PCR_i$  is the PCR of the  $i$ -th CC.

Notice that  $PDV_{norm}$  is also in the range of  $[0, 100]$ .

Concatenating all features from Sections 3.4.1 to 3.4.4, we have a 93-dimensional feature vector containing VDV, HDV, ZDV and PDV that represent one single document page. Then, the feature vector will be used in a support vector machine for training and prediction.

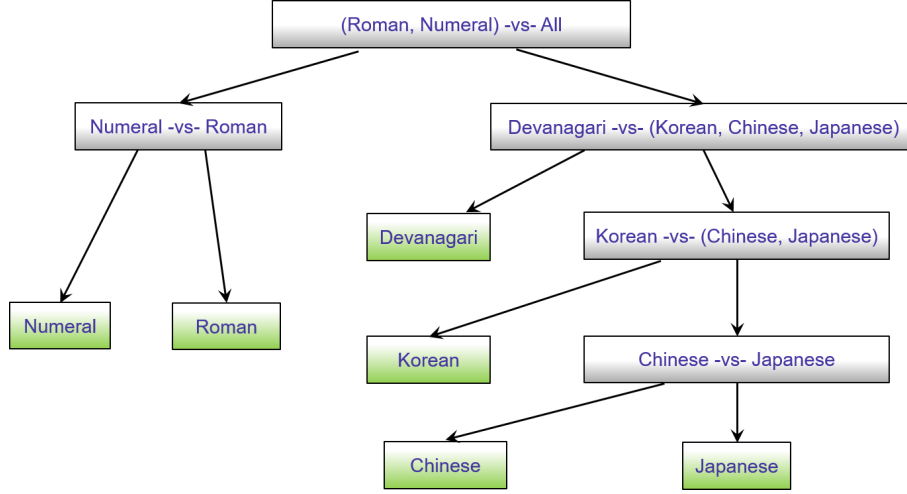
### 3.4.5 Support Vector Machines

After the feature vector is obtained, we employ a support vector machine (SVM) to classify the document into different script and orientation.

SVM is useful for data classification. It maps the data into a higher dimensional space, and finds a linear separating hyperplane with the maximal margin in it [47]. In this project, we used the LIBSVM [52] and radial basis function (RBF) kernel to do the training, and we use the parameters found to build a header file used for prediction.

### 3.4.6 Script Detection Hierarchy

We use support vector machine (SVM) to detect the script before the orientation. So a script detection hierarchy is built (shown in Fig. 3.17). It is based on the similarity of different scripts. First, we treat Numeral and Roman as a group, and tell them apart from other scripts, which include Devanagari, Korean, Chinese, and Japanese. Then a model is developed to distinguish Numeral from Roman. Since Devanagari is a lot different from Korean, Chinese, and Japanese, we distinguish it first. Then Korean is identified from Chinese and Japanese. Finally, Chinese and Japanese are distinguished from each other.



**Figure 3.17.** Script Detection Hierarchy.

## 3.5 Experimental Results

### 3.5.1 Data Collection

We collect our own data for training or testing. The collection process follows the routine below.

(1) Pdf files are found from the internet, and only pages with a considerable number of text characters are collected.

(2) Pages are printed using an HP LaserJet 500 color MFP M575.

(3) Pages are scanned using an HP Officejet Enterprise Color Flow MFP X585, and scanned at 300 dpi.

(4) For pages in other resolutions, 150 dpi and 200 dpi for example, we use the Unix “pamscale” [86] command with the default method “pixel mixing” to convert pages of 300 dpi into the resolutions mentioned above.

The scripts and their corresponding number of scanned images collected are shown in Table 3.1.

**Table 3.1.** All scripts and the number of pages

<b>Script</b>	<b>Page#</b>
Chinese	455
Devanagari	360
Japanese	378
Korean	438
Roman	2,476
Numeral	638
<b>Overall</b>	<b>4,745</b>

In this algorithm, Roman script contains English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. Their corresponding image numbers are listed in Table 3.2.

**Table 3.2.** Roman script and the number of pages

<b>Script</b>	<b>Page#</b>
English	324
French	315
German	271
Greek	408
Italian	263
Portuguese	300
Russian	296
Spanish	299
<b>Overall</b>	<b>2,476</b>

Note that pages we collected are all in 0 degree, and by simple rotations we can have images in 90, 180, and 270 degrees. So the total number of images in our data set is  $4,745 \times 4 = 18,980$ .

### 3.5.2 Test Results

We first use the whole data set as the training set. After getting the training parameters generated by SVM, we test their prediction accuracy on the whole data set. Table 3.3 shows the prediction results of script and orientation detection for all scripts.

**Table 3.3.** Training prediction accuracy of script and orientation for all scripts.

Script	Orientation detection accuracy(%)	Script detection accuracy(%)
Chinese	99.7	100.0
Devanagari	98.3	98.4
Japanese	100.0	100.0
Korean	99.9	99.8
Roman	99.7	99.8
Numeral	100.0	99.8
<b>Overall</b>	<b>99.7</b>	<b>99.7</b>

From Table 3.3 we can see that for the whole training set, our algorithm can achieve an accuracy of 99.7% in both orientation and script detection. It even gets 100% accuracy when detecting the orientations of Japanese and Numeral. We also achieve an accuracy of 99.7% in orientation and 99.8% in script detection for all Roman scripts. For each individual script, we have Table 3.4 showing their prediction accuracy.

**Table 3.4.** Training prediction accuracy of script and orientation for Roman scripts.

<b>Script</b>	<b>Orientation detection accuracy (%)</b>	<b>Script detection accuracy (%)</b>
English	100	99.8
French	99.8	100.0
German	99.8	100.0
Greek	99.4	99.8
Italian	99.3	99.5
Portuguese	99.2	99.6
Russian	100.0	99.7
Spanish	100.0	99.9
<b>Overall</b>	<b>99.7</b>	<b>99.8</b>

From Table 3.4 we can see that the algorithm has a good performance on all Roman scripts. It also achieves 100% in detecting the orientation of pages in Russian and Spanish.

After getting the prediction accuracy of the training set, we use 3-fold cross-validation to test the accuracy of our algorithm. Namely, we separate the whole data set into three equal-numbered folders, and use two of them for training, and one for testing. And we will switch folders used for training and testing. The accuracy is computed as the average of three testing accuracies. Images here are all in 200 dpi.

Table 3.5 shows the cross-validation result of script and orientation detection for all scripts. From this table we can see that for script detection, the overall accuracy for all scripts and four orientations is 98.2%. The best result is 99.1% with Japanese, while the lowest accuracy is 96.2% with Numeral. For orientation detection, the overall accuracy for all scripts of all four orientations is 99.2%. The best orientation accuracy is 99.8% with Japanese script, while the lowest accuracy is 98.3% with the numeral script.

**Table 3.5.** Cross-validation result of script and orientation detection for all scripts

<b>Script</b>	<b>Orientation detection accuracy (%)</b>	<b>Script detection accuracy (%)</b>
Chinese	99.0	96.8
Devanagari	99.4	99.1
Japanese	99.8	98.2
Korean	99.2	98.3
Roman	99.4	98.7
Numeral	98.3	96.2
<b>Overall</b>	<b>99.2</b>	<b>98.2</b>

The Roman script actually is comprised of eight scripts: English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. Its detection result is obtained by summing the detection results of all eight scripts. Table 3.6 shows the detailed detection results of these eight scripts. From it we can see that, although Greek and Russian look quite different from other scripts, they do get fairly good detection results, with both at 99.5% in orientation accuracy, and 99.1% and 99.2% in script accuracy, respectively. The lowest orientation accuracy happens with German, at 98.5%, and the lowest script accuracy happens with Portuguese at 98.3%. Spanish has the best orientation and script detection accuracies, with the values being 100% and 99.9%, respectively.

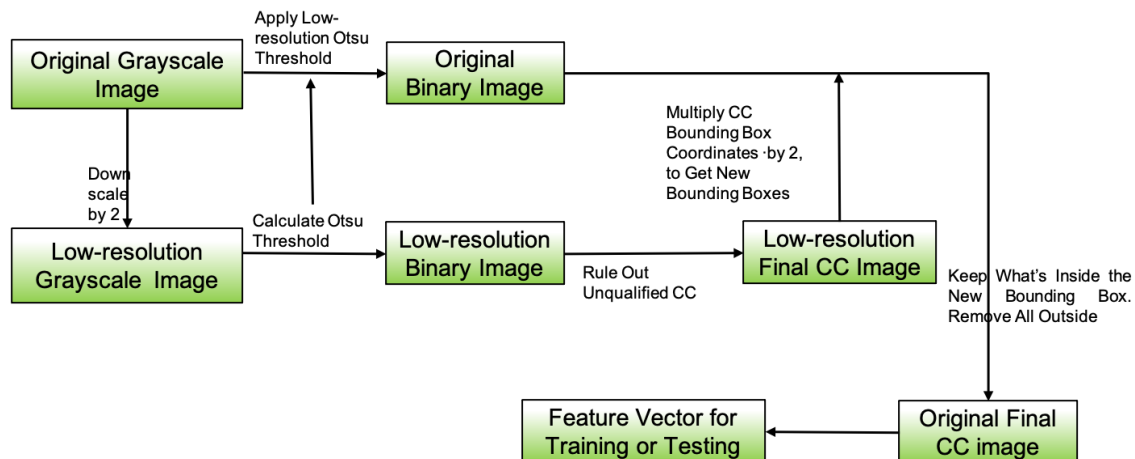
**Table 3.6.** Cross-validation result of script and orientation detection for Roman scripts

<b>Script</b>	<b>Orientation detection accuracy (%)</b>	<b>Script detection accuracy (%)</b>
English	99.5	98.9
French	99.6	99.8
German	98.5	98.9
Greek	99.5	99.1
Italian	99.4	99.4
Portuguese	99.1	98.3
Russian	99.5	99.2
Spanish	100	99.9
<b>Overall</b>	<b>99.4</b>	<b>98.7</b>

Experimental results shows that our algorithm is very fast, too. Its mean of processing time is 0.34 seconds, with a standard deviation of 0.07 seconds for all pages in our data set.

### 3.6 Low-resolution Connected Components Labeling Method

In Fig. 3.3, we used the original image to do the connected components labeling and analysis. If the image is in high resolution, 300 dpi, for example, the algorithm may need a lot of memory to store these images. And this may not work well with low-end printers. Therefore, a new version of the algorithm, ‘lowres-label (low resolution labeling)’ version, is proposed. It generates an image, which is half the resolution of the input image, and uses it for connected components labeling and analysis. The workflow of this algorithm to get the feature vector is shown in Fig. 3.18.



**Figure 3.18.** The lowres-label algorithm workflow to obtain feature vector.

As shown in Fig. 3.18, after the original color image is converted to grayscale image, we downscale the image by two and get the lowres grayscale image. Then Otsu’s method is applied to the lowres grayscale image and thresholds of each sub-image are calculated. Using the same thresholds we threshold both the original and lowres image and get their corresponding binary images, respectively. The connected components labeling is applied on the lowres binary image and rules are also conducted to get rid of non-text CC’s. In this way we can save a lot of memory, compared with using the image of the original resolution. After the final CC image in lower resolution is achieved, we map their bounding boxes’ coordinates to the original binary image, so we can get the final CC map in the original resolution. Finally, we get the feature vector that is used for training and testing.

**Table 3.7.** Memory usage of lowres-label and original algorithms

Resolution (DPI)	Lowres-label (MB)	Original (MB)
300	24.28	34.80
200	11.42	16.10
150	6.92	9.55

Table 3.7 shows the memory usage of the lowres-label and original algorithms. Here, the image is assumed to be standard letter size. From this table we can see that for images in

150, 200 or 300 dpi, the low-res algorithm can save approximately 30% of the total memory, compared with the original algorithm.

**Table 3.8.** Training orientation detection results of lowres-label and original algorithms

<b>Script</b>	<b>Lowres-label accuracy (%)</b>	<b>Original accuracy (%)</b>
Chinese	98.9	99.0
Devanagari	99.9	99.4
Japanese	99.6	99.8
Korean	98.5	99.2
Roman	99.0	99.4
Numeral	97.2	98.3
<b>Overall</b>	<b>98.8</b>	<b>99.2</b>

Table 3.8 shows the training orientation detection accuracy of the lowres-label algorithm and the original algorithm. From the table we can see that of all pages, the lowres-label achieves an accuracy of 98.8%, which is a little bit lower than the original algorithm’s 99.2%. Combining the result of Table 3.7, we can say that lowres-label algorithm is a good replacement on low-end scanners with limited memory.

### 3.7 Major Contributions

My major contributions for this project are as follows:

- Generate a large dataset containing 13 different scripts and 4,745 pages in each orientation.
- To the best of our knowledge, this is the first work to identify halftone patterns. We also introduce two new features: transition limit and transition density limit, to remove them.
- Introduce a brand new feature: profile component run, to record the background information in bounding boxes of connected components.

- Introduce brand new features: isolated connected components, and one pixel on each edge, to eliminate non-text connected components.
- Application of these new features has greatly increased the detection accuracy of simple numeral scripts.
- Application of these features has a high overall orientation detection accuracy in our dataset.

## 4. COLOR TABLE COMPRESSION

### 4.1 Introduction

Color Management plays a crucial role in reproduction and transformation of color information among various devices. Color look-up tables (LUT) embedded in devices provide color management systems with the information necessary to convert color data between native device color spaces and device-independent color spaces. But it also consumes much memory. The size of these color tables will also increase with finer sampling of the spaces and larger bit depths. Thus, a method to compress these LUTs is desirable for the purpose of conserving memory and storage in devices, and also reducing network traffic and delay.

Lossy and lossless compression are two forms of compression. In lossless compression, no digital difference exists between the original data and the reconstructed data from compressed files. In contrast, a portion of the original data is lost upon reconstruction of lossy compressed files [87]. Compression methods usually exploit two separate characteristics of data: irrelevance and redundancy. Its objective is to reduce the irrelevance and redundancy of the data in order to be able to store or transmit the data effectively. Irrelevance refers to information that does not need to be kept for the reconstruction. Redundancy refers to the repetitive or statistical dependencies in the data. Lossy methods exploit irrelevance, and lossless methods do not. Redundancy refers to the repetitive or statistical dependencies in the data, and can be removed via both lossless and lossy methods.

A variety of methods have been proposed to compress images and videos. But few methods are used to compress color tables. Balaji et al. [88] proposed a preprocessing method for lossless compression of color LUTs. Their method is based on hierarchical differential encoding and cellular interpolative predictive coding methods. Some lossy methods used for videos and images can also be applied to this problem. Sudhakar et al. [89] summarized image compression methods using coding of wavelet coefficients. Shapiro et al. [90] and Said et al. [91] each proposed an efficient wavelet compression method that has been proven very successful in still image coding. Kim et al. [92], and Tang et al. [93] each proposed a three-dimensional wavelet compression method to efficiently encode 3D volumetric image

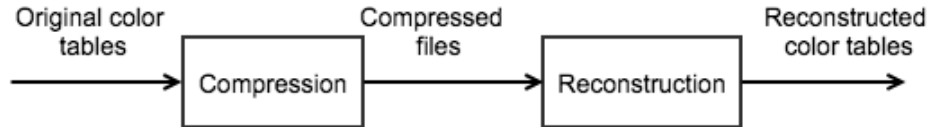
data and video data. Besides wavelet-based methods, the Discrete Cosine Transform (DCT) has also been widely used in image compression [94] and video compression [95].

Our work is based on the work of Chuohao Tang et al. [2] on lossy compression of 3D and 4D color tables through the DCT. Their method used a specific lossy method, followed by a general lossless method to compress the color tables. The specific lossy method was tailored to the specific characteristics of the color tables to be compressed. The general method was generic, and can be used to compress a variety of different types of data streams.

While their work does a good job in compressing 3D and 4D LUTs, they did not try to do a lossless compression. So here we propose a lossless compression method on 1D LUTs. The compressed data consists four files: quantized DCT coefficients from color tables, quantized DCT Coefficients Bit Assignment Tables (CBAT), a residue table whose values are the difference between the original color tables and the initial reconstructed color tables, and the residue table bit assignment tables (RBAT).

## 4.2 Methodology

Fig. 4.1 provides a high-level view of the color table compression-reconstruction process. The compression process consists of two stages: a specific lossless stage that exploits specific characteristics of the color table data to be compressed, followed by a general lossless stage that is designed to compress any kind of data. The reconstruction process mainly consists of a decoding step that inverts the encoding steps [2].



**Figure 4.1.** Compression-reconstruction process.

Fig. 4.2 contains a detailed block diagram of the compression method. Our method can simultaneously compress multiple tables together. We assume there are  $N$  color tables to compress:  $LUT_1, LUT_2, \dots, LUT_N$ . For 1D color tables, there is one channel in the input

color space, and one channel in the output color space. For each output channel, we assume that the LUT contains  $M$  nodes.

At the start of our specific lossless compression stage, we compute the color tables under study minus a reference color table, and get the difference tables. Usually, for 1D color tables, the reference table also contains  $M$  nodes, the same as the original tables to be compressed. But its values are all integers growing monotonically from 0 to  $M - 1$ . After applying the 1-dimensional DCT transform to the difference color tables, we obtain as many DCT coefficients as the number of nodes in the original color tables. In the next step, we quantize the DCT coefficients using a fixed step size  $\Delta$ , and round them to the nearest integer, as shown in Equation 4.1.

$$DCTcoef_q = \left\lfloor \frac{DCT\ Coefficient}{\Delta} \right\rfloor, \quad (4.1)$$

where  $\lfloor \cdot \rfloor$  denotes the rounding operation.

For the rounded 1D quantized DCT coefficients, we multiply them by the same step size  $\Delta$  used above, and then do the inverse DCT. After rounding the inversed DCT results to integers and adding back the reference table values, we get the initial reconstructed color

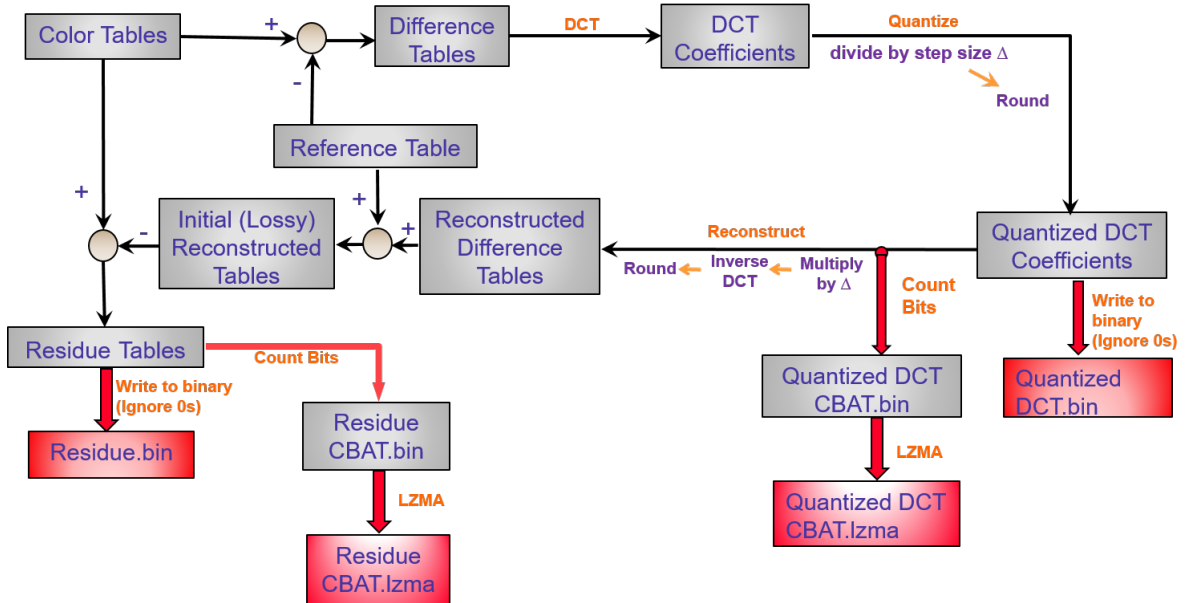


Figure 4.2. Workflow of the compression method.

tables. But this would not guarantee that the initial reconstructed color tables we get are exactly the same as the original LUTs, since we have lost some information in the two rounding processes, and the truncation of the DCT coefficients. So we need to subtract the initial reconstructed color tables' nodes values from original color tables' nodes, and we will get the residue tables. If we add the residue table and initial reconstructed tables, we will perfectly get the original color tables. In this way, our specific compression stage is lossless.

After obtaining the rounded 1D quantized DCT coefficients and residue tables, we can write each of them to a binary file. We do not apply LZMA to these two files, because their values are small and have little redundancy. As a result, the LZMA compression would have a bad compression performance on the two files. Then we use the quantized DCT coefficients to calculate its coefficients bit assignment table (CBAT), and residue tables to get the residues bit assignment table (RBAT). These two files would help identify how many bits each quantized DCT coefficient or residue table value needs, and they will be used for the decoding process. We then write the CBAT and RBAT values to binary files, followed by a general compression method like LZMA. That is because for the CBAT and RBAT tables, the values in them are highly redundant (details on how to get them can be found in next section). So LZMA compression would have good performance.

#### 4.2.1 Coefficients Bit Assignment Table and Residues Bit Assignment Table

The coefficients bit assignment table (CBAT) and the residues bit assignment table (RBAT) are used to store how many bits each rounded quantized DCT coefficient and residue table value uses, respectively. They are calculated in the same way. So here we use CBAT as an example to illustrate how they are calculated.

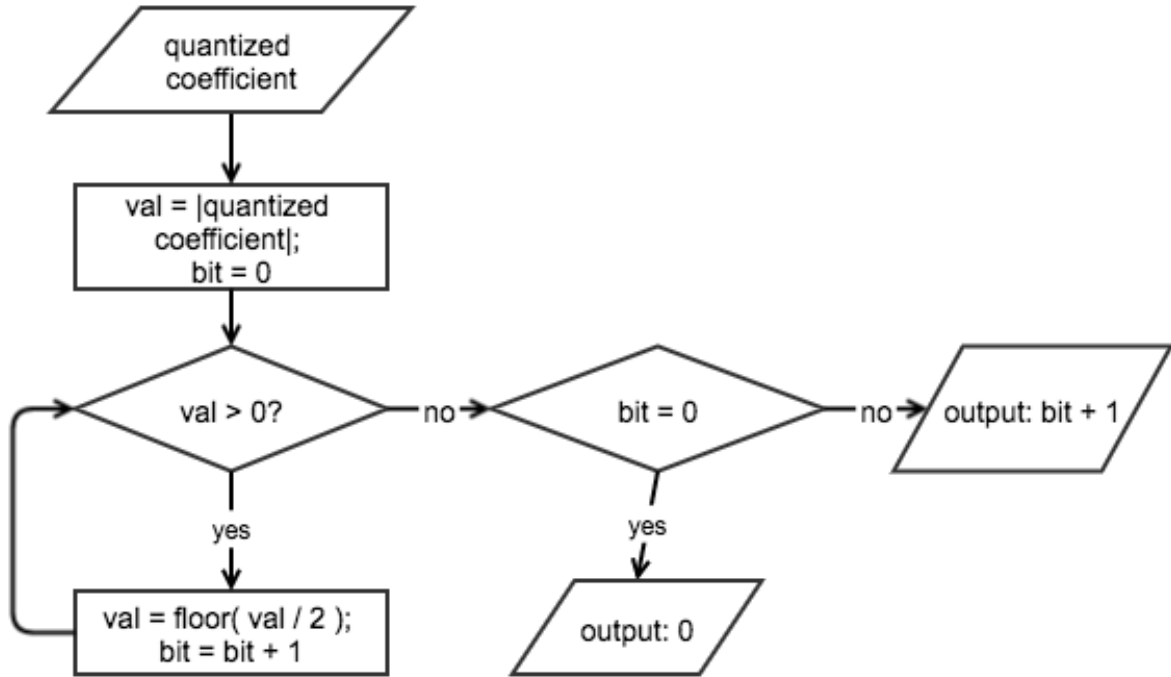
CBAT stores the information of how many bits are assigned to each coefficient. Suppose a rounded quantized DCT coefficient is  $L$ , then we will need  $\lceil \log L \rceil$  bits to store it. As an alternative to implementing the logarithm function directly, Fig. 4.3 [2] shows the method for determining  $\lceil \log L \rceil$  that can be implemented very efficiently in an embedded system. As the coefficient can be a negative number, we need one more bit for the sign.

The nodes in the CBAT are the same as those in the original color tables. Now we will discuss how to calculate each CBAT. We will use the quantized DCT coefficients to calculate the CBAT. For a certain output channel, we denote the rounded quantized DCT coefficient as  $Q_{i,j}$ , where  $i = 1, 2, \dots, N$  is the color table number, and  $j = 1, 2, \dots, M$  is the node number. The number of bits needed for  $Q_{i,j}$  is calculated using Equation 4.2

$$B_{i,j} = \begin{cases} \lceil \log|Q_{i,j}| \rceil + 1 & \text{if } |Q_{i,j}| > 0 \\ 0 & \text{if } |Q_{i,j}| = 0 \end{cases}. \quad (4.2)$$

We use  $L_j$  to represent the CBAT value in node location  $j$ , and it can be calculated by

$$L_j = \max_{1 \leq i \leq N} (B_{i,j}). \quad (4.3)$$



**Figure 4.3.** Algorithm to count bits needed for a given DCT coefficient.

The remaining question about the CBAT is the total size needed to store it. For the CBAT, we assign a fixed number of bits for every node. Assume we assign  $a$  bits for every node in the CBAT, then

$$a = \max_{1 \leq j \leq M} (\lceil \log L_j \rceil). \quad (4.4)$$

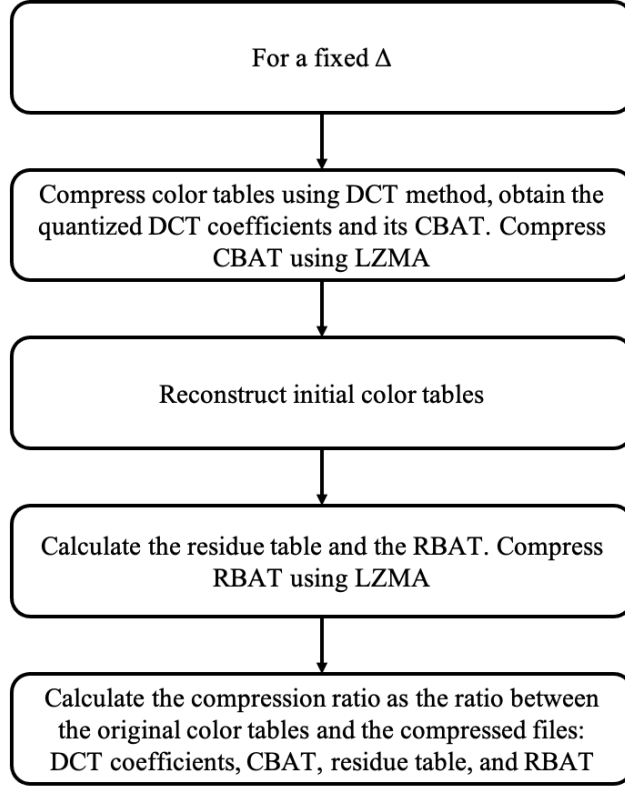
So the total size of one CBAT is  $aM$  bits. Although it is a fixed number, the neighboring nodes tend to have similar values. Thus the total CBAT size can be reduced significantly by the general lossless compression (such as the LZMA) stage. The RBAT is calculated in the same way as the CBAT, and its size can also be reduced greatly by the LZMA algorithm.

Also, the way CBAT and RBAT are calculated can give us a promising way to reduce the size of the rounded quantized DCT coefficients and residue values if we are compressing one LUT a time. Suppose we have a value  $v$  and  $v \neq 0$ . According to Equation 4.2, we will need  $\lceil \log |Q_{i,j}| \rceil + 1$  bits to store it; and we will put the number of bits used in the CBAT or RBAT. Since the highest bit (except for the sign bit) is always “1”, we can just leave it out when storing the rounded quantized DCT coefficient or residue table value in the binary files. Correspondingly, when reconstructing the original color table, we need to put “1” back to the DCT coefficient or residue table value to get the true value back. We can not apply this method to compress multiple tables since their rounded quantized DCT coefficients may vary by a large amount from one coefficient to the other. Thus there is no guarantee that they all have the value “1” after the sign bit.

#### 4.2.2 Determine the Step Size $\Delta$

Another question is how to choose the parameter  $\Delta$  used in DCT coefficient quantization. If  $\Delta$  is large, we can achieve smaller quantized DCT coefficients; but the residue table values will become larger at the same time. If  $\Delta$  is small, the residue table values are small; but the quantized DCT coefficients will be large. The compression ratio is calculated as the result of the size of the original color tables divided by the size of all compressed files. So in order to achieve a high compression ratio, we need to balance the size of quantized DCT coefficients and the residue table.

For a fixed  $\Delta$ , we compress the color tables using the method discussed above. After inverting this compression process, we obtain the reconstructed color tables. Fig. 4.4 shows how to calculate the compression ratio for a given  $\Delta$ .

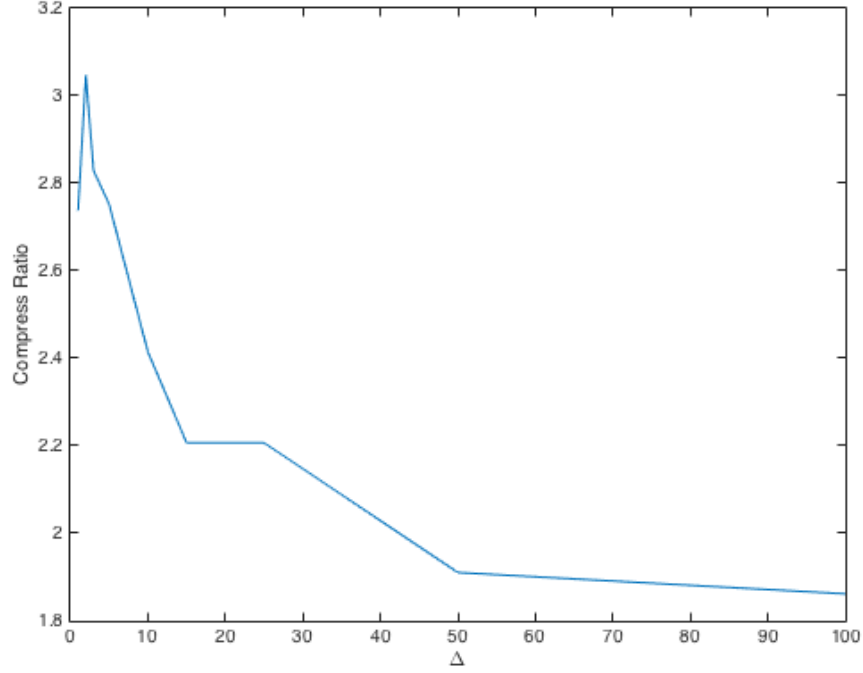


**Figure 4.4.** Process to calculate compression ratio for a given  $\Delta$ .

Fig. 4.5 shows the relationship between compression ratio and step size  $\Delta$  when we store the quantized DCT coefficients and residue table. The color tables on which this data is based are described in the results section.

We can see from the figure that as  $\Delta$  increases, the compression ratio first increases, and then decreases after a peak compression ratio value is reached. That is because when  $\Delta$  increases, the quantized DCT coefficients decrease aggressively, and the residue values of color tables and initial reconstructed tables get larger. Thus, the storage needed for residue tables will increase, thereby reducing the efficiency of our specific lossless stage. Therefore, the peak value is the optimal balance between the compression ratio and the color difference.

Our final output consists of four files: rounded quantized DCT coefficients, residue table, CBAT, and RBAT. When  $\Delta$  increases, the size of rounded quantized DCT coefficients will decrease but the residue table values will increase. So we want to find a balance between the two parts, which is the optimal  $\Delta_{opt}$  that achieves the maximum compression ratio in the compression ratio delta curve.



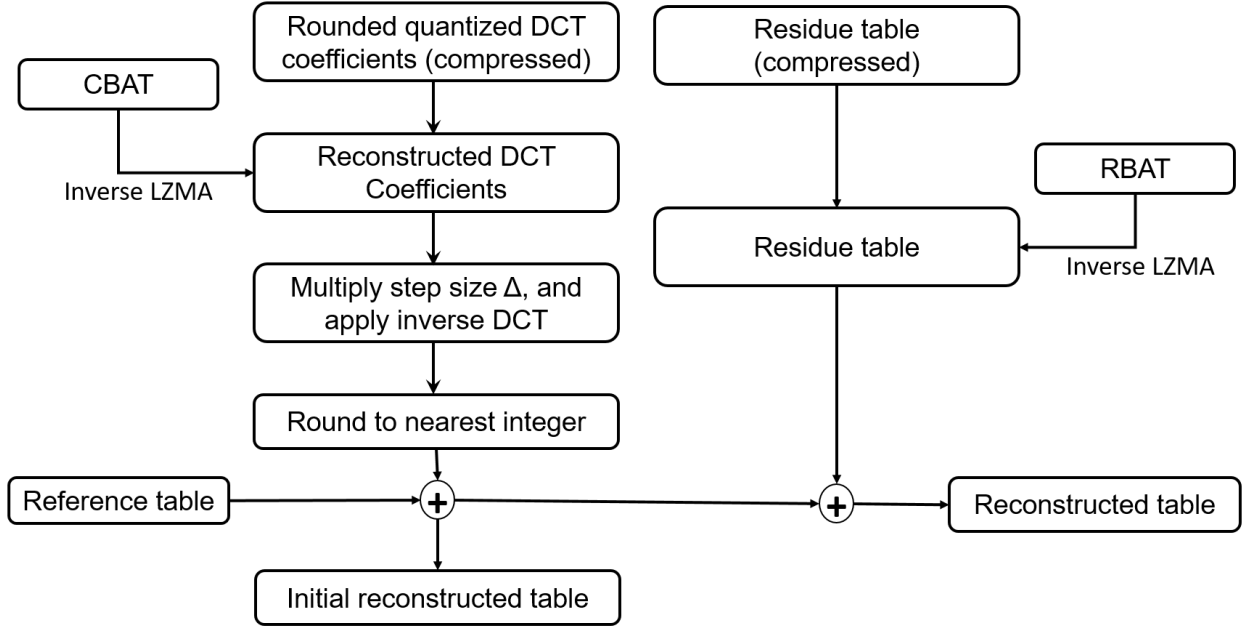
**Figure 4.5.** Compression ratio curve with different step sizes.

### 4.3 Decoding Process

The decoding process extracts the CBAT and RBAT, the rounded quantized DCT coefficients, and the residue table from the compressed data.

Fig. 4.6 shows the workflow of the decoding process. First, we apply the standard lossless decompression technique, such as inverse LZMA, to the CBAT and RBAT files to get the binary stream. Next, we use the CBAT for the quantized DCT to tell how many bits of the binary stream belong to each node location. Then, we can get the reconstructed DCT coefficients. We apply the inverse DCT transform to the coefficients, multiply by the

quantized step size  $\Delta$ , round them to the nearest integer, and add them back to the reference table values to obtain the initial reconstructed color tables. The same method is used for residue table file and the RBAT, except that we do not need to add back the reference table. In this way, we will get the residue table. Adding the initial reconstructed color tables and residue tables, we can get the final reconstructed color tables, which are the same as the original color tables that were to be compressed.



**Figure 4.6.** Decoding process.

#### 4.4 Results

We present experimental results for the compression of two 1D LUTs from HP Inc. The reference table contains 256 nodes, which take on the values 0, 1, 2, ..., 255. As a result, there is no need to store the reference table in final compressed files. Here  $M = 256$ . The byte depth of the table is  $b = 1$ . In total, there are  $bM = 1 \times 256 = 256$  bytes in one table, and 512 bytes in all 2 tables.

From Fig. 4.5 we can see that the maximum compression ratio of 3.05 can be achieved at step size  $\Delta = 2$ . Table 1 shows the size of the four compressed files and their total size

in bytes. Note that the rounded quantized DCT coefficients and residue table are stored directly as binary files, while the CBAT and RBAT are stored as compressed by LZMA.

**Table 4.1.** Storage Needed for Compressed Files

Compressed Files	Size/bytes
Residue Table	35
RBAT	76
Rounded Quantized DCT Coefficients	23
CBAT	34
Total Size	168

In total, we only need 168 bytes to store the compressed files of two color tables above (instead of 512 bytes for original tables), and we can perfectly reconstruct them. This proves that our method achieves good performance.

## 4.5 Conclusion

We have proposed a lossless compression framework for 1D color look-up tables. Our compression method enables the encoding of color tables by storing the information in four parts: rounded quantized DCT coefficients, residue table, CBAT, and RBAT. By decoding these four files we can get the perfect reconstruction of the 1D tables to be compressed.

## 4.6 Major Contributions

My major contributions for this project are as follows:

- Employ a simple reference table, which we do not need to store, to help compress color tables.
- Introduce a method to use the difference between original and initial reconstructed color tables to get the lossless compression.
- Introduce residue tables, and residue bits assignment tables.

- For the task of compressing only one table, we propose a method to use one less bit to store each rounded quantized DCT coefficient and residue value.

## REFERENCES

- [1] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Trans. Syst., Man, Cybern.*, vol. 9, pp. 62–66, Jan. 1979.
- [2] C. Tang, W. Wang, S. Collison, M. Shaw, J. Gondek, A. Reibman, and J. Allebach, “ICC profile color table compression,” in *Proc. 24th Color Imaging Conf.*, San Diego, California, Nov. 2016, pp. 260–265.
- [3] D. G. Mackay, C. Zeller, R. A. Cordery, and H. L. Brunk, “Method for determining a printer’s signature and the number of dots per inch printed in a document to provide proof that the printer printed a particular document,” Mar. 2003, U.S. Patent 6,533,385.
- [4] K. Watson, “Scanning: DPI does count,” [Online]. Available: <http://www.rideau-info.com/photos/scanning.html>, (accessed: 09.10.2018).
- [5] A. C. Bovik, “Automatic prediction of perceptual image and video quality,” *Proc. IEEE*, vol. 101, no. 9, pp. 2008–2024, 2013.
- [6] K. Seshadrinathan and A. C. Bovik, “Automatic prediction of perceptual quality of multimedia signals—a survey,” *Multimedia Tools Appl.*, vol. 51, no. 1, pp. 163–186, 2011.
- [7] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.
- [8] R. Szeliski, “Computer vision: Algorithms and applications,” *Springer Sci. Bus. Media*, 2010.
- [9] D. Wang, F. Speranza, A. Vincent, T. Martin, and P. Blanchfield, “Toward optimal rate control: A study of the impact of spatial resolution, frame rate, and quantization on subjective video quality and bit rate,” in *2003 Visual Commum. Image Process.*, vol. 5150, 2003, pp. 198–209.
- [10] H. R. Sheikh and A. C. Bovik, “Image information and visual quality,” *IEEE Trans. Image Process.*, vol. 15, no. 2, pp. 430–444, 2006.
- [11] E. C. Larson and D. M. Chandler, “Most apparent distortion: Full-reference image quality assessment and the role of strategy,” *J. Electron. Imaging*, vol. 19, no. 1, p. 011 006, 2010.

- [12] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European Conf. Comput. Vision*, Springer, 2016, pp. 694–711.
- [13] A. M. Demirtas, A. R. Reibman, and H. Jafarkhani, "Full-reference quality estimation for images with different spatial resolutions," *IEEE Trans. Image Process.*, vol. 23, no. 5, pp. 2069–2080, 2014.
- [14] H. Yeganeh, "Cross dynamic range and cross resolution objective image quality assessment with applications," *UWSpace*, 2014. [Online]. Available: <http://hdl.handle.net/10012/8667>.
- [15] R. Soundararajan and A. C. Bovik, "RRED indices: Reduced reference entropic differencing for image quality assessment," *IEEE Trans. Image Process.*, vol. 21, no. 2, pp. 517–526, 2011.
- [16] Q. Li and Z. Wang, "Reduced-reference image quality assessment using divisive normalization-based image representation," *IEEE J. Selected Topics Signal Process.*, vol. 3, no. 2, pp. 202–211, 2009.
- [17] Z. Wang, G. Wu, H. R. Sheikh, E. P. Simoncelli, E.-H. Yang, and A. C. Bovik, "Quality-aware images," *IEEE Trans. Image Process.*, vol. 15, no. 6, pp. 1680–1689, 2006.
- [18] F. Pan, X. Lin, S. Rahardja, W. Lin, E. Ong, S. Yao, Z. Lu, and X. Yang, "A locally-adaptive algorithm for measuring blocking artifacts in images and videos," in *2004 IEEE Int. Symp. Circuits Syst. (IEEE Cat. No. 04CH37512)*, IEEE, vol. 3, 2004, pp. III–925.
- [19] A. C. Bovik and S. Liu, "DCT-domain blind measurement of blocking artifacts in DCT-coded images," in *2001 IEEE Int. Conf. Acoustics, Speech, Signal Process. Proceedings (Cat. No. 01CH37221)*, IEEE, vol. 3, 2001, pp. 1725–1728.
- [20] X. Feng and J. P. Allebach, "Measurement of ringing artifacts in JPEG images," in *Digital Publishing*, Int. Society Opt. Photonics, vol. 6076, 2006, 60760A.
- [21] S. Hu, Z. Pizlo, and J. P. Allebach, "JPEG ringing artifact visibility evaluation," in *Image Quality Syst. Performance XI*, Int. Soc. Opt. Photonics, vol. 9016, 2014, 90160E.
- [22] P. Marziliano, F. Dufaux, S. Winkler, and T. Ebrahimi, "A no-reference perceptual blur metric," in *Proc. Int. Conf. Image Process.*, IEEE, vol. 3, 2002, pp. III-57–III-60.
- [23] Z. Wang, H. R. Sheikh, and A. C. Bovik, "No-reference perceptual quality assessment of JPEG compressed images," in *Proc. Int. Conf. Image Process.*, IEEE, vol. 1, 2002, pp. I-477–I-480.

- [24] A. Mittal, A. K. Moorthy, and A. C. Bovik, “Blind/referenceless image spatial quality evaluator,” in *2011 Conf. Rec. 45th Asilomar Conf. Signals Syst. Comput.*, IEEE, 2011, pp. 723–727.
- [25] H. Chen, S. S. Tsai, G. Schroth, D. M. Chen, R. Grzeszczuk, and B. Girod, “Robust text detection in natural images with edge-enhanced maximally stable extremal regions,” in *2011 18th IEEE Int. Conf. Image Process.*, IEEE, 2011, pp. 2609–2612.
- [26] J. Grice and J. P. Allebach, “The print quality toolkit: An integrated print quality assessment tool,” *J. Imaging Sci. Technol.*, vol. 43, no. 2, pp. 187–199, 1999.
- [27] L. Zhang, A. Veis, R. Ulichney, and J. Allebach, “Binary text image file preprocessing to account for printer dot gain,” in *IEEE Int. Conf. Image Process. (ICIP)*, IEEE, 2014, pp. 2639–2643.
- [28] M. Cannon, J. Hochberg, and P. Kelly, “Quality assessment and restoration of type-written document images,” *Int. J. Document Anal. Recognition*, vol. 2, no. 2-3, pp. 80–89, 1999.
- [29] H. Li and D. S. Doermann, “Text quality estimation in video,” in *Document Recognition Retrieval IX*, Int. Soc. Opt. Photonics, vol. 4670, 2001, pp. 232–243.
- [30] N. Nayef and J.-M. Ogier, “Metric-based no-reference quality assessment of heterogeneous document images,” in *Document Recognition Retrieval XXII*, Int. Soc. Opt. Photonics, vol. 9402, 2015, p. 94020L.
- [31] B. Zhang, J. P. Allebach, and Z. Pizlo, “An investigation of perceived sharpness and sharpness metrics,” in *Image Quality Syst. Performance II*, Int. Soc. Opt. Photonics, vol. 5668, 2005, pp. 98–110.
- [32] H. Li, F. Zhu, and J. Qiu, “DeepITQA: Deep based image text quality assessment,” in *Int. Conf. Neural Inform. Process.*, Springer, 2018, pp. 397–407.
- [33] J. Tyson, “How scanners work,” [Online]. Available: <https://computer.howstuffworks.com/scanner.htm>, (accessed: 01.19.2019).
- [34] S. Fiel, M. Diem, and F. Kleber, “Nomacs image lounge,” version 3.12, May 27, 2019. [Online]. Available: <https://nomacs.org>.
- [35] I. Skiljan, “Irfanview,” version 4.40, [Online]. Available: <https://www.irfanview.com/>.
- [36] C. Lanczos, “An iteration method for the solution of the eigenvalue problem of linear differential and integral operators,” 1950.

- [37] A. V. Oppenheim, J. R. Buck, and R. W. Schafer, *Discrete-Time Signal Processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.
- [38] P. G. Freitas, W. Y. Akamine, and M. C. Farias, “Using multiple spatio-temporal features to estimate video quality,” *Signal Process. Image Commun.*, vol. 64, pp. 1–10, 2018.
- [39] I. Sobel, “An isotropic  $3 \times 3$  image gradient operator, presentation at Stanford artificial intelligence project (sail),” 1968.
- [40] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 6, pp. 679–698, 1986.
- [41] A. Michelson, “Studies in optics,” *U. Chicago Press, IL*, 1927.
- [42] N. Travnikova, “Efficiency of visual search,” *Mashinostroyeniye, Moscow, USSR*, 1984.
- [43] G. T. Fechner, “Elemente der psychophysik,” *Breitkopf u. Härtel*, vol. 2, 1860.
- [44] E. Peli, “Contrast in complex images,” *J. Optical Soc. Am. A*, vol. 7, no. 10, pp. 2032–2040, 1990.
- [45] S. A. Drew, C. F. Chubb, and G. Sperling, “Precise attention filters for Weber contrast derived from centroid estimations,” *J. Vision*, vol. 10, no. 10, pp. 20–20, 2010.
- [46] G. D. Boreman, “Modulation transfer function in optical and electro-optical systems,” *SPIE press Bellingham, WA*, vol. 4, 2001.
- [47] H.-T. Lin and C.-J. Lin, “A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods,” *submitted to Neural Computation*, vol. 3, pp. 1–32, 2003.
- [48] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, *et al.*, “Comparison of classifier methods: A case study in handwritten digit recognition,” in *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*, IEEE, vol. 2, 1994, pp. 77–82.
- [49] S. Knerr, L. Personnaz, and G. Dreyfus, “Single-layer learning revisited: A stepwise procedure for building and training a neural network,” in *Neurocomputing*, Springer, 1990, pp. 41–50.
- [50] J. C. Platt, N. Cristianini, J. Shawe-Taylor, *et al.*, “Large margin DAGs for multiclass classification,” in *NIPS*, vol. 12, 1999, pp. 547–553.

- [51] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE Trans. Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.
- [52] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Trans. Intelligent Syst. Technol.*, vol. 2, 27:1–27:27, 2011, Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [53] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” *Mach. Learning*, vol. 46, no. 1, pp. 389–422, 2002.
- [54] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2016, pp. 770–778.
- [55] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *arXiv preprint arXiv:1506.01497*, 2015.
- [56] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2016, pp. 779–788.
- [57] L. Wang, Y. Guo, Z. Lin, X. Deng, and W. An, “Learning for video super-resolution through HR optical flow estimation,” in *Asian Conf. Comput. Vision*, Springer, 2018, pp. 514–529.
- [58] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Int. Conf. Medical Image Comput. Computer-assisted Intervention*, Springer, 2015, pp. 234–241.
- [59] H. Talebi and P. Milanfar, “NIMA: Neural image assessment,” *IEEE Trans. Image Process.*, vol. 27, no. 8, pp. 3998–4011, 2018.
- [60] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [61] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2016, pp. 2818–2826.
- [62] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

- [63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2018, pp. 4510–4520.
- [64] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, “Searching for mobilenetv3,” in *Proc. IEEE/CVF Int. Conf. Comput. Vision*, 2019, pp. 1314–1324.
- [65] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: More features from cheap operations,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2020, pp. 1580–1589.
- [66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *IEEE Conf. Comput. Vision Pattern Recognition*, IEEE, 2009, pp. 248–255.
- [67] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2018, pp. 7132–7141.
- [68] P. Contributor, *QUANTIZATION*. [Online]. Available: <https://pytorch.org/docs/stable/quantization.html>, (accessed: 05.15.2020).
- [69] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2018, pp. 2704–2713.
- [70] J. Guo, X. Liu, Y. Chen, and X. Wang, “A revised feature extraction method for detecting text page up/down orientation,” in *Proc. 2011 Int. Conf. Appl. Superconductivity Electromagnetic Devices (ASEMD 2011)*, Sydney, NSW, Australia, Dec. 2011, pp. 105–108.
- [71] J. van Beusekom, F. Shafait, and T. M. Breuel, “Combined orientation and skew detection using geometric text-line modeling,” *Int. J. Document Anal. Recognition (IJDAR)*, vol. 13, pp. 79–92, Jun. 2010.
- [72] V. Roy, K. G. Srinidhi, Y. Wu, and P. Bauer, “System and method for document orientation detection,” Apr. 2014, U.S. Patent 8712188.
- [73] Z. Fan, M. R. Campanelli, and D. Venable, “Page orientation detection based on selective character recognition,” Jun. 2012, US Patent 8200043.

- [74] S. Ghosh and B. B. Chaudhuri, "Composite script identification and orientation detection for indian text images," in *Proc. 2011 11th Int. Conf. Document Anal. Recognition (ICDAR 2011)*, Beijing, China, Sep. 2011, pp. 294–298.
- [75] S. F. Rashid, F. Shafait, and T. M. Breuel, "Discriminative learning for script recognition," in *Proc. 2010 IEEE Int. Conf. Image Process.*, Hong Kong, China, Sep. 2010, pp. 2145–2148.
- [76] S. Lu and C. L. Tan, "Script and language identification in noisy and degraded document images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, pp. 14–24, Jan. 2008.
- [77] C. Jain, C. V. Goudar, K. G. Srinidhi, and Y. Wu, "System and method for script and orientation detection of images using artificial neural networks," Nov. 2014, U.S. Patent 8891822.
- [78] L. G. Shapiro, "Connected component labeling and adjacency graph construction," *Mach. Intell. Pattern Recognition*, vol. 19, pp. 1–30, 1996.
- [79] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Connected components labeling," 2003 (accessed October 15, 2018). [Online]. Available: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm>.
- [80] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, pp. 583–598, Jun. 1991.
- [81] A. AbuBaker, R. Qahwaji, S. Ipson, and M. Saleh, "One scan connected component labeling technique," in *Proc, 2007 IEEE Int. Conf. Signal Process. Commun. (ICSPC 2007)*, Dubai, United Arab Emirates, Nov. 2007, pp. 1283–1286.
- [82] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture process.," *J. of ACM (JACM)*, vol. 13, pp. 471–494, Oct. 1966.
- [83] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, pp. 749–756, Aug. 2007.
- [84] P. W. Wong, "Inverse halftoning and kernel estimation for error diffusion," *IEEE Trans. Image Process.*, vol. 4, pp. 486–498, Apr. 1995.
- [85] M.-C. Jung, Y.-C. Shin, and S. N. Srihari, "Mach. printed character segmentation method using side profiles," in *Proc. 1999 IEEE Int. Conf. Syst., Man, Cybern.*, Tokyo, Japan, Oct. 1999, pp. 863–867.

- [86] B. Henderson, “Getting netpbm,” *Sourceforge*, Retrieved 2 February 2021.
- [87] Z. Hu, C. Tang, T. M. Nelson, M. Q. Shaw, J. P. Allebach, and A. R. Reibman, *Color table compression*, U.S. Patent App. 15/642929, Jan. 2018.
- [88] A. Balaji, G. Sharma, M. Shaw, and R. Guay, “Preprocessing methods for improved lossless compression of color look-up tables,” *J. of Imaging Sci. Technol.*, vol. 52, pp. 040901-1–9, Jul. 2008.
- [89] R. Sudhakar, R. Karthiga, and S. Jayaraman, “Image compression using coding of wavelet coefficients—a survey,” *ICGST-GVIP J.*, vol. 5, pp. 25–38, Jun. 2005.
- [90] J. M. Shapiro, “Embedded image coding using zero trees of wavelet coefficients,” *IEEE Trans. Signal Process.*, vol. 41, pp. 3445–3462, Dec. 1993.
- [91] A. Said and W. A. Pearlman, “A new, fast, and efficient image codec based on set partitioning in hierarchical trees,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, Jun. 1996.
- [92] B.-J. Kim, Z. Xiong, and W. A. Pearlman, “Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT),” *IEEE Trans. Circuits and Syst. Video Technol.*, vol. 10, pp. 1374–1387, Dec. 2000.
- [93] X. Tang and W. A. Pearlman, “Three-dimensional wavelet-based compression of hyperspectral images,” in *Hyperspectral Data Compression*, Springer, 2006, pp. 273–308.
- [94] A. B. Watson, “Image compression using the discrete cosine transform,” *Math. J.*, vol. 4, p. 81, Jan. 1994.
- [95] B.-L. Yeo and B. Liu, “Volume rendering of DCT-based compressed 3D scalar data,” *IEEE Trans. Visualization Comput. Graph.*, vol. 1, pp. 29–43, Mar. 1995.

## VITA

Zhenhua Hu got his B.S. degree in Biomedical Engineering from Shandong University, China in the year 2011. He got his M.S. degree in Biomedical Engineering from Zhejiang University, 2014. He started graduate school in Purdue as a master student in the August 2015, and joined Prof. Allebach's group in May 2016. He transferred to Ph.D. program in February 2017, and passed the qualifying exam in August 2017. He passed his Preliminary Exam in December 2018. His current research focuses on image processing, image quality, and machine learning.