DATA COMPRESSION AND IMAGE PROCESSING

Li Yicheng*

email: l.y.c.liyicheng@gmail.com

USC Viterbi of Engineering

*github link: https://github.com/IAMLYCHEE

I. Entropy Coding

I. Shannon-Fano Coding

I.1 Basic description

Shannon-Fano coding is a technique for constructing a prefix code based on a set of symbols and their probabilities. Prefix code is a useful coding method that the code of each symbol would not be the prefix of other's code. In Shannon-Fano coding, the symbols are arranged in order from most probable to least probable, and devided into two sets whose total probability are as close as possible to being equal. Repeat doning this for each of the two sets until there is only one symbol in each set.

I.2 Algorithm

Algorithm description

•1 Develop a map mapping from each symbol of the data file to their relative frequency of occurrence.
•2 Sort the lists of symbols according to the frequency, with the most frequently occuring symbols at the left and the least common at the right.
•3 Divide the list into two parts, first give the total probability of the list of symbols, then starting from the first symbol, accumulate the probabilities until the it reaches half of the total probability. Record the place and divide the list into two parts.
•4 The left part of the list whose symbols has greater probability is assigned the binary digit 0, and the right part is assigned the digit 1.
•5 Recursively apply the steps 3 and 4 to each of the two lists until only one symbol exists in the list.

Flow Chart

*tip
*some notations in flow chart are in C++ mode.
*vector stores pairs in a vector way
*a pair is a two element tuple, in this case we have a tuple of a symbol and its relative frequency

flow chart:

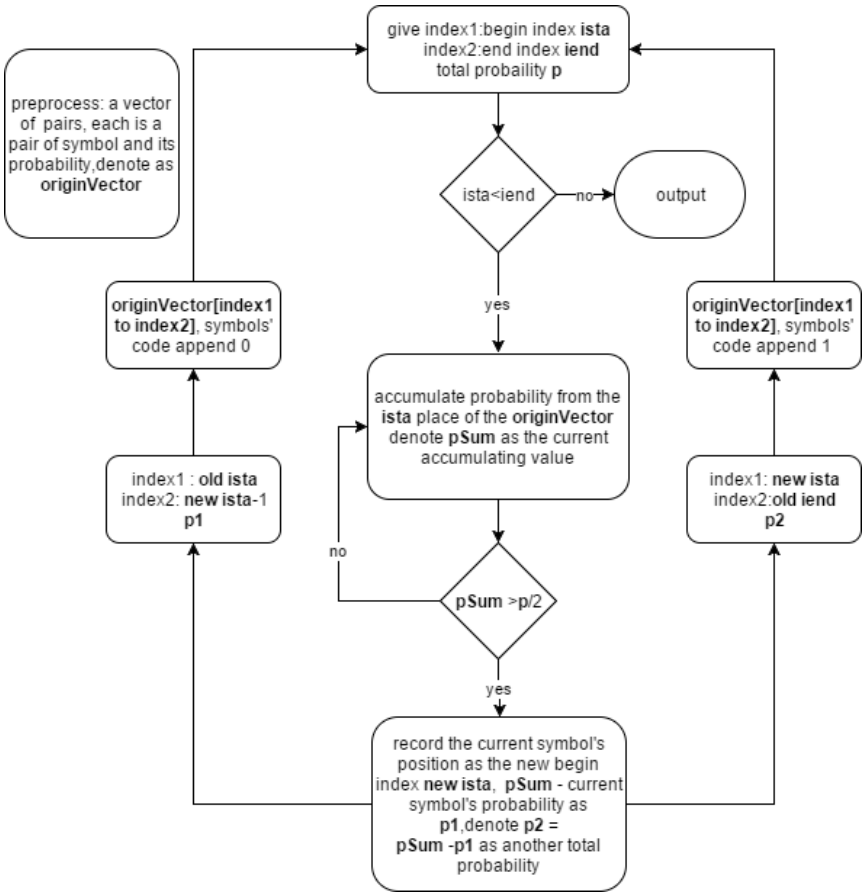


Figure 1: *Shannon-Fano coding Flow Chart*

I.3 Statistics and result

Requirement:

1: Compute the entropy, ralative frequency of symbols and any neccessary statistical information from the source for each input file.

2: Develop a Shannon-Fano encoder based on the global statistics

3: Apply the encoder to each input file . Calculate the compressed file size and compressionration.

Experiment environment:
Windows SDK version:10.0.15063.0
Compiler : default Microsoft Visual C++ compiler

Procedure: First ,a class named Statistics is created to compute and record the statistics of a given file. Then,a class named ShannonFano is created to perform the ShannonFano coding. Finally in the main file , we generate the probability table using Statistic object and perform Shannon-Fano coding on that object.

Table 1: *Description of class 'Statistic'*

class name: Statistics (file statistics.h)		
User	method name	description
(+)	Statistics(char* ,long)	Constructor call genProbTable() and initialize probability table
(+)	double genEntropy()	calculate the entropy for users to retrieve
(+)	void description()	print the entropy, the probability table
(-)	std::map<char, long> genProbTable(char * data,long length)	calculate the relative frequency for each symbol

Table 2: *Description of class 'ShannonFano'*

class name: ShannonFano (file: ShannonFano.h)		
User	member name	description
(-)	std::map<char,std::string>codebook	stores the symbol and its code
(-)	std::vector<std::pair<char,double» sortedProbTable	stores the sorted probability table in descending order
User	method name	description
(+)	void encodeShannon(string,string)	give the input file location,output file location write the coded information into output file. call EncShannon() inside
(-)	void EncShannon(int,int,double)	perform the ShannonFano algorithm

Table 3: *ShannonFano coding result(codebook not included)*

Statistics and Compression Performance							
Filename	Symbol count	Entropy	Size		Bit/Sym (after coded)	Compression Ratio	Saving Percent
			Original	Compress			
text.dat	70	4.47069	8762	5061	4.62086	0.57760	42.23%
audio.dat	249	6.45639	65540	53855	6.57370	0.82171	17.83%
image.dat	230	7.59311	65536	64120	7.82714	0.97839	2.16%
binary.dat	2	0.18324	65536	8192	1	0.125	87.5%

I.4 Performance Analysis

Performance of Shannon-Fano coding would be discussed in the following section when we finish building the Huffman coding method. Two coding methos would be compared and discussed.

II. Huffman Coding with global Statistics

II.1 Basic description

The codes generated by Huffman coding methods are also prefix codes and are optimum for a given model. The Huffman procedure is based on two observations regarding optimuprefixed codes.

1:In an optimum code, symbols that occur more frequently will have shorter codewords than symbols that occur less frequently.

2:In an optimum code, the two symbols that occur least frequently will have the sam length.

The Huffman procedure is obtained by adding a simple requirement to these two observations.This requirement is that the codewords corresponding to the two lowest probability symbols differ only in the last bit. That is, if γ and δ are the two least probable symbols in an alphabet, if the codeword for γ was $\{0,1\}^m + 0$ (here $+$ denotes concatenation), than the code word for δ would be $\{0,1\}^m + 1$.

These above observations gives a very simple coding procedure.

II.2 Algorithm

Algorithm description

•1 Develop a map mapping from each symbol of the data file to their relative frequency of occurrence.(same of the Shannon-Fano coding)

•2 This time we build a heap for the nodes. First push leaf nodes, which are all the symbols with their probability and NULL parent.Also build a heap that always put the nodes with least probability at the top, and they leave first(so this is a queue data structure)

•3 Build the binary tree starting at the leaf nodes. Find the two symbols with smallest probabilities. Give them a parent node. Assign the parent node the sum of the two probabilities,connect the parents to node to its two offspring. Finally, remove the two leaf from the heap we created.

•4 Repeat step 3, until the queue has size 0, or the weight of the node reach 1.

•5 Code assign, my method is to use preorder travel the tree(also a recursive algorithm), travel the tree until meet some leaf, assign the path to the symbol.

Flow chart

1.an exmaple of 5 symbols

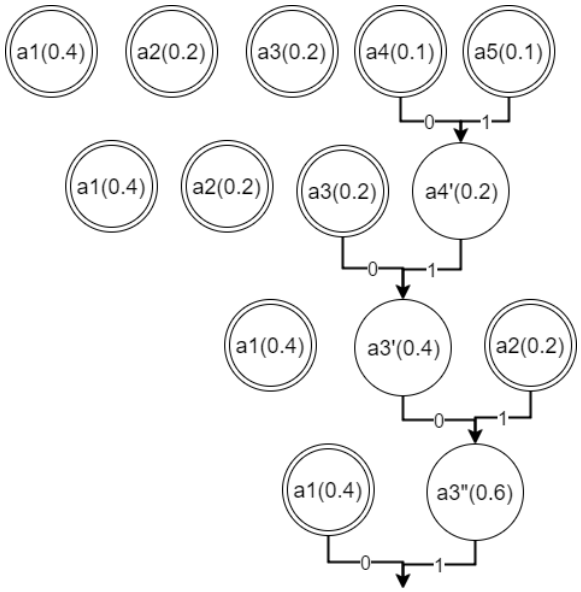


Figure 2: *An example*

2.flowchart

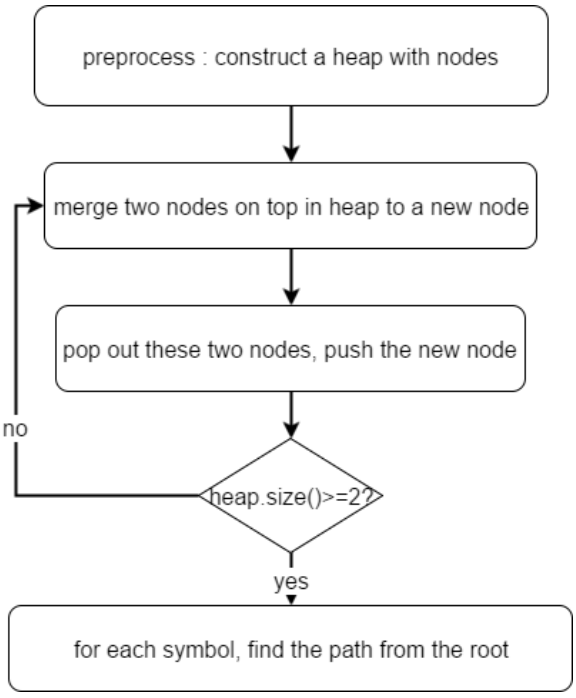


Figure 3: *flowchart*

Table 4: *Node Structure*

structure: Huffmantree	
member:	operator
symbol c frequency cfreq left Huffmantree* right Huffmantree*	Constructor(char,double, Huffmantree*,Huffmantree*) bool operator() //Compare

Table 5: *Methods designed in main.cpp*

methods in main.cpp		
build_tree	input	<vector<pair<char,double»
	output	HuffmanTree*
	usage	reutrn the root of the tree
preorder _Travel	input	HuffmanTree*
	output	NULL
	usage	Travel the tree depth first
hasPath	input	HuffmanTree*,probTable,char
	output	bool
	usage	find path for a symbol

II.3 Experiment and Results:

Requirement:

1:Develop the Huffman encoder based on the global statistics.

2:Apply the encoder to each input file. Report the compressed file sizes and the compression ratio.

Procedure: Using the class Statistics built before, gather the probability table. Then build the Huffman tree structure. In Huffman coding programming, I did not do the encapsulation to seperate it into an independent class, but I would describe it in the same way.

Result:

Table 6: *Huffman Coding Performance*

Statistics and Compression Performance				
Filename	Size		Ratio	Saving Percent
	Original	Compress		
text.dat	8762	4939	0.56368	43.63%
audio.dat	65540	53170	0.81126	18.87%
image.dat	65536	62433	0.9526	4.73%
binary.dat	65536	8192	0.125	87.5%

II.4 Performance Analysis

To show the comparison we first build the table below:

Table 7: *Comparison between Huffman Code and Shannon-Fano code*

Statistics and Compression Performance						
Filename	Symbols count	Entropy	Bit/Sym		Compression Ratio	
			Shannon-Fano	Huffman	Shannon-Fano	Huffman
text.dat	70	4.47069	4.62086	4.5094	0.57760	0.56368
audio.dat	249	6.45639	6.57370	6.4901	0.82171	0.81126
image.dat	230	7.59311	7.82714	7.6212	0.97839	0.9526
binary.dat	2	0.18324	1	1	0.125	0.125

To begin with, we compare the compression ratio between files, that is the binary file has a best compression ratio, while the image has the worst. The reason is the entropy, that is, the distribution of symbols in each source file matters. Those data are imported into matlab, and the following is the probility distribution figure I designed to show how that matters to entropy and compression ratio.(the binary case is discard since it would be discussed in another way)

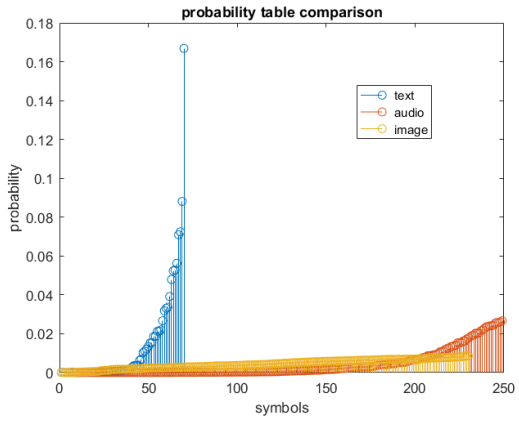


Figure 4: *probability distribution*

From the figure, it is easy to see that a source file with less symbol could achieve more compression ration. This is rational, since the code length would be reduced if we have less symbol. However, the image file has less symbol than the audio file, it performs worth in compression. As we can see in the figure, the variance of probability of symbol in the image data are so small that it almost looks like a uniform distribution. The audio file, however, has more symbols whose probabilities are so small , and some symbol with greater probability. That is the reason it performs more compression.

Next, let us compare the performance of two coding method. Clearly, whatever the file is, the huffman coding always performs better and its result is really close to the entropy we computed. As we can see from **table 7**, the differences between the entropy and the Huffman bit/Symbol have an average of 0.03. Huffman performs better than Shannon-Fano because it is a bottom-up method to build a tree so that the symbol with less probability would be assigned with very long code. Shannon-Fano

works in blocks, so some symbol with higher probability may be assigned with longer bits. If we use Huffman coding or Shannan-Fano file with only two symbols, that is not reasonable enough, because just assign 1 to one of the symbol and 0 to another symbol. After that, we again perform the Huffman or Shannon-Fano coding, that is, we perform the coding twice, so when we decode, we also need to decode twice. However,since we only have two symbol, we have various other ways to deal with it. The next section, run-lenght coding is discussed and we can also perform QM coding.

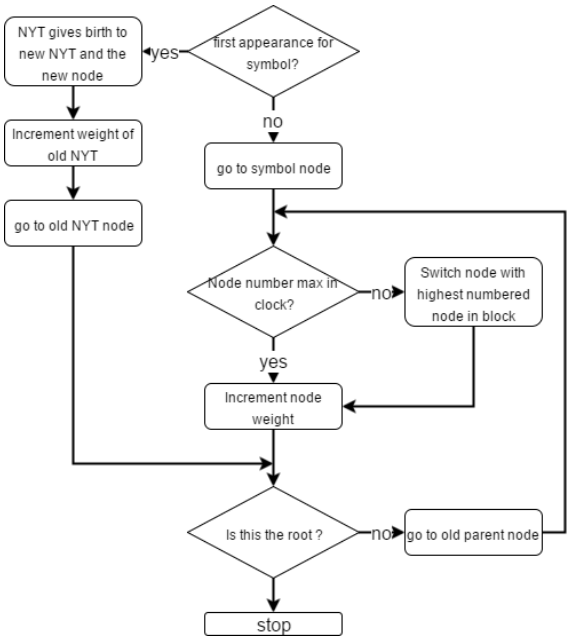


Figure 5: *Update procedure for the adaptive Huffman Coding*

III. Adaptive Huffaman Coding

The previous Huffman Coding requires global statistics which is not realistic when we deal with real-time problems. We need to convert the previous algorithm into a one-pass procedure, that every time a symbol is passed, the source is encoded in the second pass[1]. In adaptive Huffman coding procedure, neither transmitter nor receiver knows anything about the statistics of the source sequence at the start of transmission. The tree at both the transmitter and the receiver consists of a single node that corresponding to all symbols not yet transmitted(NYT) and has a weight of zero[1].The tree is updated with an update procedure,see blow figure[1]:

The core idea of the update procedure, is that the node that has more weight should have the higher order. Therefore, everytime a symbol goes into this tree, some branches are swaped, some nodes are added, and some weight and order are changed.Actually, according to the flow chart, if we have a new symbol, we increase the weight of the old NYT node and check whether the parent of it has the largest order number, or if we have an old symbol, goto that symbol node and check whether it has the largest order number and its parent node. Generally, we just continously goto current nods's parent node and check whether that node has the largest number, because the current node's parent always should increase weight.

The follwing two figures gives the example, and actually it is easy to understand how to operate the updating schedule.(It is an exam-

ple modified according to [1] FIGURE3.7)

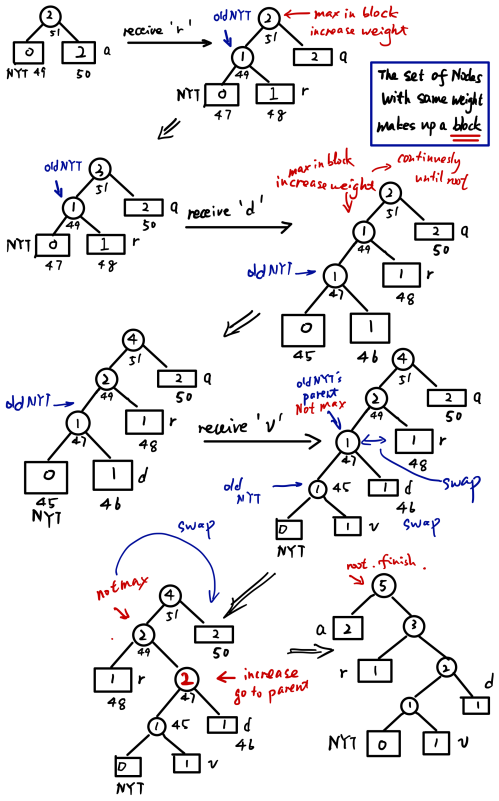


Figure 6: *An example explains updating procedure*

III.1 Result

*important: After trying hard on programming adaptive huffman coding for 3days, still some bugs exist(not syntax error), so I used to codes from github written by Djuned Fernando Djusdek,*github link: https://github.com/santensuru/adaptive-huffman* to obtain the results. Experiment environment:

Windows SDK version:10.0.15063.0

Compiler : default Microsoft Visual C++ compiler

Table 8: *Adaptive Huffman result*

Adaptive Huffman Coding				
file	origin size	Compress size	Compress ratio	Percent Saved
text	8792	5023	0.5713	42.87%
audio	65540	53607	0.8179	18.21%
image	65536	62844	0.9589	4.11%
binary	65536	8423	0.1285	87.14%

Analysis The final compress ratio is almost the same with the coding with global statistic, however in this case, the updating procedure used by both transmitter and receiver is identical so that encoding and decoding processes remain synchronized. We can realized a good realtime compression coding.

II. Run-Length Coding

I. Three schemes

I.1 Basic Scheme

The basic version of run-length coding is simple, just find out those continous repeated symbols and code them into the repeat times plus the repeat symbol. A data structure stack is suitable in this algorithm. We create a stack to store the current symbol, if the new symbol read is the same with the symbol in the stack, we push the symbol into the stack. When the new symbol read is not the same, pop elements in the stack until the stack is empty, and push the new symbol in the stack as the current symbol. In this experiment, we use 2 bytes for recording the repeated size, so the repeat amount value range from 1 to 256. So there exist some threshold hold operation, that once

a symbol repeats more than 255 times, we first output *0xff* and the symbol first continuously until the size reduce to less than 255.

flow chart:

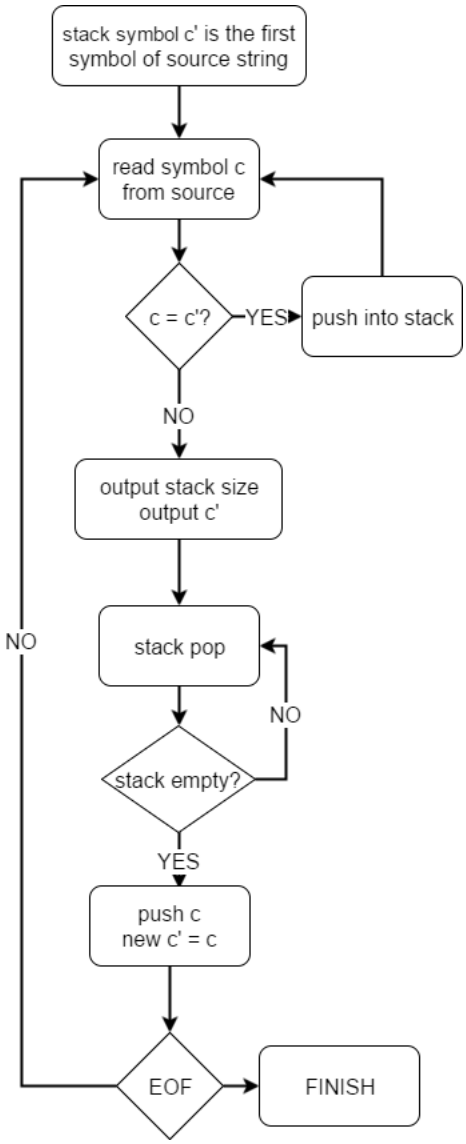


Figure 7: *flow chart of basic scheme*

I.2 Modified Scheme

This scheme modifies the basic scheme, because it is obvious that if a symbol only occurs one time, it would be coded into longer length. The modified scheme modified this by adding a flag. When the coder meet repeated symbols, it adds '0x80' to the repeat size so that when the decoder meet a symbol whose MSB = 1, it knows this is a value of size and the symbol after that is the repeated symbol. The following flow chart shows the algorithm.

flow chart:

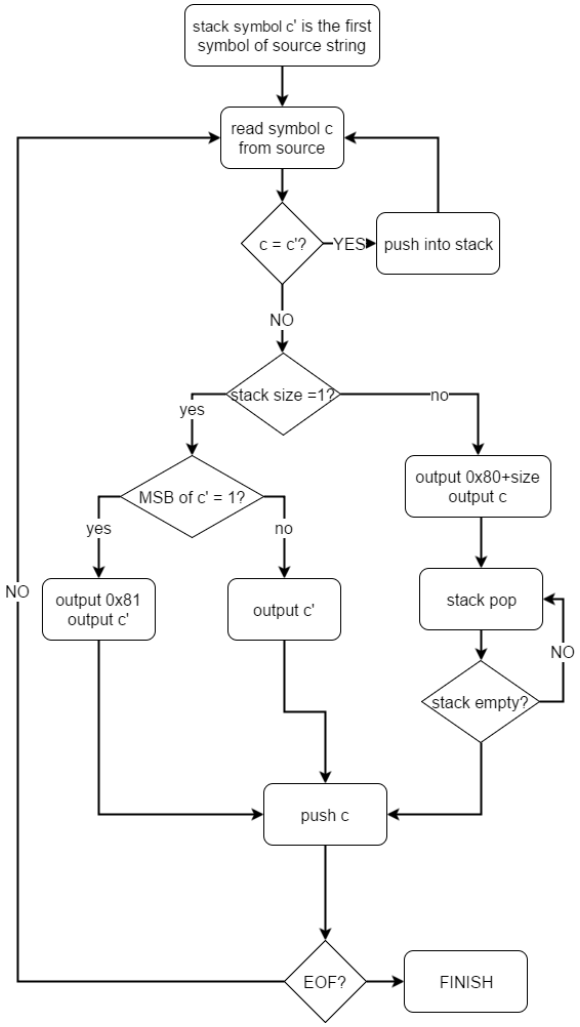


Figure 8: *flow chart of modified scheme*

Moreover, in this experiment, we use 2

bytes for recording the repeated size, since the MSB is already sure to be 1, we have 7 other bits to record the value, so the repeat value range from 1 to 127.

I.3 Burrows-Wheeler Transformation

Background and an Example

In 1983, Michael Burrows and David Wheeler proposed the Burrows-Wheeler Tranformation(BWT). This algorithm divides a source file into parts by deciding the block size. For example, if the source file has a length of x, it is divided in the several block strings each block with length N,. For each block, we circular shift the string to form N-1 another strings and merge them into a matrix. Then the matrix is sorted according to the first symbol, the last column and the index of the origin string's position in the sorted matrix is transmited.

**Example, string: compress*

•1 **circular shift:**

C	O	M	P	R	E	S	S
O	M	P	R	E	S	S	C
M	P	R	E	S	S	C	O
P	R	E	S	S	C	O	M
R	E	S	S	C	O	M	P
E	S	S	C	O	M	P	R
S	S	C	O	M	P	R	E
S	C	O	M	P	R	E	S

•2**sort:**

C	O	M	P	R	E	S	S
E	S	S	C	O	M	P	R
M	P	R	E	S	S	C	O
O	M	P	R	E	S	S	C
P	R	E	S	S	C	O	M
R	E	S	S	C	O	M	P
S	C	O	M	P	R	E	S
S	S	C	O	M	P	R	E

•3**output**

In this case the output would be: *0 SROCMPSE* , because the origin string is at position 0 in the sorted matrix and the last column is *SROCMPSE*.

**Code design*

we can use the standard library for sort the string by using the simple definition of string comparison.

Encoding Process

*1.For construct the matrix, we have string s_1, we take its substring from begining to the second last, that is $s_1[0 \cdots N-2]$, then append the $s_1[N-1]$ to the begining of this substring to form s_2. Repeat doing so for the next strings.

*2. The string matrix is sorted with 'string', check the C++ operation for the string comparision, basicly it refer to the position of a symbol in the alphbet.

*3. Output the string formed by the last column, and the actual position of the origin string in the matrix.

Decoding process

*1 Sort the string and append it .

*2 Sort the matrix.

*3 append the coding string to the front , goto 2 until square matrix formed

*4 output the string of position given in the matrix.

I.4 Realization and Result

A class named RunLength is designed to do all the computation:

Table 9: *Description of class "RunLength"*

class name : RunLength (file:runLength.h)		
User	method name	description
(+)	string encodeBasic(string,string)	encode using the basic scheme, input: source file path coded file path output: coded string
(+)	string encodeMRLC(string,string)	encode using the modified scheme input:source file path coded file path output:coded string
(+)	string encodeBWT(unsigned short, string,string)	encode using BWT algorithm input: block size for BWT source file path coded file path output:coded string
(+)	void decodeBasic(string,string)	decode using the basic scheme, input: source file path decoded file path
(+)	void decodeModified(string,string)	decode using the modified scheme input: source file path decoded file path
(-)	pair<unsigned short,string>BWT(unsigned short,string)	perform the BWT for a string block input: block size source file string output: index and sorted string
(-)	string getBasicScheme(char*,long)	coding with basic scheme input: source data source length output:coded string
(-)	string getModifiedScheme(char*,long)	coding with modified scheme input: source data source length output:coded string

After create objects in the main file, and we call those method, we have the following result:

Table 10: *Compression Result*

Run-Length coding (For BWT we first do BWT, then MRLC)											
filename	origin size	RLC		MRLC		BWT(N=20)		BWT(N=30		BWT(N=50)	
		size	ratio	size	ratio	size	ratio	size	ratio	size	ratio
text.dat	8792	17128	1.9481	8759	0.9962	9590	1.0908	9243	1.0530	9114	1.0413
audio.dat	65540	108542	1.6561	62625	0.9555	70377	1.0738	68190	1.0404	66488	1.0145
image.dat	65536	124318	1.8969	65272	0.9960	71905	1.0972	69731	1.0601	67987	1.0374
binary.dat	65536	4780	0.0729	4346	0.6631	15619	0.2383	11517	0.1757	8219	0.1254

References

[1] Khalid Sayood *Introduction to Data Compression,Third Edition*. ISBN:978-0-12-620862-7

III. QM Coder

I. Motivation & Introduction

QM coder is an arithmetic coder which is used in JPEG,JPEG 2000 and JBIG. It handles only binary strings or input and it is designed for simplicity and speed. To increase efficiency, it uses approximation for multiplication operation, fixed-precision integer arithmetic with renormalization of the probability interval from time to time.Since QM coding is an arithmetic coding, it removes redundancy in the data and is suitable to be the final step for entropy encoding.The main idea of the QM-coder is to classify the input bit as **M**ore **P**robable **S**ymbol and **L**ess **P**robable **S**ymbol. Before the next bit is input, the QM-coder uses a statistical model to predict which one of the bits will be the MPS.

II. Principle & Approach

Just like basic ideas of arithmetic coding, the QM coder also cares about interval, the start of the interval, and the probability.

II.1 Basic version:

Notation:
L: the lower bound of the interval
U: the upper bound of the interval
P_l: Lower Bound Probability of Symbol
P_u: Upper Bound Probability of Symbol

The basic routine of arithmetic coding:
$L_{new} := L + (H - L) \times P_l$
$H_{new} := L + (H - L) \times P_u$

II.2 QM basic version:

Because now we only have two symbol, one is of more probability so we change to the following notations:

Notation:
C: the lower bound of the interval
A: the length of the interval
MPS: More Probable Symbol
LPS: Less Probable Symbol
Q: Probability of LPS

Compare to the basic version:

$L \Rightarrow C$

$H - L \Rightarrow A$

$for\ MPS : P_l \Rightarrow 1 - Q, P_u \Rightarrow 1$

$for\ LPS : P_l \Rightarrow 0, P_u \Rightarrow Q$

So we have the following version encoding method:

```
1 MPS is encoded:
2   C := C;
3   A := A(1−Q);
4 LPS is encoded:
5   C:= C + A(1−Q);
6   A:= AQ;
```

III. Optimization & Implementation

To increase the efficiency of the coding, we need to make sure the A is maintained as 1 so that the operation os multiplication would be eliminated. In order not to violate the assumption that A is close to 1. Whenever A goes below 0.75 the QM coder goes through a series of rescaling until the value of A goes higher than 0.75. The rescaling operation is simply doubling which corresponds to a left shift if A is represented in binary.The modified algorithm is:

```
1 MPS is encoded:
2   C := C;
3   A := A−Q;
4   if(A<0.75) then
5     Renormalize A and C;
6   fi
7 LPS is encoded:
8   C:= C + A − Q;
9   A:= Q;
10  Renormalize A and C;
```

One problem introduced by approximated renormalization is that the subinterval allocated to the MPS may be smaller than the subinterval allocated to the LPS. This is because we use A= A-Qe to approximate A=A(1-Qe). For example, suppose we have Qe=0.45, and we have four MPS in a row. Initially, we have C=0, A = 1. After first MPS, A becomes 0.55, and renormalized to 1.1, while C remains 0. After second MPS, A becomes 0.65, and re-normalized to 1.3, while C remains 0. After third MPS, A becomes 0.85, and no re-normalization is performed. When we encode the fourth MPS, the subinterval allocated to the MPS becomes $A - Qe = 0.40$, while the sub-interval allocated to LPS is 0.45. Now, the sub-interval allocated to MPS is smaller than the LPS subinterval. It can be seen that the condition for this situation is that Qe > A / 2, or Qe > A - Qe. The solution is to perform conditional exchange: interchanging the two intervals under the afore-mentioned condition. Because conditional exchange is caused by the approximated re-normalization, the test for conditional exchange is only performed when renormalization is needed.

Final Implementation

```
1 MPS is encoded:
2   C := C;
3   A := A−Qe;
4   if(A<0.75) then
5     if(A<Qe) then
6       C:=C+A
```

```
7         A:=Qe
8       fi
9       Renormalize A and C;
10    fi
11  LPS is encoded:
12    if (A-Qe >= Qe)
13      C:= C + A - Qe;
14      A:= Qe;
15    else
16      A:= A - Qe;
17    fi
18    Renormalize A and C;
```

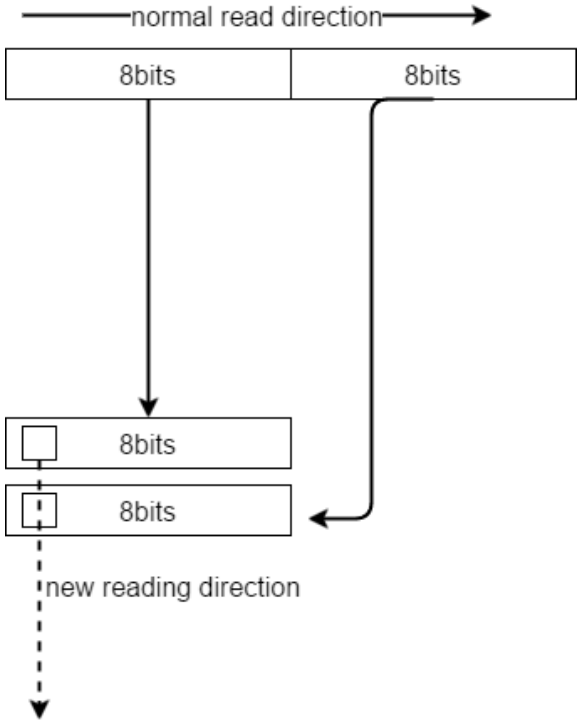


Figure 9: *Bitplane illustration*

IV. Implementation & Result

IV.1 Implementation

According to the pseudo code, the implementation is straight forward.Transition table see appendix.See file 'code.pdf' to check the codes. Also two mapping method for the data file is implemented. One is bitplane mapping and the other is just the normal mapping.The bit plane mapping can be illustrated in the following diagram:

The Transition table is in Appendix. The Program has two input Run the executive file in the coonsole, I would give the hint on how to use the program.

platform:Windows 10_x64

```
C:\Users\lychee\Documents\USC\EE669\hw2\codes\QMCoder>QMCoder.exe
Usage: QMCoder.exe filename [mapping type]
Example: >> QMCoder.exe ./data/text.dat normal
mapping in normal way
Example: >> QMCoder.exe ./data/text.dat bit
bit plane mapping
```

Figure 10: *run the programme*

One example:

```
C:\Users\lychee\Documents\USC\EE669\hw2\codes\QMCoder>QMCoder.exe ../data/text.dat bit
Reading8698characters ....
all characters read successfully
output size(byte):6249
compression ratio:0.718441
```

Figure 11: *compress sample text file*

IV.2 Result

Initial condition: MPS = 0,Current State=12, A=0x10000,C=0x0000,carry = false

Table 11: *Result : QM coding*

file	Origin size	Normal mapping		Bitplane mapping	
		size	ratio	size	ratio
text	8698	10513	1.20867	6249	0.71844
audio	65536	78800	1.20239	61571	0.93950
image	65536	79698	1.21609	63235	0.96469
binary	65536	6505	0.099258	7922	0.12088

Comment: Obviously , if using normal mapping, no compression is implemented, actually this is because the Qe transition table we are using is simple, here simple I mean the gap of two Qe symbols is large. From the table, we choose the initial state to be 0x1EDF, and it is easy to let A drop below 0x8000 and give output. Moreover, if the file is not preprocessed, the MPS and LPS may appear one after one which leads to less coding efficiency since every time we receive an LPS, we give output. If we map the bits string using bitplane method, we achieve some compression. This is easy to understand, take the text file for instance. We all know the alphabets in ASCII code table are consecutive and the largest gap is 'a' and 'z' which is 25, that means if transformed to binary codes, most of the consecutive two letters share same binary codes for the first three or four bits. Therefore, if we use bitplane coding, we can link a lot of same bits together.

V. CABAC

V.1 Motivation & Introduction

Context-adaptive binary arithmetic coding (CABAC) is a form of entropy encoding used in the H.264/MPEG-4 AVC and High Efficiency Video Coding (HEVC) standards. It is a lossless compression technique, although the video coding standards in which it is used are typically for lossy compression applications. CABAC is notable for providing much better compression than most other entropy encoding algorithms used in video encoding, and it is one of the key elements that provides the H.264/AVC encoding scheme with better compression capability than its predecessors.

V.2 Implementation

We have a QM coder implemented with a class type, therefore it is easy for us just to create an instance and call the methods we need.

implementation in main

```
1  FILE *fp;
2  fp = fopen(outputFilename.c_str(),"w+");
3  QM oneTrial(fp);
4  //initiate
5  oneTrial.StartQM("encode");
6  //initiate context
7  for (int i = 0; i < n; i++) {
8    context += '0';
9  }
10
11 for(auto bit:inputbitString){
12   int getContext = bitStrToInt(context);
13   oneTrial.encode(bit - '0', getContext)
         ;
14   for (int i = 1; i < n ; i++) {
15     context.at(i - 1) = context.at(i);
16   }
17   context.at(n - 1) = bit;
18 }
```

Line3: a QM instance is created and denote the file to be processed
Line5: I call the method StartQM to let the coding type be encoding.
Line7-8: append the n '0's to the begin of the file, n is the length of context
Following: call the encode mthod and give the input context and current bit.

Build it into a program and execute it from console in the following way , in the program you can choose the length of context.

```
C:\Users\lychee\Documents\USC\EE669\hw2\codes\CABAC>CABAC.exe ../data/text.dat normal
input context size 1 to 3:1
1
Reading8698characters ....
all characters read successfully
normal mapping
coding done!
file saved to :../data/text1EncodeCABAC.dat
compression ratio:1.04012
```

Figure 12: *CABAC program example*

Run the commands to different files and different length of context, we get the following result:

Table 12: *Result: CABAC*

File	Origin size	Compression Ratio					
		Normal mapping			Bitplane mapping		
		n=1	n=2	n=3	n=1	n=2	n=3
text	8698	1.04012	1.03219	1.02127	0.616234	0.602897	0.600943
image	65536	1.02563	1.01935	1.01844	0.746078	0.741302	0.738907
audio	65536	1.01462	1.01335	1.00967	0.560944	0.559662	0.55777
binary	65536	0.0293121	0.0289459	0.0285492	0.101807	0.101044	0.100189

V.3 Comments

Clearly with n increasing we can obtain better compression, moreover, it seems that the audio file and image file achieve better compression after applying bitplane transformation and CABAC. First is the reason of the characteristic of audio and image file that lower frequency usually takes the main position therefore the diffrentiation between two pixels or two sample of audio are small. We can expect small variation between two continous symbols. And this can be the reason the CABAC are used for video coding, because it not only achieves compression, but effeciency and almost real-time as well.

IV. Scalar & Vector Quantization

I. Lloyd-Max Scalar Quantizer Design

I.1 Background and Introduction

Like the closely related k-means clustering algorithm, Lloyd's algorithm repeatedly finds

the centroid of each set in the partition and then re-partitions the input according to which of these centroids is closest.

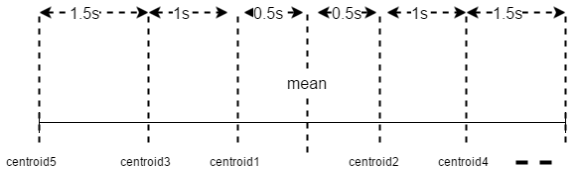


Figure 14: *initialization,s:stddev*

I.2 Input image information

Since I am going to use non-linear quantilizar,I plot the images' histogram to see the distribution of the pixels' magnitude.

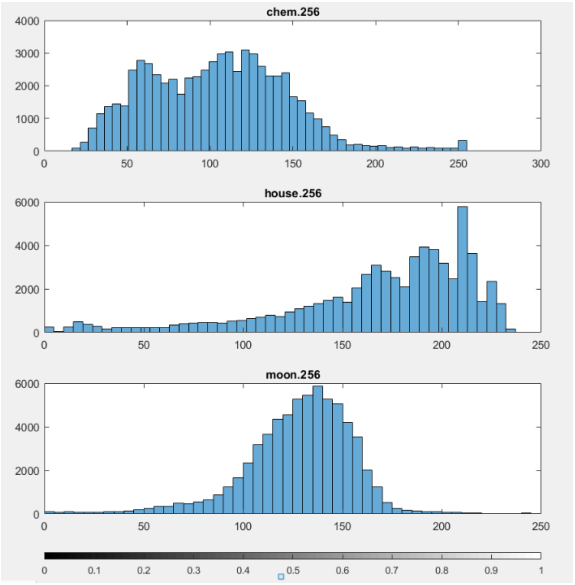


Figure 13: *Image Histogram*

I.3 Implementation

Initiation method:

To initiate the centroids, I first calculate the mean and standard deviation of the datas of the given data, A class named lloydmax is created, which has public functions for users to call initilazation. The following figure show how I initilize the centroids.s is the standard deviation of all the training data.

Since if we were given a large amount of data, we may assume laws of large number, my assumption is all image may subject to some distribution, and we use the nonlinear standard deviation to get the initial centriods.

Update routine

In my code design, and in my head I create a desk with n drawers, n is the number of classification, each drawer has a label which is the centroid, and in the drawer are the data. the flowchart is the following:

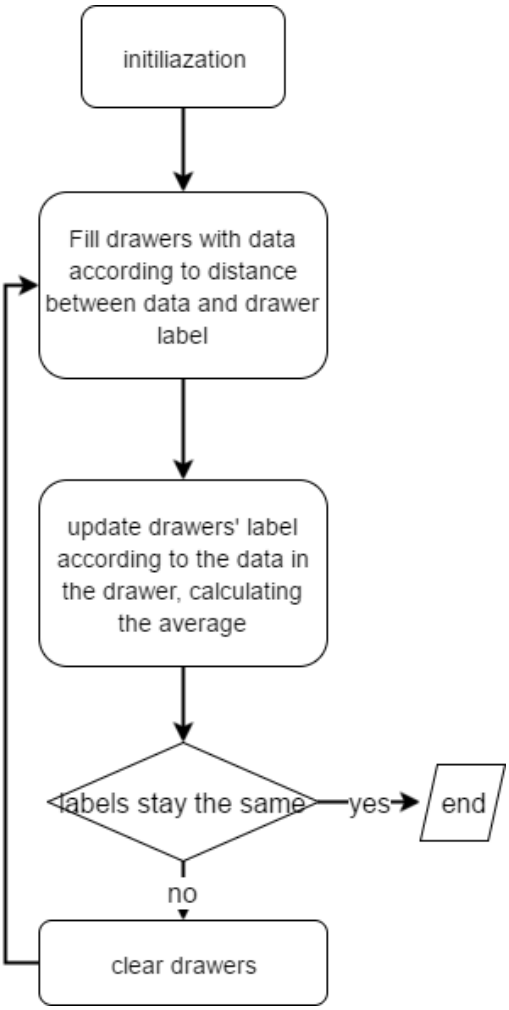


Figure 15: *update routine*

I.4 Results

2bits training

Reading65536characters ....
all characters read successfully
Reading65536characters ....
all characters read successfully
Reading65536characters ....
all characters read successfully
train data size:196608
initial centers: 85.1457 109.341 157.732 181.927
training 1: PSNR:21.904
training 2: PSNR:24.77
training 3: PSNR:24.9586
training 4: PSNR:24.9964
training 5: PSNR:25.0148
training 6: PSNR:25.0167
training 7: PSNR:25.0167
end of traning!

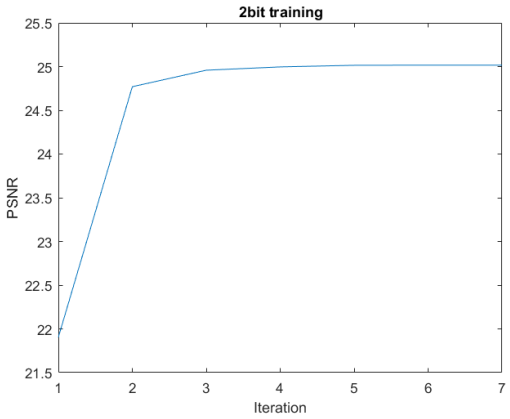


Figure 16: *Convergence 2 bits*

4bits training

training 1: PSNR:23.0269
training 2: PSNR:28.8788
training 3: PSNR:30.453
training 4: PSNR:31.4084
training 5: PSNR:32.0361
training 6: PSNR:32.5038
training 7: PSNR:32.8975
training 8: PSNR:33.2085
training 9: PSNR:33.4337
training 10: PSNR:33.6686
training 11: PSNR:33.8619
training 12: PSNR:33.9968
training 13: PSNR:34.1199
training 14: PSNR:34.2092
training 15: PSNR:34.295
training 16: PSNR:34.3514
training 17: PSNR:34.4096

training 18: PSNR:34.4725
training 19: PSNR:34.5137
training 20: PSNR:34.5424
training 21: PSNR:34.5507
training 22: PSNR:34.5793
training 23: PSNR:34.5903
training 24: PSNR:34.61
training 25: PSNR:34.6274
training 26: PSNR:34.6522
training 27: PSNR:34.6792
training 28: PSNR:34.6946
training 29: PSNR:34.7118
training 30: PSNR:34.7228
training 31: PSNR:34.7293
training 32: PSNR:34.7293
end of traning!

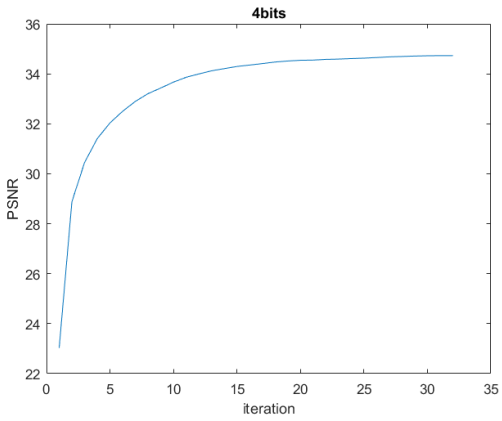


Figure 17: *Convergence 4bit*

I.5 quantized image

quantize the test images using the centroids generated above

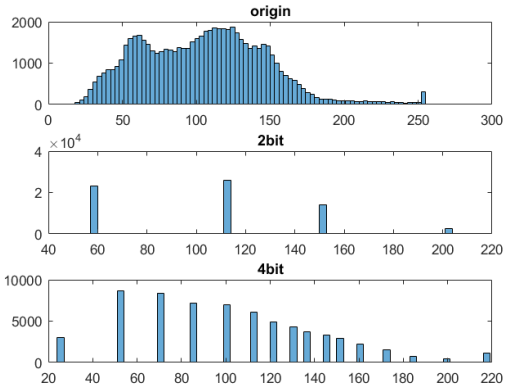


Figure 18: *Histogram for chem.256*

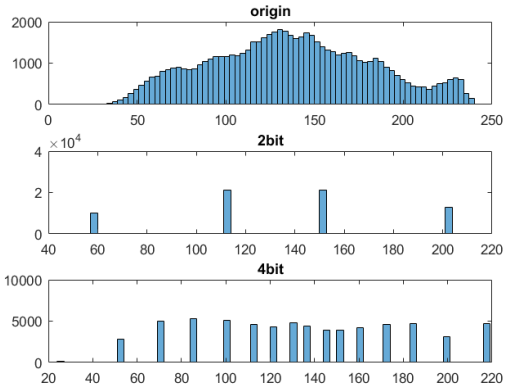


Figure 19: *Histogram for elaine.256*

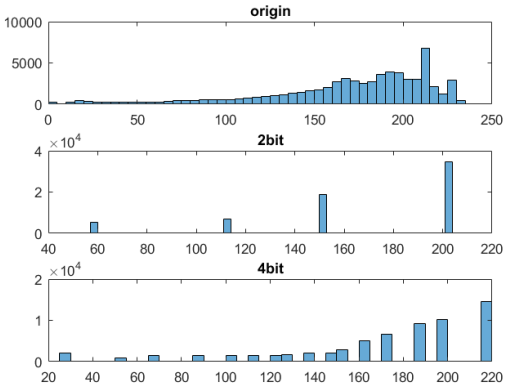


Figure 20: *Histogram for house.256*

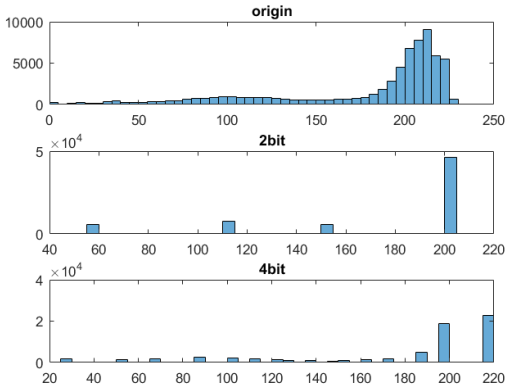


Figure 21: *Histogram for f16.256*

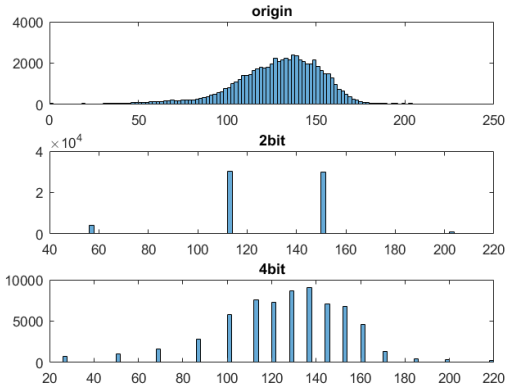


Figure 22: *Histogram for moon.256*

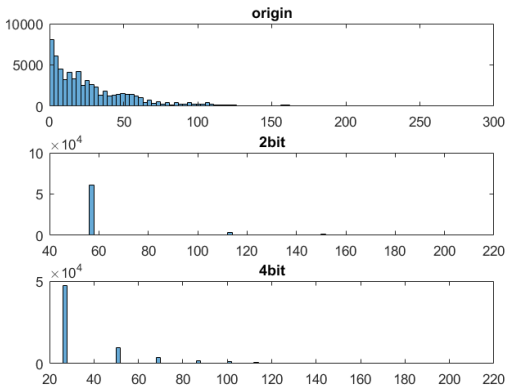


Figure 23: *Histogram for couple.256*

Figure 24: *Image effect after quantize chem.256*

Figure 25: *Image effect after quantize couple.256*

Figure 26: *Image effect after quantize elaine.256*

Figure 27: *Image effect after quantize f16.256*

Figure 28: *Image effect after quantize house.256*

Figure 29: *Image effect after quantize moon.256*

The table showing the PSNR and the entropy

Table 13: *PSNR/bbp & PSNR/Entropy*

	2bits					
file	chem	house	moon	couple	f16	elaine
PSNR	24.836	24.236	26.206	16.449	25.445	24.840
Entropy	1.7131	1.6371	1.3791	0.4278	1.3279	1.9295
	4bit					
file	chem	house	moon	couple	f16	elaine
PSNR	33.740	33.828	37.625	25.109	33.856	35.491
Entropy	3.6893	3.4624	3.4831	1.4941	2.9050	3.9018

The two rate distortion pairs and the average code word lenth figure

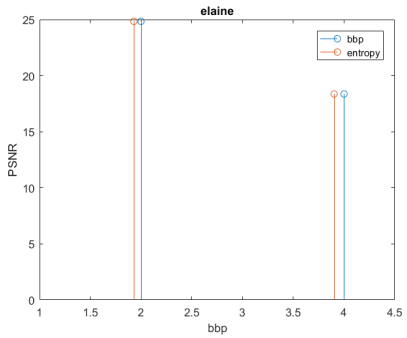


Figure 30: *elaine*

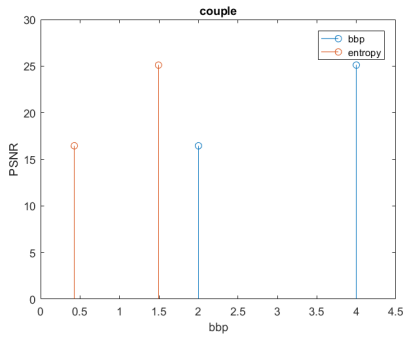


Figure 32: *couple*

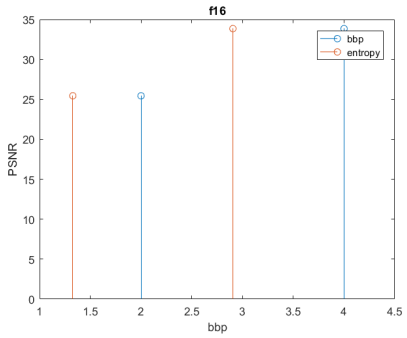


Figure 31: *f16*

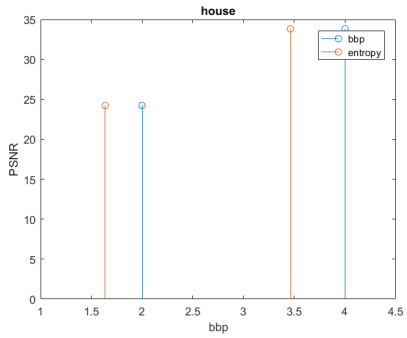


Figure 33: *house*

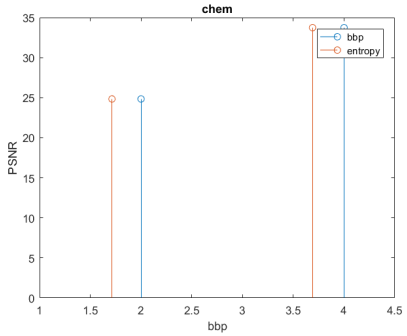


Figure 34: *chem*

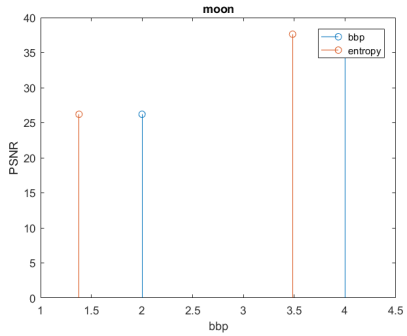


Figure 35: *moon*

I.6 Comments

II. Vector Quantization

II.1 Introduction

Vector quantization is also a kind of technique for "scalaring", instead one to one scale, it uses vector to vector scale, that is one vector maps to another vector according to the distance bectween the vector to the centroids vectors in the codebook. It is well-known technique for lossy data compression.The following figure demonstrates using a block codebook to quantize an image.

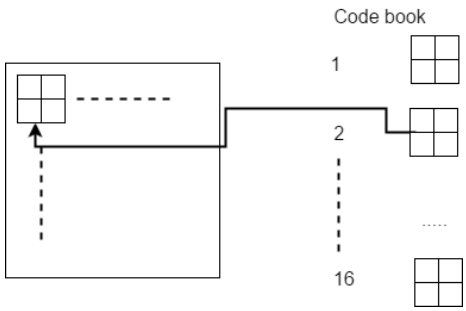


Figure 36: *An example of VQ quantization using block* 2 *codebook size 16*

II.2 Implementation

The source code of standard vector quantization is given. To use that source code, it is better to switch to Linux system, or at least some system that support static library compilation. Because the code is old and it uses /bin/ranlib and /bin/ar for building and compilation between objects, if that is not exist in the user's root binary code or in the path, the project would not build.

Image blocking The implementation of Image blocking is easy, if one picture is blocked into several $n \times n$ blocks the core code is obvious and easy:(if the original image is 256×256)

```
1  for (int i = 0; i < 257 - n; i += n) {
2    for (int j = 0; j < 257 - n; j += n) {
3      for (int u = 0; u < n; u++) {
4        for (int v = 0; v < n; v++) {
5          reorderedData.push_back(
               imagedata[i + u][j + v]);
6        }
7      }
8    }
9  }
```

Standard Vector Quantization

•1. First we need to block the training images to form the training set. the training images are f16.256,couple.256 and elaine.256, using the blocking program, reorder image pixels data

with blocks(2×2), and concatenant together. Do the same for 4×4, and 8×8. And we get the following data for training.

```
-rw-rw-r-- 1 lychee lychee 196608 Sep 29 12:59 outblock2.raw
-rw-rw-r-- 1 lychee lychee 196608 Sep 29 12:59 outblock4.raw
-rw-rw-r-- 1 lychee lychee 196608 Oct  1 22:24 outblock8.raw
```

Figure 37: *training data*

•2. Second we generate codebooks with different size and different block dimension by using the source code. For exmaple,To generate codebook with 4×4 block and codebook size 32, we can run the following application in console:

```
lychee@lychee-MacBookPro:~/Documents/EE669/hw2/stdvq$ ./stdvq -t outblock4.raw -
c block4size32 -d 16 -f 32 -h 0.001

OPTIONS   DESCRIPTIONS                          SETTINGS
-t        training sequence                    outblock4.raw
-c        codebook name                        block4size32
-d        vector dimension                     16
-f        codebook size                        32
-h        convergence threshold                0.001
-a        codeword split additive offset       0.01
-m        codeword split multiplicative offset 0.01
-s        constrained search choice            1

Codebook distortion of size 1      :  92678.257334
Codebook distortion of size 2      :  23622.921375
Codebook distortion of size 4      :  8564.845964
Codebook distortion of size 8      :  5673.300118
Codebook distortion of size 16     :  4087.552862
Codebook distortion of size 32     :  3030.954892
```

Figure 38: *exmaple of generating codebook*

the argument follows by -t is the training set

-c is the output codebook name here i use "block4size32" to mean that it is a codebook with size 32 and vector 4×4

-d is the size of vector because here is 4×4 so it is 16

-f is the size of the codebook

-h is the accuracy of training

Repeat this for other parameters and we get 9 codebooks:

```
-rw-rw-r-- 1 lychee lychee    520 Sep 29 13:09 block2size16
-rw-rw-r-- 1 lychee lychee   1032 Oct  1 22:18 block2size32
-rw-rw-r-- 1 lychee lychee   2056 Oct  1 22:18 block2size64
-rw-rw-r-- 1 lychee lychee   2056 Oct  1 22:27 block4size16
-rw-rw-r-- 1 lychee lychee   4104 Oct  1 22:28 block4size32
-rw-rw-r-- 1 lychee lychee   8200 Oct  1 22:28 block4size64
-rw-rw-r-- 1 lychee lychee   8200 Oct  1 22:30 block8size16
-rw-rw-r-- 1 lychee lychee  16392 Oct  1 22:31 block8size32
-rw-rw-r-- 1 lychee lychee  32776 Oct  1 22:31 block8size64
```

Figure 39: *codebooks*

•3.Finally , we can use "./stdvqe" to compress the test files, chem.256,house.256 and moon.256. An example is shown below the compression program: Importance before we compress the images we should also using the block program to reorder the pixels of the training images for the compression. For example, if we need to compress the moon.256 using the codebook that is for block 4×4, we need first reorder the image data from moon.256 using the block diagram with parameter 4.

```
lychee@lychee-MacBookPro:~/Documents/EE669/hw2/stdvq
$ ./stdvqe -c block2size16 -i chem.256 -o chemb2c16
-D

File to encode:       chem.256
Codebook file:        block2size16
Number of Codewords:  16
Encoded file:         chemb2c16
Vectors encoded:      16384
Average distortion:   783.549191
Rate (bpv):           4.000000
Empirical entropy:    3.143227


Codeword  Count     Distortion

0         0         0.000000
1         6         102.377197
2         311       138.207128
3         964       235.688829
4         1993      346.476743
5         2199      728.430125
6         2437      938.543236
7         3159      911.473719
8         0         0.000000
9         2311      809.774596
10        1811      885.938982
11        611       901.654179
12        182       1757.809850
13        115       1586.757508
14        69        1153.883418
15        216       2311.093293
```

Figure 40: *An example of Compression*

II.3 Result:

Table 14: *Vector Quantization On Testing Data Result*

$\mathbb{H}$:Empirical Entropy ϵ: Average Distortion $\mathbb{S}$:Codebook Size										
		block 2×2			block4 $\times$ 4			block8 $\times$ 8		
$\mathbb{S}$		chem	house	moon	chem	house	moon	chem	house	moon
16	$\mathbb{H}$	3.1780	3.3394	2.5586	3.0991	3.2222	2.3599	3.0407	3.1380	2.5468
	ϵ	591.012	1031.366	331.860	5555.307	7907.601	2329.792	36481.76	48640.43	18108.59
32	$\mathbb{H}$	3.8787	4.1173	3.0154	3.8274	4.0957	2.8552	3.5848	3.9590	2.8595
	ϵ	458.784	705.360	255.551	4569.624	6551.293	1835.414	32445.75	43567.33	12882.18
64	$\mathbb{H}$	4.7548	4.9889	3.8958	4.6713	4.8775	3.4452	4.3772	4.6702	3.2327
	ϵ	340.378	510.381	195.369	3876.931	5508.725	1596.325	30021.76	39233.64	10676.24

II.4 Comment

•1 Clearly increase the codebook size improves the quality,as we can see the distortion goes lower. The reason is obvious since we have more different choice to quantize the image with larger size of codebook. However, it also increase the bits/pixel.
•2 The test data are independent of the training data, therefore some image may not perform a good quantization with the trained code book.
•3 Increase the size of the block will degrade the quality of quantized image, this is also obvious because the VQ quantize the blocks as a whole, so there may be noise and sharp edge be introduced to the edge of the blocks.
•4 Coding process is slow since the block have to compare with all the entries in the codebook, the following tree structured vector quantization tend to solve the problem.

III. Tree Structured Vector Quantization(TSVQ)

III.1 Introduction

TSVQ has an $O(logN)$ time complexity per vector for encoding where N is the number of code vectors. The encoding complexity for the TSVQ compares favorably to the $O(N)$ encoding complexity for the exhaustive full search which is generally required for VQ with an optimal codebook.

III.2 Implementation

We still use the Ubuntu system and follow the Usage document of the tsvq source code.
Using the blocked training sequence from the previous section
Train the data:**using ./tsvq**

```
lychee@lychee-MacBookPro:~/Documents/EE669/hw2/tsvq$ ./tsvq
Usage: ./tsvq -t trainingsequence -c codebook -s statfile
   -d dimension -r rate -m offset -h threshold -B

OPTIONS   DESCRIPTIONS                        DEFAULTS
-t        blocked training sequence
-c        codebook
-s        codebook statistics
-d        vector dimension                    4
-r        stopping rate (bits per vector)     0
-m        multiplicative offset               0.01
-h        convergence threshold               0
-B        produce a balanced tree
```

Figure 41: *Generate Codebook*

encoding the files: **using ./tsvqe**

```
lychee@lychee-MacBookPro:~/Documents/EE669/hw2/tsvq$ ./tsvqe
-c block2c -i chemout2raw -D

Codebook file:              block2c
Vector dimension:           4
Number of Nodes:            137
Image to encode:            chemout2raw
Encoded file:               chemout2raw.tsvq
Number of pixels encoded:   65536
Average rate:               4.689819
Empirical entropy:          3.503692
Average distortion:         527.630812
Maximum codeword length:    12
Statistics file:            chemout2raw.tsvq.stat
```

Figure 42: *Example of TSVQ encoding*

III.3 Result

Table 15: *Bitrate 4.0*

Bitrate 4.0	$\mathbb{H}$: Empirical Entropy		ϵ: Averge Distortion	
image		2×2	4×4	8×8
chem	$\mathbb{H}$	3.504	3.594	2.949
	ϵ	527.6	5041.8	36643.1
house	$\mathbb{H}$	3.437	3.438	3.158
	ϵ	907.4	7072.2	48695.6
moon	$\mathbb{H}$	2.795	2.723	1.973
	ϵ	286.3	2082.0	16017.7

Table 16: *Bitrate 5.0*

Bitrate 5.0	$\mathbb{H}$: Empirical Entropy		ϵ: Averge Distortion	
image		2×2	4×4	8×8
chem	$\mathbb{H}$	3.905	4.318	4.171
	ϵ	426.3	4135.2	31887.5
house	$\mathbb{H}$	4.249	4.410	4.083
	ϵ	616.2	5765.3	42168.3
moon	$\mathbb{H}$	2.988	2.883	2.871
	ϵ	261.4	1929.3	11636.5

Table 17: *Bitrate 6.0*

Bitrate 6.0	$\mathbb{H}$: Empirical Entropy		ϵ: Averge Distortion	
image		2×2	4×4	8×8
chem	$\mathbb{H}$	5.230	5.160	4.609
	ϵ	284.6	3645.3	29589.3
house	$\mathbb{H}$	5.120	4.990	4.954
	ϵ	365.7	5061.8	39676.8
moon	$\mathbb{H}$	3.866	3.334	2.988
	ϵ	192.9	1665.6	11100.7

III.4 Comment

•1 :Increase bitrate leads to better quality of image, but also increase the size.

•2 :Increase the block size leads to lower quality the reason is the same explained above in STDVQ.

•3 :Some images are sensitive to the codebook. Because the test data and training set are isolated.

V. JPEG Compression Standard

I. DCT and Quantization for JPEG

DCT(Discrete Cosine Transform) is a Fourier-related transform similar to the discrete Fourier transform (DFT). But because DFT gives complex spectrum data, to implement some application in engineering, we use only the real numbers. Therefore, DCT is generated, to express a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), which would be observed in the following experiments. In quantization, we use Qmatrix to quantize the transformed image.

II. Experiments

•1.We are give an image of size 16×16, we block the image into 4 8×8 blocks.
•2. For each block, we first substract 128 from the pixels magnitude then perform the DCT transform.
•3.Using three different Quantization matrix to quantize the coefficients.

The first part is done in previous section 2.2.2, for the DCT we use the following formular: an image is given by

$$D_{i,j} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p_{x,y} \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)i\pi}{2N}\right)$$

The third part we use the quantizers Q50, and using the following matrix to transorm between different quantize matrix.

$$Q_n(i,j) = \begin{cases} \frac{100-N}{50} Q_{50}(i,j) & N > 50 \\ \frac{50}{N} Q_{50}(i,j) & N < 50 \end{cases}$$ **the Q50 matrix**

```
int Q50[8][8] = {
  {16,11,10,16,24,40,51,61},
  {12,12,14,19,26,58,60,55},
  {14,13,16,24,40,57,69,56},
  {14,17,22,29,51,87,80,62},
  {18,22,37,56,68,109,103,77},
  {24,35,55,64,81,104,113,92},
  {49,64,78,87,103,121,120,101},
  {72,92,95,98,112,100,103,99}
};
```

II.1 Result

```
the dct coefficients after Q50:
-16  -17    3    1    0    0    0    0
-18   -2   -1    3    0    0    0    0
 -2    1   -2    0    0    0    0    0
  1   -1    2   -1    0    0    0    0
  1    2    0    0    0    0    0    0
 -1    0    0    0    0    0    0    0
 -1    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0

-26   14    8    1   -1    0    0    0
-16   -6   -4    0    1    0    0    0
  0   -4   -2    0    0    0    0    0
  2    4    2    0    0    0    0    0
  2    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    0    0    0    0    0    0

-20   -1   10   -3    1   -1    0    0
 22   -4   -8    2   -2    1    0    0
  0   -3    2    0    0    0    0    0
 -4    3    0   -1    0    0    0    0
  1   -1    0    0    0    0    0    0
  1   -1    0    0    0    0    0    0
  0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0

-20   20   11   -2   -1    0    1    0
 15   -9    0    4    1    0   -1   -1
  1    4    0    1   -1   -1    0    0
 -1    3   -1   -2    0    0    0    0
 -1   -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    0    0    0    0    0
```

Figure 43: *4 blocks coefficients after quantization with Q50 matrix*

```
the dct coefficients after Q10:
 -3   -3    1    0    0    0    0    0
 -4    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    0    0    0    0
  0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0

 -5    3    2    0    0    0    0    0
 -3   -1   -1    0    0    0    0    0
  0   -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    0    0    0    0    0    0
  0    0    0    0    0    0    0    0

 -4    0    2   -1    0    0    0    0
  4   -1   -2    0    0    0    0    0
  0   -1    0    0    0    0    0    0
 -1    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    0    0    0    0    0    0
  0    0    0    0    0    0    0    0

 -4    4    2    0    0    0    0    0
  3   -2    0    1    0    0    0    0
  0    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    0    0    0    0    0    0
  0    0    0    0    0    0    0    0
```

Figure 44: *4 blocks coefficients after quantization with Q10 matrix*

```
the dct coefficients after Q90:
 -78   -87    14    4   -1    1    1   -1
 -88   -11    -5   13   -2    2    0   -1
 -10     7   -10    1    0   -1   -1    0
   3    -6     8   -4    0   -1    0    0
   6     8     2   -1    0    0    0   -1
  -5    -1    -1    1    1    0    0    0
  -5    -2    -1    1    0    0    0    0
   2     2    -1   -1    0    0    0    0

-130    70    41    3   -5    2    2   -1
 -78   -29   -19    1    4   -1    0    1
  -2   -21   -12    0    2   -1   -1    1
  10    18     9   -1   -1    1    0   -1
   8     5     1    0    0    0    0    0
  -2     0     0    0    0    0    0    0
  -2    -2    -1    0    0    0    0    0
  -1    -1     0    0    0    0    0    0

 -99    -4    48  -15    6   -5    1    0
 111   -21   -40   11  -10    4   -2    2
  -2   -16    12    2    2    0   -1   -1
 -19    16     2   -3   -2    0    0    0
   7    -5     0   -2    1    0    0    0
   6    -4     0    1    0    0    0    0
  -1     0     0    0    0    0    0    0
  -1     1     0   -1    0    0    0    0

 -99   101    53  -12   -7    1    3    1
  76   -46     1   18    5   -2   -3   -3
   4    18     2    7   -6   -5    0    2
  -3    16    -6   -8    1    2    2    1
  -3    -5     2    0    2    0   -2   -2
   4     1     2    1   -1    0    1    1
   0     1     0    0   -1    0    0    0
  -1    -1     0    0    0    0    0    0
```

Figure 45: *4 blocks coefficients after quantization with Q90 matrix*

II.2 Comments

why such nonlinear quantization is applicable considering the characteristic of the human visual system

First we recover the coefficients by multipling element by element with the Q50 matrix. Then we do the iDCT and check the difference. Obviously only the high frequency part is eliminated, and almost 25% low frequency data is preserved(see the result above), and for human eye , eliminate some higher frequency just lead the image not so sharp in some edges or noise, which not affect much of the image quality.

Which of the quantized matrix id best for entropy coding

If under the circumstance that we use zigzag coding, clearly the quantize matrix Q10 is best for entropy coding. However the problem is that such matrix cut so much part of high frequency data that we may lose some information. Therefore , we should find the suitable quantization matrix to achieve a balance between compression quality and entropy coding.

III. JPEG Compression Quality Factor

III.1 Implementation

First, let us change to my Ubuntu system again. We download the JPEG software, we first install the software by using the command './config'. After that, we can find the makefile in the folder, using the command 'make', we get the executive files.

cjpeg [switches] [imagefile] >jpegfile : this is the general use of this software.

Becasue we would like to see how quality factors affect the compression, The command line switches for cjpeg are:

-quality: N Scale quantization tables to adjust image quality. Quality is 0 (worst) to 100 (best); default is 75.

Because we are also only compress the a grayscale image, we use -grayscale command.

Example:

```
lychee@lychee-MacBookPro:~/Documents/jpeg-6b$ ./
cjpeg -quality 60 -grayscale clock.bmp >clock60
```

Figure 46: *example of run testing*

Table 18: *Compression Ratio & PSNR*

	Compressed Size	Compress Ratio(%)	PSNR(dB)
100	36900	55.39	58.4932
60	6215	9.33	35.8304
40	4858	7.29	34.0552
20	3470	5.21	31.5612
10	2525	3.79	28.7632
1	1229	1.84	18.7949

III.2 Result

After running with different quality factors: we got the following result:

```
lychee   1229 Oct  5 01:11 clock1
lychee   2525 Oct  5 01:11 clock10
lychee  36900 Oct  5 01:12 clock100
lychee   3470 Oct  5 01:11 clock20
lychee   4858 Oct  5 01:11 clock40
lychee   6215 Oct  5 01:08 clock60
lychee  66614 Feb 25  2009 clock.bmp
```

Figure 47: *Compressed file*

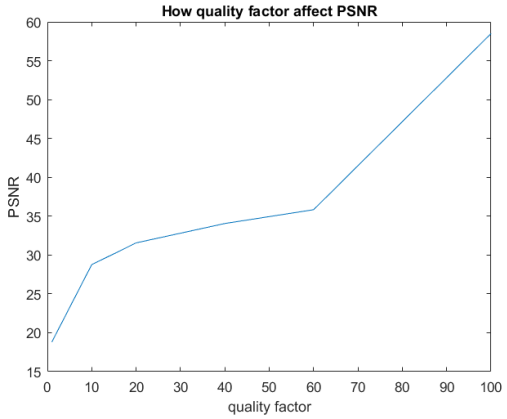


Figure 48: *PSNR AND QUALITY FACTOR*

The Compressed Images And Visual Artifact

Figure 49: *clock (quality factor 1)*

Figure 50: *clock (quality factor 10)*

Figure 51: *clock (quality factor 20)*

Figure 52: *clock (quality factor 40)*

Figure 53: *clock (quality factor 60)*

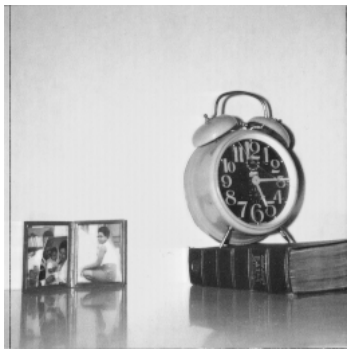


Figure 54: *origin*

III.3 Comments

•1Obviously, higher quality factor gives subjectively better image quality but worse compression ratio.

•2Lower quality factor wipes out high frequency information, including edges, small dots, salt-like noises.

IV. Post-Processing of JPEG Encoded Images

IV.1 Experiments

Convert the given data into raw format and compute the PSNR
We use the **./djpeg** to convert image to a raw file and compute the PSNR
result:

Table 19: *PSNR 10 testing jpeg images*

	clock1	clock2	clock3	clock4	clock5
PSNR	27.77	29.50	30.73	32.24	33.44
	pepper1	pepper2	pepper3	pepper4	pepper5
PSNR	23.61	24.92	25.87	26.98	27.77

Intuitive Trial : My Method Now since we see the artifact happens along the block boundary, it is easy to design some LPF(Low Pass Filter) along the boundary. The easiest one is the average window. If we design the window with length 3, then the pixel is $x_i = \frac{x_{i-1}+x_i+x_{i+1}}{3}$. Using the knowledge in digital signal processing, we know we are actually convolute the kernel $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$. But to me, I tend like to use the Gaussian kernel, that is a weighted average, Here I choose the kernel size to be 5, the kernel is :
Therefore, for the pixel at $(8n, 8m), n \in \mathbb{N}, m \in \mathbb{N}$, we do the average filtering.
In the actural experiment, I used the 5 pixels along the block edge, I first filter along the col-

umn, then I filter along the row: the algorithm is illustrated below:

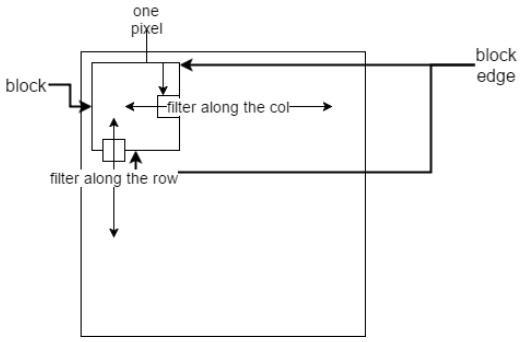


Figure 55: *Where to do the filter*

I use the average filter, the core code as follows:

```
for (int col = 7; col < 248; col+=8) {
  for (int row = 0; row < 255; row++) {
    Vec3d sum = 0;
    for (int i = -2; i < 3; i++) {
      sum += image.at<Vec3b>(col+i, row)
          ;

    }
    sum = sum / 5;
    image.at<Vec3b>(col, row) = (Vec3b)
        sum;
  }
}
for (int row = 7; row < 248; row += 8) {
  for (int col = 0; col < 255; col++) {
    Vec3d sum = 0;
    for (int i = -2; i < 3; i++) {
      sum += image.at<Vec3b>(col,row+i);
    }
    sum = sum / 5;
    image.at<Vec3b>(col, row) = (Vec3b)
        sum;
  }
}
```

My method Performance I give two example of my filter's performance (may be not so clear)

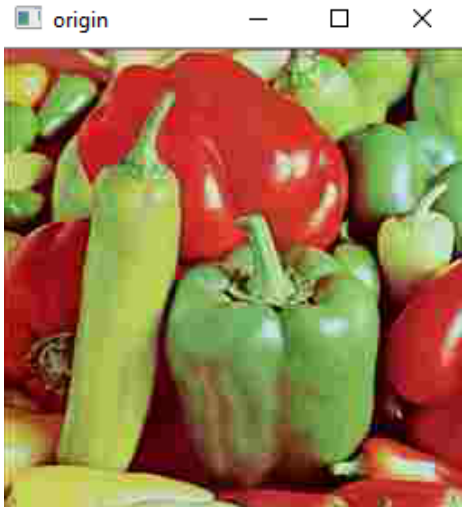


Figure 56: *original pepper2.jpg*

Figure 57: *after deblocking by average filter*

If you look at the highlight on the red pepper or the surface of green pepper, the edges become more smooth.
Do such operation on 10 images and get the following PSNR:

Table 20: *Deblocking using my method*

	clock1	clock2	clock3	clock4	clock5
PSNR	27.89	29.70	30.79	32.30	33.48
	pepper1	pepper2	pepper3	pepper4	pepper5
PSNR	24.56	25.32	26.09	27.11	27.90

A Deblocking Filter with Two Separate Modes in Block-Based Video Coding

The paper present a method for the deblocking process, the flowchart is as follows:

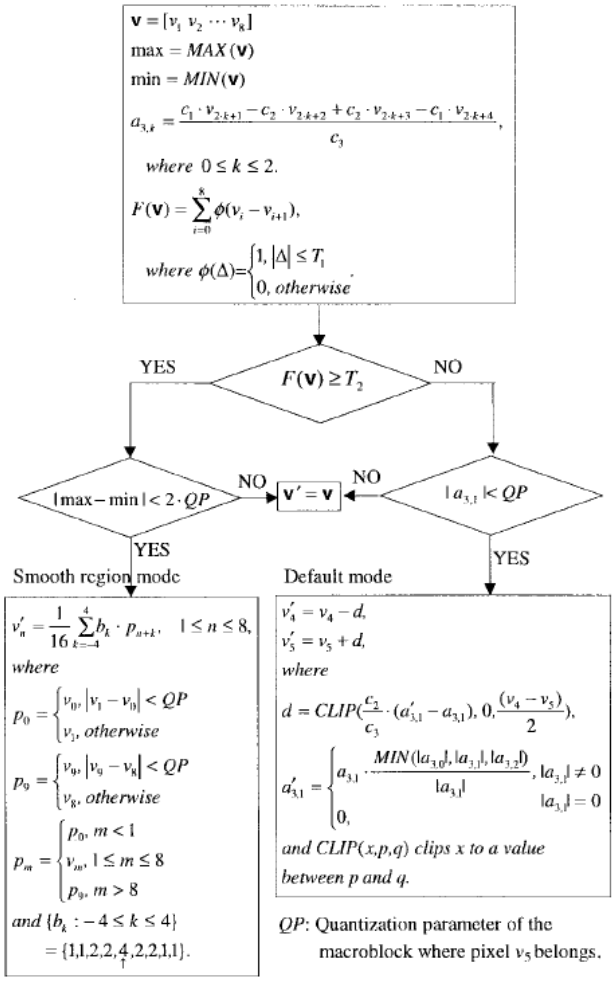


Figure 58: *deblock process*

Unlike my method, the paper using the 10 pixels orthogonal along the block edge, but it also uses the weighted average, with some other constraint optimization.
According to the smooth progress, we can see the C++ implementation of smooth method:

```
vector<Vec3b> getNewPixelSmooth(vector<Vec3b> pixels, int Qp) {
  Vec3d p0;
  Vec3d p9;
  int b[9] = {1, 1, 2, 2, 4, 2, 2, 1, 1};
  if (getDif(pixels.at(0), pixels.at(1))<Qp) {
    p0 = pixels.at(0);
  }
  else {
    p0 = pixels.at(1);
  }
  if (getDif(pixels.at(9), pixels.at(8))<Qp) {
    p9 = pixels.at(9);
  }
  else {
    p9 = pixels.at(8);
  }
  vector<Vec3d> ps;
  for (int i = 0; i < 3; i++) {
    ps.push_back((Vec3d)pixels.at(1));//p(-3) to p(-1) is v1
  }
  ps.push_back(p0);//p(0) is p(0)
  for (int i = 1; i < 9; i++) {
    ps.push_back((Vec3d)pixels.at(i));//p(1) to p(8) is vi
  }
  ps.push_back(p9);
  for (int i = 0; i < 3; i++) {
    ps.push_back((Vec3d)pixels.at(9));
  }

  for (int i = 1; i < 9; i++) {
    Vec3d newPixel;
    for (int j = i-1; j < i+8; i++) {
      newPixel += ps.at(j) * b[j+1-i];
    }
    newPixel = newPixel / 16;
    pixels.at(i) = newPixel;
  }
}
```

Result: For the paper method , I use the same parameters as it is in the paper(that is c1, c2 ,c3 is 2,5,8,respectively, T1 = 2, T2 = 6, The Q matrix I use the Q50).

Table 21: *Deblocking using method in the paper*

	clock1	clock2	clock3	clock4	clock5
PSNR	28.07	29.82	31.09	32.33	33.48
	pepper1	pepper2	pepper3	pepper4	pepper5
PSNR	24.08	25.24	26.04	27.05	27.78

We put three data together to compare the performance of the three method, with the PSNR,

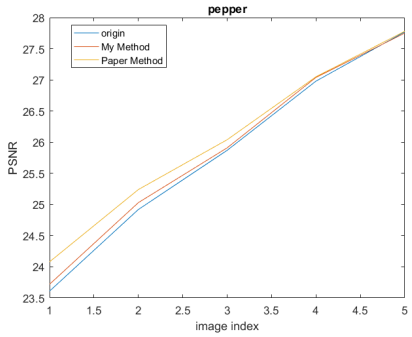


Figure 59: *Pepper deblock datas*

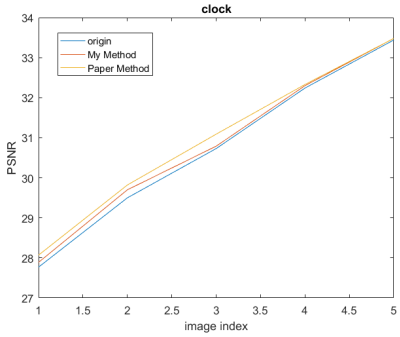


Figure 60: *Clock deblock datas*

Figure 61: *pepper1*

Figure 63: *pepper2*

Figure 62: *pepper1 deblocked*

Figure 64: *pepper2 deblocked*

Figure 65: *pepper3*

Figure 67: *pepper4*

Figure 66: *pepper3 deblocked*

Figure 68: *pepper4 deblocked*

Figure 69: *pepper5*

Figure 70: *pepper5 deblocked*

Figure 71: *clock1*

Figure 72: *clock1 deblocked*

Figure 73: *clock2*

Figure 75: *clock3*

Figure 74: *clock2 deblocked*

Figure 76: *clock3 deblocked*

Figure 77: *clock4*

Figure 79: *clock5*

Figure 78: *clock4 deblocked*

Figure 80: *clock5 deblocked*

Another method: reapplying_JPEG

The algorithm is really simple and straightforward
The algorithm is summarized below:
1. Shift the compressed images in vertical and horizontal directions by (i,j).
2. Apply JPEG to shifted image.
3. Shift the result back, i.e. vertically and horizontally by (-i,-j).
4. Repeat for all possible shifts in the range [-3,4]× [-3,4] 5. Average all images **expriment**
1.Write a program for the image shifting
2.

IV.2 Comment

•1 : the image subjective quality greatly increased if the original image is not good enough, however, if the image is already good enough, there is not very much improve some-

times lead to degrade.

•2 : Although for the big background the method is good, for very narrow edge, and for the RGB image, since in the code to compare whether to perform weighted average, I first tranform the RGB to gray then do the comparison, which may lead to some colored sphere in the image.

•3 : If the original image is already good enough, deblocking may lead to over smooth and some sharp part may not sharp enough, so the deblocking may not be a good choice.

VI. Appendix

written problem

encoding the bit stream:0100110010

```
current state:12
A:0x e121          C:0x0    Qe:0x1edf       MPS:0
current state:11
A:0x f6f8          C:0x1210         Qe:0x2516       MPS:0
current state:11
A:0x d1e2          C:0x1210         Qe:0x2516       MPS:0
current state:11
A:0x accc          C:0x1210         Qe:0x2516       MPS:0
current state:10
A:0x 9458          C:0x6718         Qe:0x299a       MPS:0
current state:8
A:0x a668          C:0x4758         Qe:0x32b4       MPS:0
current state:9
A:0x e768          C:0x8eb0         Qe:0x2e17       MPS:0
current state:9
A:0x b951          C:0x8eb0         Qe:0x2e17       MPS:0
current state:8
A:0x b85c          C:0x67a8         Qe:0x32b4       MPS:0
current state:8
A:0x 85a8          C:0x67a8         Qe:0x32b4       MPS:0
```

transition table

```
0  59EB  0.49582  1  S
1  5522  0.46944  1  1
2  504F  0.44283  1  1
3  4B85  0.41643  1  1
4  4639  0.38722  1  1
5  415E  0.36044  1  1
6  3C3D  0.33216  1  1
7  375E  0.30530  1  1
8  32B4  0.27958  1  2
9  2E17  0.25415  1  1
10 299A  0.22940  1  2
11 2516  0.20450  1  1
12 1EDF  0.17023  1  1
13 1AA9  0.14701  1  2
14 174E  0.12851  1  1
15 1424  0.11106  1  2
16 119C  0.09710  1  1
17 0F6B  0.08502  1  2
18 0D51  0.07343  1  2
19 0BB6  0.06458  1  1
```

21	20	0A40	0.05652	1	2
22	21	0861	0.04620	1	2
23	22	0706	0.03873	1	2
24	23	05CD	0.03199	1	2
25	24	04DE	0.02684	1	1
26	25	040F	0.02238	1	2
27	26	0363	0.01867	1	2
28	27	02D4	0.01559	1	2
29	28	025C	0.01301	1	2
30	29	01F8	0.01086	1	2
31	30	01A4	0.00905	1	2
32	31	0160	0.00758	1	2
33	32	0125	0.00631	1	2
34	33	00F6	0.00530	1	2
35	34	00CB	0.00437	1	2
36	35	00AB	0.00368	1	1
37	36	008F	0.00308	1	2
38	37	0068	0.00224	1	2
39	38	004E	0.00168	1	2
40	39	003B	0.00127	1	2
41	40	002C	0.00095	1	2
42	41	001A	0.00056	1	3
43	42	000D	0.00028	1	2
44	43	0006	0.00013	1	2
45	44	0003	0.00006	1	2
46	45	0001	0.00002	0	1

VII. H.264 & MPEG2 Comparison

I. Motion Estimation

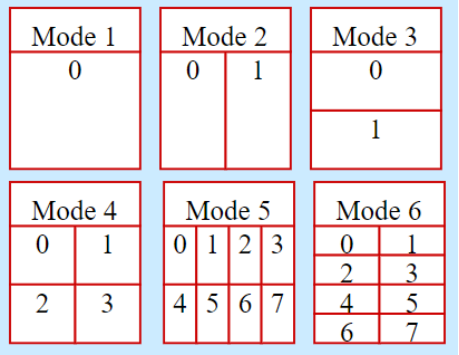


Figure 81: *Different Modes of Blocks*

•**Block sizes & Shapes** Compared to MPEG2, H.264 uses advanced motion estimation. First it uses different block sizes and shapes to do the motion vector estimation.So it supports more flexibility in the selection of motion compensation block sizes and shapes than any previous standard, with a minimum luma motion compensation block size as small as 4×4. [1] There are 4 different integer-pixel motion estimation methods provided by x264: diamond(DIA)[3], hexagon(HEX), uneven multihexagon(UMH), and successive elimination exhaustive search(SEA).[4].

• **High-precision Sub-pel Motion Vectors**Usually, the motion of blocks does not match exactly in the integer positions of samples grid. So to find good matches, fractional position accuracy is used. H.264 defined half-pel and quarter-pel accuracy for luma samples. When the best match is an integer position, just a 4×4 samples reference is needed to predict the current partition. However, if the best match is a fractional position, an interpolation is used to predict the current block. Such search method is described below:

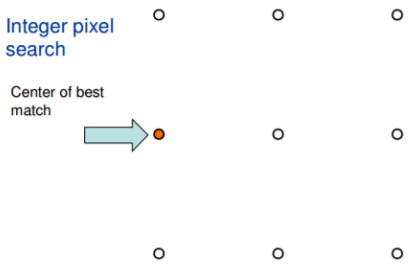


Figure 82: *Find Position Corresponding to Minimum Distortion*

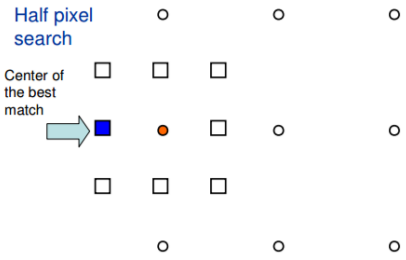


Figure 83: *Half Pixel Best Match*

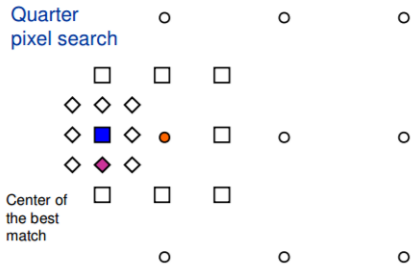


Figure 84: *Quarter Pixel Motion Estimation*

• **Multiple Reference Frames** Predictively coded pictures(the "P" pictures) in MPEG-2 and its predecessors used only one previous picture to predict the valuesin an incoming picture. H.264 enable efficient coding by allowing an encoder to select, for motion compensation purposes, among a larger number of pictures that have been decoded and stored in the decoder.

II. Intra Prediction and Coding

MPEG2 treats each macroblock independently, however H.264 studies the high spacial correlation between macroblocks. It also provides different Modes : Intra 4×4 luma blocks: 9 modes; Intra 16×16 luma blocks: 4 modes; Intra chroma blocks: 4 modes. [?] For a Intra 4×4 prediction, it has the following modes:

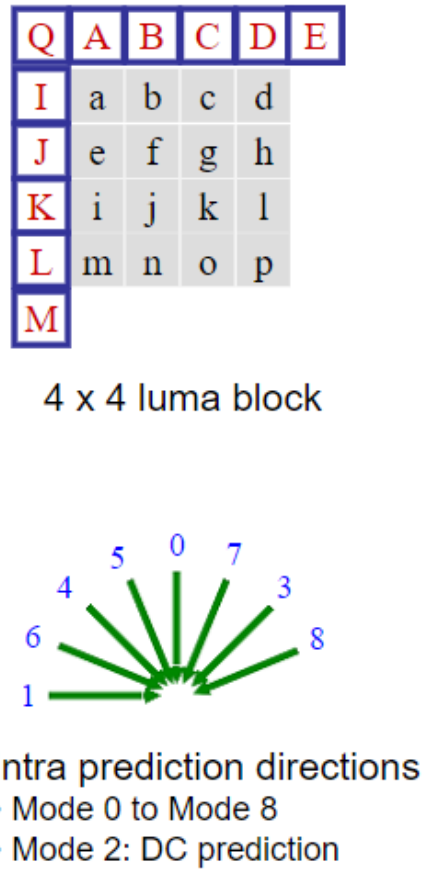


Figure 85: *Intra Predection Modes*

When using the Intra 4×4 mode, each 4×4 block is predicted from spatially neighboring samples.The process is illustrated below:

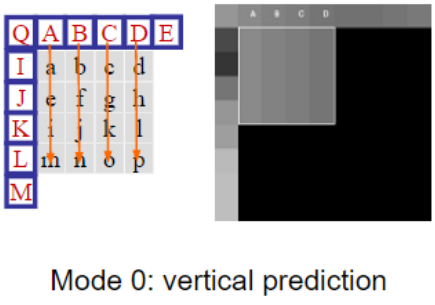


Figure 86: *Example of Mode 0*

The 16 samples of the 4×4 block which are labeled as a-p are predicted using prior decoded samples in adjacent blocks labeled as A-Q. For Mode 0, The samples above the 4×4 block are copied into the block as indicated by the arrows. Itra prediction across slice boundaries is not used, ino= order to keep all slices independent of each other.

III. Entropy Coding

In H.264 two methods of entropy coding are supported. CAVLC & CABAC [1]

• **Context-Adaptively switched Variable Length Codes**

In this scheme, VLC tables for various syntax elements are switched depending on already transmitted syntax elements. Since the VLC tables are designed to match the corresponding conditioned statistics, the entropy coding performance is improved in comparison to schemes using a single VLC table. In the CAVLC entropy coding method, the number of nonzero quantized coefficients and the actual size and position of the coefficients are coded separately. After zig-zag scanning of transform coefficients, their statistical distribution typically shows large values for the low frequency part decreasing to small values later in the scan for the high-frequency part. [1]In H.264, a total number of 32 different VLCs are used in CAVLC, CAVLC achieves 2-7% bit rate reduction compared to conventional runu length scheme.

•**Context-based Adaptive Binary Coding** This is discussed before, it uses adaptive probability models. Compared to CAVLC, CABAC typically provides a reduction in bit rate between 5% - 15%. The highest gains are typically obtained when coding interlaced TV signals.

VIII. Motion Estimation in x264

I. Estimation Methods

There are 4 different integer-pixel motion estimation methods provided by x264: diamond(DIA)[3], hexagon(HEX), uneven multihexagon(UMH), and successive elimination exhaustive search(SEA).

•**Diamond Search**: The diamond search has two patterns, the large diamond search pattern(LDSP) and the small diamond search pattern.In the searching procedure of this search, LDSP is repeatedly used until the step in which the minimum block distortion occur at the center point. The search pattern is then switched from LDSP to SDSP as reaching to the final search stage. Among the five checking points is SDSP, the position yielding the MBD provides the motion vector of the best matching block. Example :

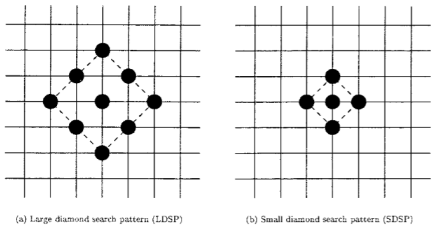


Figure 87: *Two Shapes of The Diamond Pattern*

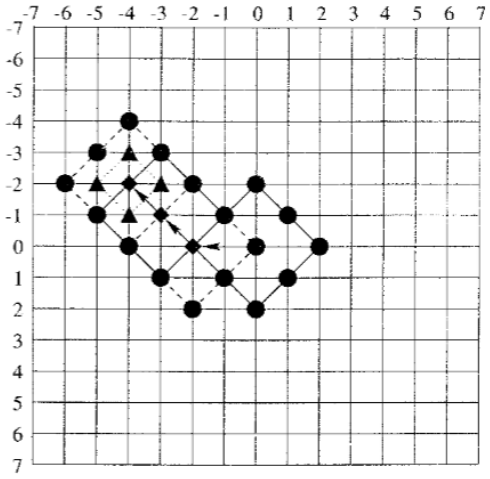


Figure 88: *Search path example which leads to the motion vector (-4,-2) in five search steps –four times of LDSP and one time SDSP at the final step. There are 24 search points in total –taking nine,five,three,three and four search points at each step*

•**Hexagon Search**: Like the diamond search the hex search also has two patterns, the large pattern is formed by pixels surrounding like the hexagon, and the small pattern is just like the small diamond as you can see in the illustration below. The search either goes to the boundary or samller search in the origin center. Also from the figure we can see that the number of check points to compare is decreased however, there can be some position where the estimation did not check and this may impact the performance

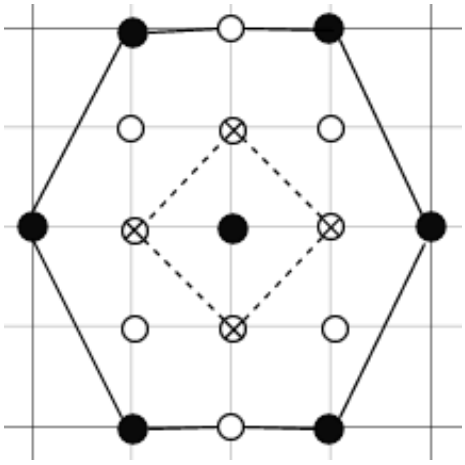


Figure 89: *The • and solid line form the position of boundary search, the ⊗ and the dash line marks the small pattern search, the ∘ marked the position that might not be used for estimation*

•**Uneven Multihexagon**: The basic form is still the hexagon search, however, to compensate the shortcoming that hexagon search may lose some position to search, uneven multihexangon search insert some checking points on the boundary which may be the estimated motion vetor direction, This is illustrated in the following figure:

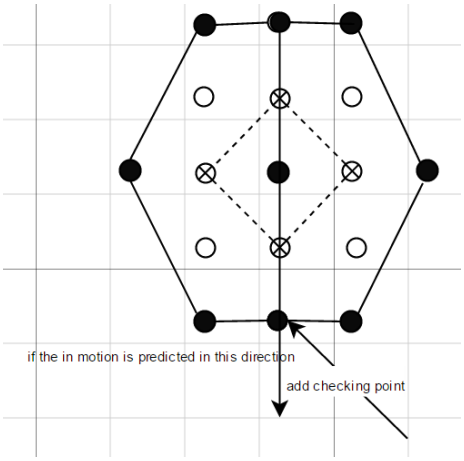


Figure 90: *Some checking points are added at the direction of the estimated motion vector direction*

•**Succesive Elimination Algorithm**: [4] The principle of this is basicly according to some mathematic deduction,

Let $f_c(i,j)$ and $f_r(i,j)$ denote the intensity of the pixel with coordinate(i,j) in the current frame and the reference frame respectively.Assume that the size of a block is $N \times N$ pixels, the motion vectorsearch window size is $(2M+1) \times (2M+1)$, and the matching criteria function is mean absolute difference (MAD). Let $B_c^{(i,j)}$ and $B_r^{(i,j,x,y)}(m,n)$ denote the two blocks in the two frames with the top left corners at (i,j) and $(i-x, j-y)$respectively, i.e.[4]

$B_c^{i,j}(m,n) = f_c(i+m, j+n)$

$B_r^{(i,j,x,y)}(m,n) = f_r(i-x+m, j-y+n)$ Suppose that the obtained optimal motion vector in theprevious search process is (x^*, y^*). At the current search point, the checked block is a better matching candidate only if $MAD(x,y) < MAD(x^*,y^*)$. We can see that the checked block is not a better matching candidate if $|R - M(x,y)| \geq MAD(x^*,y^*)$. Thus, the $MAD(x,y)$ should be calculated only if the following inequality holds:

$R - MAD(x^*,y^*) < M(x,y) < R + MAD(x^*,y^*)$

At each search position in the search window, we canrepeat this procedure to reduce the computation cost. The sum normscan be efficiently calculated on the whole image beforehand.[4]

II. ETRA for UMH

An early termination and range adaptive(ETRA) algorithm for UMH improves the speed.[2]. This method is complicated and uses various shapes of search pattern and modified initilization .

1.Modified Initialization: We consider up to 10 candidate motion vectors (MVs) that consist of four MVs from the neighboring locations in the current frame (A,B,C,D), three MVs in the previous frame (X,Y,Z), the (0,0) MV, the median MV and the temporal direct MV. These 10 MVs are used in the initialization step for all ME methods. Each block that is being encoded in the current frame is referred to as the reference index. We keep track ofthe MVs for each reference index at each position in space(referred to as motion field). The four neighboring MVs are taken from the motion field of the reference index that is currently being searched. The MVs from the previous frame are scaled based upon the amount of time they span. We comparethe sum of absolute differences (SAD) of each candidate, and choose the best SAD.

2. One step of the diamond search is applied to (0,0) and the median MV.

3. If the block size is 4x4, break from UMH and apply the hexagon search. Else, continue.

4. Apply one step of diamond search to the best MV.

5.Early termination: If step (4) did not find a new best MV, then perform the following early termination:
a. Perform diamond search of radius two.
b. If steps (1) or (2) found a new MV, run a symmetric cross search (radius 7) and an octagon search (radius two).
c. If steps (5a) and (5b) did notfind a new MV, then break.
6. Adaptive radius: Select the search range for step (7),based on the magnitude of the best SAD so far, and on the smoothness of the motion field (i.e., the variance between the predictors used in step (1)). The default search range is 16.This may decrease as low as 12 if the SAD is small and the predictors are similar, and may increase as high as 24 if the SAD is large and the predictors differ greatly.
7. Run 5×5 exhaustive search, uneven cross, multi-hexagon-grid, and iterative hexagon refinement as in JM.

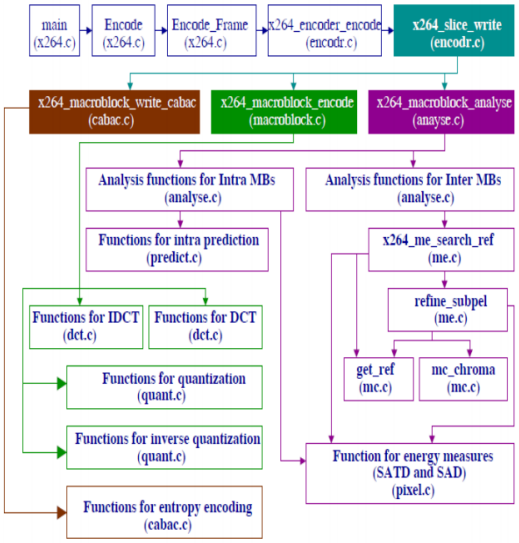


Figure 91: *hierarchy structure of x264 source code*

So from source code we see function $encode_frame$ calls $x264_encoder_encode$.

```
static int encode_frame( x264_t *h, hnd_t hout, x264_picture_t *pic, int64_t *last_dts )
{
    x264_picture_t pic_out;
    x264_nal_t *nal;
    int i_nal;
    int i_frame_size = 0;

    i_frame_size = x264_encoder_encode( h, &nal, &i_nal, pic, &pic_out );
```

Figure 92: *encode frame function*

continue we can easily find the place to calculate motion search time: we wrap $x264_mes earch$ with the clocks to acculate time the system spent on motion search time:

```
//modified here==============================//
clock_t time_begin;
time_begin = clock();
//==========================================//
x264_me_search( h, &m, lX->mvc[i_ref], i+1 );
//modified here==============================//
time_duration = clock()-time_begin + time_duration;
```

Figure 93: *place to put time*

There are 15 $x264_mes earch$ in the source , wrap them all with a clock and use a global variable $time_duration$ to get the motion search time.

III. Experiment Result

Experiment: modify the source code of x264 to compare different schemes of motion search. First the source code of x264 has a hierarchy structure in the following figure:

we can just comment the codes to trun off early termination.
for the initialize of predict motion vector,

```
/* save mv for predicting neighbors */
CP32( h->mb.mvr[0][i_ref][h->mb.i_mb_xy], m.mv );
CP32( a->l0.mvc[i_ref][0], m.mv );
```

Figure 94: *no extra motion estimator candidates*

To turn on or turn off early termintion, we can see in analyse.c

```
/* early termination
 * SSD threshold would probably be better than SATD */
if( i_ref == 0
    && a->b_try_skip
    && m.cost-m.cost_mv < 300*a->i_lambda
    && abs(m.mv[0]-h->mb.cache.pskip_mv[0])
     + abs(m.mv[1]-h->mb.cache.pskip_mv[1]) <= 1
    && x264_macroblock_probe_pskip( h ) )
{
    h->mb.i_type = P_SKIP;
    x264_analyse_update_cache( h, a );
    assert( h->mb.cache.pskip_mv[1] <= h->mb.mv_max_spel[1] || h->i_thread_frames == 1 );
    return;
}

m.cost += m.i_ref_cost;
i_halfpel_thresh += m.i_ref_cost;
```

Figure 95: *Early Termination*

We comment early termination or comment extra motion vector to get the following results:

Result:

Table 22: *Motion Search Scheme Comparision*

Scheme	time (clocks/5e6)	PSNR
DIA_x264	**7.9749**	**38.656**
HEX_x264	**8.9931**	**38.678**
UMH_x264	**15.403**	**38.685**
UMH_no_emv	**18.605**	**38.680**
ESA_x264	**26.487**	**38.687**
DIA_no_emv	**9.074**	**38.569**
HEX_no_emv	**11.478**	**38.673**
UMH_no_ETRA	**20.461**	**38.680**
UMH_no_both	**21.386**	**38.679**
ESA_no_emv	**29.970**	**38.691**

Comment We can see that if the source is 'foreman.yuv', the PSNR did not show much difference, but the time consume varies much. For example, the ESA took much time, this can be understood since ESA search all the position. We can also see that, the search scheme with extra motion candidates behave well both in speed and PSNR(although this is not very obvious in this case). Also we can see the time is increasing from DIA to ESA, since the position to search is increasing, like dia search 4 position at a time, and Hex search six then four and UMH search eight then four.

IX. Rate Control

Rate control is what a video encoder does when it decides how many bits to spend for a given frame. As we all know different frame has different size after compression.The goal of (lossy) video encoding is to save as many as bits as possible, reducing the file size over the original input file, while retaining as much quality as possible. Rate control allows selectoin of encoding parameters to maximize quality under the bitrate constraint and the decoder buffer constraint. Rate Control in H.264 can be performed at three different granularities - group of pictures level, picture level, and macroblocks level. At each level, the rate control algorithm selects the quantization parameter(QP) values that determine the quantization of the transformed coefficients. As the QP increases, the quantization of the transformed coefficients. H.264 includes five different modes, one two-pass and four one-pass modes.

I. Video buffer verifier compliant constant bitrate (CBR)

I.1 Video Buffer Verifier(VBV)

The Video Buffering Verifier (VBV) is a theoretical MPEG video buffer model, used to ensure that an encoded video stream can be correctly buffered, and played back at the decoder device. By definition, the VBV shall not overflow nor underflow when its input is a compliant stream, (except in the case of low_delay). It is therefore important when encoding such a stream that it comply with the VBV requirements.One way to think of the VBV is to consider both a maximum bitrate and a maximum buffer size. You'll need to know how quickly the video data is coming into the buffer. Keep in mind that video data is always changing the bitrate so there is no constant number to note how fast the data is arriving. The larger question is how long before the buffer overflows. A larger buffer size simply means that the decoder will tolerate high bitrates for longer periods of time, but no buffer is infinite, so eventually even a large buffer will overflow.

I.2 CBR

CBR is a one-pass mode designed for real-time streaming. CBR means the number of bits fed to the decoder is constant overtime. In other words, the **data transfer rate to the decoder is constant**. It has nothing to do with the number of bits of individual frames. The buffer verifier is here to check the buffer never overflow or underflow. The advantage of CBR is that it keeps a constant bitrate, so it is good for video streaming. Disadvantage, it may waste the bandwidth which is provided for transfering.
steps:
•Run a fast motion estimation algorithm over a half−resolution version of each frame, use the Sum of Absolute Hadamard Transformed Differneces of the residuals as the complexity.
•With the knowing size of the buffer, the scaling factor is computed with the desired bitrate and the local average complexity.
•After encoding each frame, the future QPs are updated to account for the misprediction in size, this is referred to as long-term compensation. Also, there is short-term compensation factor $C^{real\,filesize-predicted\,filesize}$ and C is defined by user.

```
Command:
ffmpeg -i <input> -c:v libx264 -x264-
    params "nal-hrd=cbr:force-cfr=1" -b:v
    1M -minrate 1M -maxrate 1M -bufsize 2M
     <output>
```

II. Average Bitrate (ABR)

ABR is a one-pass scheme which produces near-constant quality within a specified file size. steps:
•Run a fast motion estimation algorithm over a half−resolution version of each frame, use the Sum of Absolute Hadamard Transformed Differneces of the residuals as the complexity. (Same as CBR)
•The scaling factor is set to the one that would have resulted in the desired bitrate if is had been applied to all the frames so far.
•After encoding each frame, the future QPs are updated to account for the misprediction in size, this is referred to as long-term compensation. Also, there is short-term compensation

factor $C^{real\,filesize - predicted\,filesize}$ and C is defined by user.

```
Command:
ffmpeg -i <input> -c:v libx264 -b:v 1M <output>
```

The advantage is this method is only the speed, but bad for image quality.

III. Constant Rate Factor mode (CRF)

The Constant Rate Factor (CRF) is the default quality (and rate control) setting for the x264 encoders. You can set the values between 0 and 51, where lower values would result in better quality, at the expense of higher file sizes. Higher values mean more compression, but at some point you will notice the quality degradation. For x264, same values are between 18 and 28. The default is 23, so you can use this as a starting point.CRF is a âĂIJconstant qualityâĂİ encoding mode, as opposed to constant bitrate (CBR). Typically you would achieve constant quality by compressing every frame of the same type the same amount, that is, throwing away the same (relative) amount of information. It will compress different frames by different amounts, thus varying the QP as necessary to maintain a certain level of perceived quality. It does this by taking motion into account. anything lower will probably just waste file size. Values of Âś6 will result in about half or twice the original bitrate. The only downside with this mode is that you donâĂŹt know what the resulting file size will be.

```
Command:
ffmpeg -i <input> -c:v libx264 -crf 23 <output>
```

The advantage is that such method is good for archive and may achieve the best possible quality. The disadvange is that it is not suitable for streaming and obtain a certain bitrate.

IV. Constant Quantilizer mode (CQP)

In tech terminology, you maintain a constant QP (quantization parameter). The quantization parameter defines how much information to discard from a given block of pixels (a Macroblock). This typically leads to a hugely varying bitrate over the entire sequence. The QPs in CQP mode simply based on whether the frame is I,P or B-frame. This mode is used only when the rate control option is disabled.

V. Two Pass

It is easy to see from the name, we pass the source video to the encode channel twice, so that it is possible to estimate what's ahead in time. It can calculate the cost of encoding a frame in the first pass and then, in the second pass, more efficiently use the bits available. For the 2-pass method in paper [2], the 2-pass implemented as follows:

•1. The relative number of bits to be allocated between P-frames is selected independently of the total number of bitsand it is calculated empirically by $bits \propto complexity^{0.6}$,where complexity is the predicted bit size of a given frame at constant QP. The I- and B-frames use the QP from nearby P-frames with a constant offset.

•2. The results of step (1) are scaled to fill the requested file size.

•3. After encoding each frame, the future QPs are updated to account for the mispredictions in size (this is referred to as long-term compensation). If the second pass is consistently off from the predicted size, then the target size of all future frames is multiplied by a factor,and their QPs are recomputed.

VI. Experiment Result

We test each rate control schemes with the Elephants Dream sequence at ta target bitrate of 800 kb/s. The sequence is 640×480 pixels resolution, 25 frames per second and total 15691 frames.

Table 23: *Comparision between differnet rate control method*

Mode	PSNR(dB)	bitrate(kb/s)
CBR	**40.705**	**799.75**
ABR	**41.983**	**782.78**
CRF(22)	**42.529**	**705.17**
CQP(25)	**42.735**	**779.36**
Two pass	**43.225**	**798.35**

Comment From the result, we can see that the 2-pass achieved both PSNR and desired bitrate, so it make great use of the bandwidth. This is actually achieved by knowing the future information, therefore, this method is not good for quick and realtime encoding. Also we can see that if we use CQP with QP be 25, we can also achieve good quality, but actually we need to try QPs to fit the desired bitrate, and with the certain QP a certain bitrate can not be assured since the video can be complicated. CRF has the same problem with CQP, constant rate factor also can not give certain bitrate. CBR and ABR provides not so good quality but CBR keeps a constant bitrate. ABR is quick but not safe, it just jive guess to parameters and depends on the buffer to keep safe, so it can be bad for quality, for a constant bitrate, and bad for efficient use of a bandwidth.

For CQP we do more test based on different choice of QPs.

Table 24: *Different choice of QP*

QP	PSNR	bitrate
2	60.115	10342.5
4	57.412	8413.88
6	55.380	6609.20
8	53.894	5230.16
10	52.622	4175.86
12	51.350	3382.76
14	50.053	2730.67
16	48.728	2206.32
18	47.416	1777.68
20	46.057	1409.89
22	44.753	1116.39
23	44.646	989.65
24	43.377	877.07
25	42.735	779.36
26	42.012	684.30
28	40.702	531.89
30	39.377	412.44
32	38.087	318.98
34	36.870	245.53
36	35.694	190.03
38	34.526	147.10
40	33.405	115.53
42	32.328	90.63
44	31.231	71.81
46	30.183	57.32
48	29.168	45.66
50	27.950	36.50
51	27.626	33.05

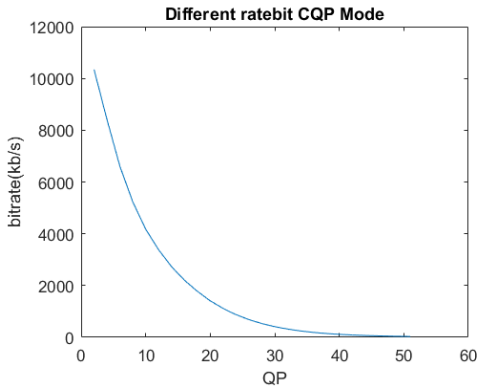


Figure 96: *How QP Affect Bitrate*

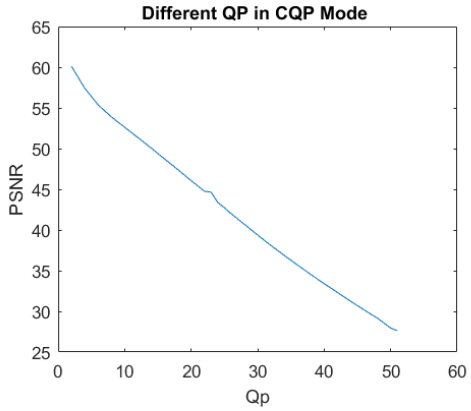


Figure 97: *How QP Affect Bitrate*

References

[1] Thomas Wiegand, Gary J.Sullivan, Gisle B. and Ajay Luthra *"Overview of the H.264/AVC Video Coding Standard"*. IEEE Trans. on Circuits and System For Video Technology, vol. 13, No. 7, pp. 560-576, 2003

[2] Loren Merritt and Rahul Vanam *"Improved Rate Control And Motion Estimation For H.264 Encoder"*. IEEE Trans. on ICIP, V-309-312, 2007

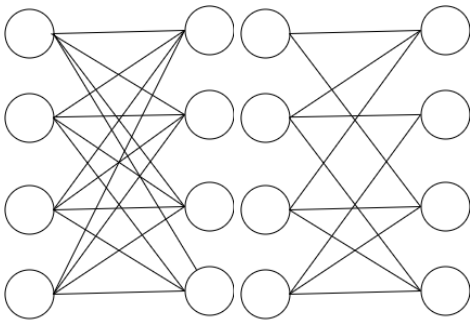
[3] Shan Zhu and Kai-Kuang Ma *"A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation"*. IEEE Trans. on Image Proc., vol 9, no. 2,pp.287-290, 2000

[4] X.Gao, C.J.Duanmu, and C.R.Zou, *"A multilevel successive elimination algorithm for block matching motion estimation"* IEEE Trans. on Image Proc., vol. 9, no. 3,pp.501-504, 2000.

X. CNN Training and Its Application to the MINST Dataset

I. CNN architecture understanding

Fully connected Layer Fully connected layers connect ecery neuron in one layer to every neuron in another layer, Example below show this concept:



(a) *fully connected* **(b)** *not fully connected*

Figure 98: *Example of fully connected layer*

Convolutional layer Convolutional layer apply 'convoltuion' operations with different filters. The network tunes the parameters of these filters. We actually perform the inner product operation when some filter overlay some image block. Belows shows an example:

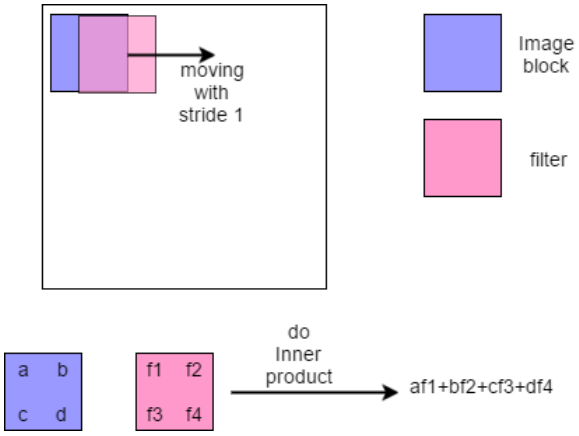


Figure 99: *Convolutional Layer*

another example:

$$Imageblock = \begin{bmatrix} 123 & 234 & 25 \\ 59 & 89 & 163 \\ 23 & 26 & 254 \end{bmatrix}$$

$$filter = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix} \quad result = \begin{bmatrix} 147 & 57 \\ 56 & 95 \end{bmatrix}$$

max pooling layer
The max pooling layer outputs the max element in each block.The following figure shows the process:

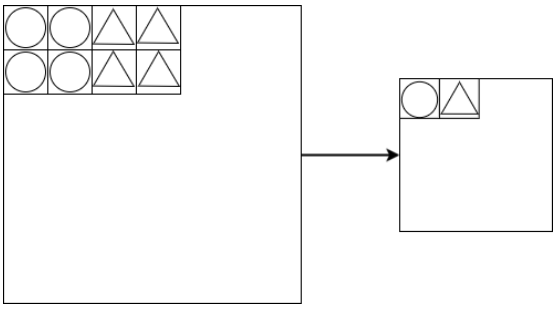


Figure 100: *Max polling layer*

The activation function First is the structure of one single neuron:

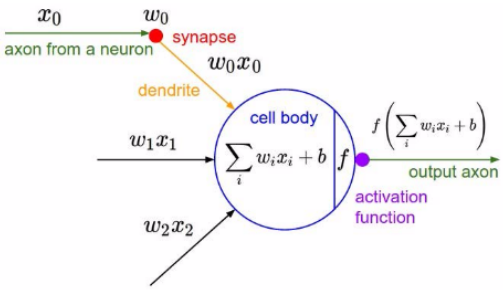


Figure 101: *The structure of one neuron*

The activation function deals with the weighted sum from the synapse. There are many different types of the activation function. The most frequently used activation functions are $tanh$,$ReLU(RectifiedLinearUnit)$,$Logistic$ etc.

the softmax function

the softmax function, or normalized exponential function, is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range [0, 1] that add up to 1. The function is given by $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} for j = 1,,K$. Here for the classification project we usually use softmax function to represent how much similar the object is belong to the class.

II. CNN

Why CNN

Instead of traditional feature extraction kernels, CNN uses the network to keep tring to find the best suitable feature extration kernel. In the image classification project, beacuse CNN did well in feature extraction, therefore it works much better than traditional methods.

Over-fitting

Over-fitting happens when the trained network fit the training network so well while leading to the inefficiency to the testing data. For example, a model may learn the detail and the noise in the training data to the extent that it negatively impacts the performance of the model on new data. To overcome over-fitting, we can reduced the number fo filters, or assign a reasonable ratio of the number of training image to number of the testing image. Also, we set a parameter of dropout to drop some element in the network while traing. For exmaple in tensorflow we use arguments like keep_prob to denote the probability that each element is kept.

III. Experiments

III.1 Initialization

We use the Lenet-5 as the basic structure, therefore the first convolutional layer we use 6 filters, I choose the following 6 filters as the initial filter:(0:black, 1:white)

Figure 102: *6 initial filters*

For the next 16 filters, we randomly generate numbers and assign them as the filter weights. For the learning rate, the optimizer I chose is the GradientOptimizer, and set the learning rate 0.001. For the cost funtion we use the cross-entropy and for training we set the probability of keep the neuron to be 0.5. Also, I set the batch size to be 60 and run 2000 epoch

III.2 Result

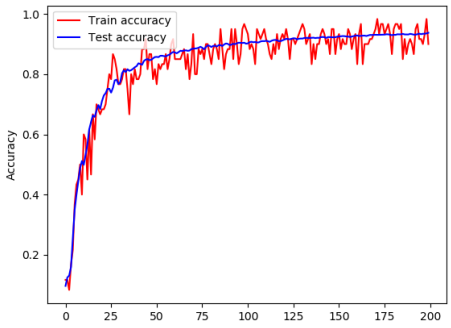


Figure 103: *LeNet-5 Network, platform Tensorflow*

Observations: under this setting we can find the training accuracy curve fluctuates more intensively. Sometimes the test accuracy is over training accuracy since the training image set and the testing image set are different. Moreover, these two curves both tend to converge. **following experiments tune the learning rate and keep_prob to get better test accuracy**

Table 25: *epoch 2000, keep_prob 0.5*

learning rate	test accuracy
0.0001	0.9367
0.001	0.9826
0.01	0.9791
0.1	0.101

Table 26: *epoch 2000, learning rate 0.001*

keep_prob	test accuracy
0.2	0.9778
0.4	0.9818
0.6	0.9843
0.8	0.9864

under this setting:

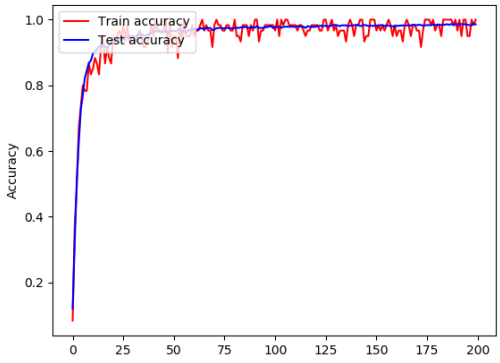


Figure 104: *learning rate:0.001,keep_prob:0.8,epoch 2000, batch size 60, Lenet-5 CNN setting*

Under this setting, the performance tends

to converge more qulickly and the flutuation tends to be less intense.

IV. Improve the LeNet-5

Still like the LeNet-5, three convolutional layers are used, four experiments are conducted, we change the filter numbers in the convolutional layers, we still choose 3 convolution layers. Other settings we use the parameters found in the previous step.

Table 27: *dropout_rate: 0.2, learning rate: 0.001, optimizer: AdamOptimizer epoch 2000*

filter number			accuracy(test data)
c1	c2	c3	
16	32	512	0.9907
32	64	1024	0.9903
48	96	1536	0.9857
64	128	2048	0.9857

Comment More convolution layers may not lead to a better result, maybe because of the overfitting problem. And we can find that the accuracy goes to 99.07% in 2000 epoch. and the first setting gives the best result.
Set all the 'optimal' parameters we can find so far, I ran 5000 epoches and the best accuracy we can get is **99.21%**.

XI. Saak Transform and its application to the MNIST dataset

I. Saak Tranform

Unlike the CNN which uses lots of supervised data to train the filters and find the image feature, Saak(Subspace approximation with augmented kernels) transform just performs forward operations. The basic forward Saak transform can be illustrated in the following image:

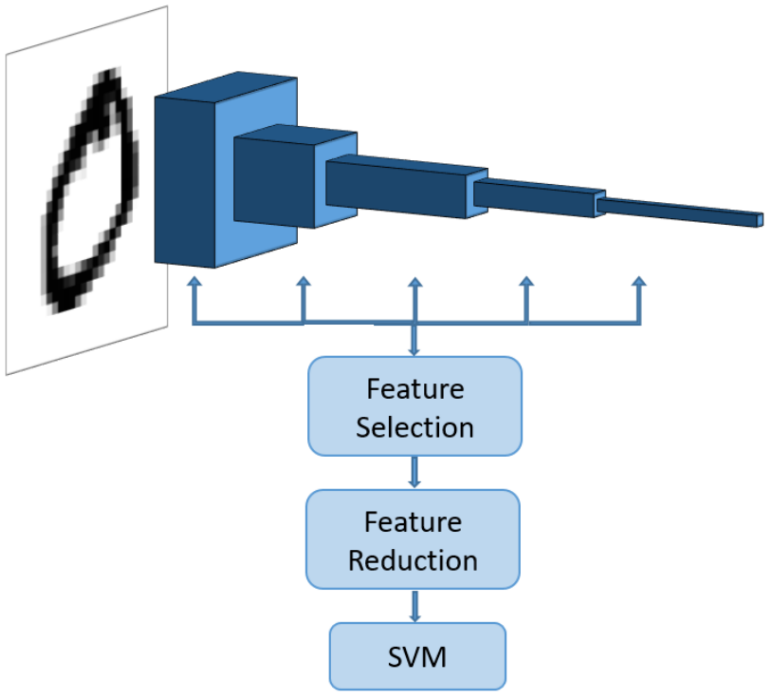


Figure 105: *Illustration of forward Saak tarnsform*

•The first step is the Karhunen-Loeve tranformation. That is for a set of datas with some dimension, we can reduce the dimension by finding subspace that can represent these datas. In image case, we take the first step as an example. We first split the image into none overlap 2×2 blocks, if we have an image of size 32×32 than we have 256 points in a space with four dimensions. We can perform the KLT transform and find the 4 directions(basic functions) in the hyperspace. After getting the four basic functions , we can get the 4 responses with one dc responce.

•Then is the sign to position operation, this is an easy operation, for example if we have a layer with KLT responses: first we add a layer (call it back layer) after this layer. For each element in the layer, if the element is a response of apositive number say 5, than we keep this in the front layer and add a zero to the back layer. If the element is a response of a negative number say -3, than we put 0 in the front layer and put 3 in the back layer.

• After the first Saak tranform stage, we have a cuboid, with 7 channels, we still split the 16×16 with 2×2 masks, now we have a haperspace with dimensions $4 \times 7 = 28$. Then we keep doing the KLT. However it is easy to see that the dimension of the space would become larger and larger. So we perform the truncated KLT also known as PCA. We keep dimensions that we are interested in.

II. SVM

S(**S**upport)V(**V**ector)M(**M**achine) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. The basic SVM is a non-probabilistic binary classifier. Given a set of training examples, each marked as belongings to one or the other of two categories, an SVM training algorithm builds a model that can assign new example to one category or the other.
In this case obviously we need an SVM with 10 categories, so we need an SVM for multi-class classification. The dominant approach for doing so is to reduce the single multiclass problem into multiple binary classification problems.Common methods for such reduction include:
Building binary classifiers which distinguish (i) between one of the labels and the rest (one-versus-all) or (ii) between every pair of classes (one-versus-one). Classification of new instances for the one-versus-all case is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class (it is important that the output functions be calibrated to produce comparable scores). For the one-versus-one approach, classification is done by a max-wins voting strategy, in which every classifier assigns the instance to one of the two classes, then the vote for the assigned class is increased by one vote, and finally the class with the most votes determines the instance classification.

III. Experiments

I designed the following steps to conduct experiemnts:
•1 take 1000 samples from the training data set, perform forward saak transform
•2 perform F-test on the result feature data (1000 images each with feature data vector of dimension 1999) keep the α% information, and got feature data vector with size $2000 \times \alpha$.
•3 we got the feature data with shape(1000,1500) , add the feature data to feature data set,also add label information to the label data set
•4 repeat step **1-3** for N times,(N>=60), collect data from training dataset to have a feature data set with shape(N*1000,1500)(In my experiment I set N to be 80 because I turned the Shuffle option on , and I am trying to training the SVM with all the images in train data)
•5 Perform another round of PCA to the (N*1000,1500) feature data set matrix and reduce the dimension to D (I choose 64,128,192), now feature data set has shape(N*1000,D)
•6 Train the SVM with the feature data set.Settings: ($one - against - oneclassifier$)
•7repeat step**1-6**, but instead of getting data from training data set, we get it from test data set, and set N to be 12 since the test dataset has a smaller size.
•8 put the Saak response of the test images to the classifier we training, compare the estimated label to the actually label.

IV. Result

I choose different α(the ratio of important information size to the total feature size) , and different D(64,128,192). The accuracy of the test data shows as follows:

Table 28: *Test images accuracy*

	last PCA dimension		
Dimension after F-test	64	128	192
2000	93.82	93.07	93.22
1500	93.66	93.29	93.56
1000	93.92	94.18	94.09
750	93.65	93.72	93.88

comment: Compared to CNN Obvious the result is not better than the CNN, but almost the same performance regarding to the LeNet-5 setting to the MNIST dataset. But the most important feature of Saak Tranform is that it is an one-pass operation.This accuracy already did well. Robust test is not performed but with augmented kernel, Saak transform maybe more robust than CNN.

Why augmented kernels are neccesary in the multi-stage Saak transform

The Saak tranform is motivated by the RECOS(REctified-COrrelations on a Sphere) transform which has the approximation loss and the rectification loss. The estimation loss can be solved by using the KLT. And we need augmented kernel to solve the rectification loss brought by the ReLU. We augment each tranform kernel with its negative vector, and an input vector will project into the positive and negative kernel pair. For example, if we have two image with a black cat and a white cat each has the same feature with our human eye, if using ReLU just train the white cat, may be the rectifictaion loss would lead to can not recognize the black cat.

Potential improvement

•1In this Saak tranform, one image is decomposed into four overlapping quadrants recursively to build a quad-tree whose leaf node is a small patch of size 2×2 and maybe we can also try 3×3 or some rectangular shape say4×2 or 2×4.

•2 according to the experiemnts results, maybe there also exist overfitting problem, so choosing a suitable number of important components is important, it should not be too large or too small.

•3 I took the wrong prediction sample and found that even humanbeing can make a wrong guess, it is a picture looks like 6 but labeled 0 and predicted as 6, so maybe some better dataset can improve the training.

XII. Image Understanding and Compression

First, we can reconstruct an image with a lossy Saak transform by choosing the number of important Saak coefficients, therefore we can set a parameter which works like the quality factor in the JPEG. If we want good reconstruction quality, we set the number to be high which means choose as many as coefficients as possible to achieve lossless.
Secondly, about assigning more bits on RoI(Region of Interest), actually we can analyze what those low frequency patches(like the background) becomes after the Saak Tranform. Clearly all the patches with the same feature that have low frequency becomes points on a certain line in the hyperspace. Because their response to the basic function would be the same direction just different magnitude, we can extract those data from some direction in the hyperplane. Such points magnitude took from the specific direction in the hyperplane can represent the background data, actually this works more like the RLC(Run Length Coding). If we look into the hyperplane we can always find some directions we are intersted in and these directions are actually the feature of an image. Therefore assign more bits on some direction data, is what makes the Saak Tranform perform compression job efficiently.
To sum up, my pipeline should work like this: For each stage of the Saak transform, after get the response layers(each layer is actully one subspace), I assign some importance to some subspace, with the supported vector(derived from PCA), if I want background, I took the subspace with more background information by assign a 'Q-matrix' with lower quantization step. Also we can do the same thing to the object information we want. Therefore, a choice of 'Q-matrix' to each stage each subspace not only can extract the information we want and achieve the compression work.

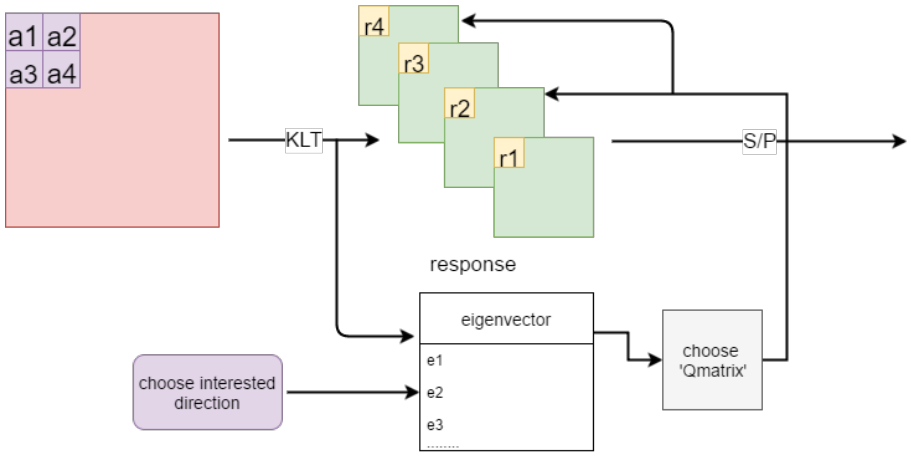


Figure 106: *My thought in image understanding and compression along the Saak transform pipeline*

XIII. Basic Image Manipulation(Problem 1)

I. Color Space Transformation(Problem 1(a))

I.1 Abstract and Motivation

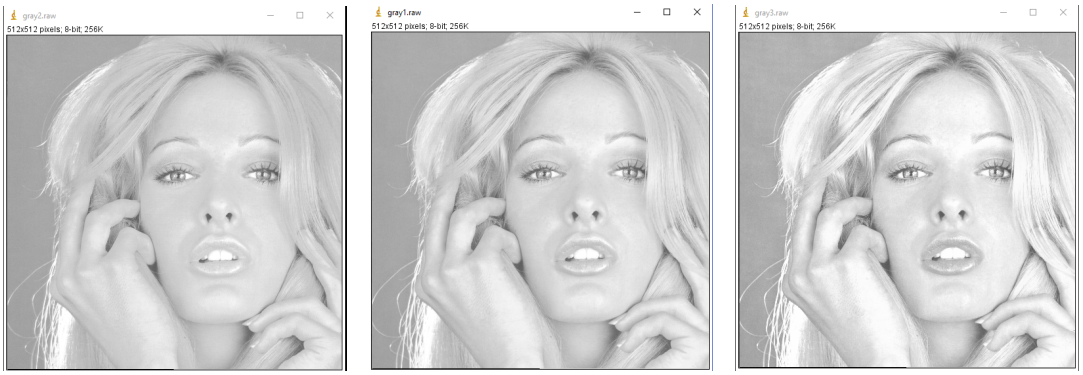
A color space is a specific organization of colors. In combining different color representation, it allows for reproducible representations of color, in both analog and digital representations like how we paint. A color space may be arbitrary, with particular colors assigned to a set of physical colors,what our eyes can perspect, and a color space can be defined as numerical number like the 24-bit true color, or what we used in industry and high-technique areas. The most commonly used color model for study is the grayscale, which is suitable for feature detection, the RGB model which is suitable for color image editing, and the CMYK model for the priting job.

I.2 Color-to-Grayscale Conversion

There are many ways to convert a color image to its grayscal image.
lightness method:$f(p) = (max(R(p), G(p), B(p)) + min(R(p), G(p), B(p)))/2,\ p(position)$
average method:$f(p) = (R(p) + G(p) + B(p))/3,\ p(postition)$
luminosity method:$f(p) = 0.21 \times R(p) + 0.72 \times G(p) + 0.07 \times B(p),\ p(position)$
Experiment result: **Result**

platform:Visual Studio 2017, Debug and Release under x64,language C++

Perform the three methods on the T́iffany.rawf́ile:



(a) *lightness* **(b)** *average* **(c)** *luminosity*

Figure 107: *Three RGB to Grayscale method*

Discussion Maybe the three images look the same at a galnce but we can still see some difference if we look at the lips,hands and the face face lightness. Clearly as the name implies , the lightnessmethod makes the images looks bright, the lips are shinning and the hair is almost the same

bright color.Moreover the face is more smooth than the other two but the face looks gloomy.Then for the luminosity method, we can see more contrast and details in the face. We can not only see the dark part but also some bright part as well. It feels more realistic than the lightness one.And for the average method, the effect is just between the lightness and the luminosity one. We can see bright part in face also the dark part and face is more smooth than the luminosity one. I personally prefer the luminosity one since more contrast may provide more details.

I.3 CMY(K)Color Space(Problem 1(b))

The Cyan-Magenta-Yellow-(Black)(CMY(K)) color space is frequently used in image printing. It is defined by: $\begin{cases} C = 1 - R \\ M = 1 - G \\ Y = 1 - B \end{cases}$

Experiments

Perform the transform on the 'Bear.raw' and 'Dance.raw' files.

steps:

•1Extract R,G,B components seperately

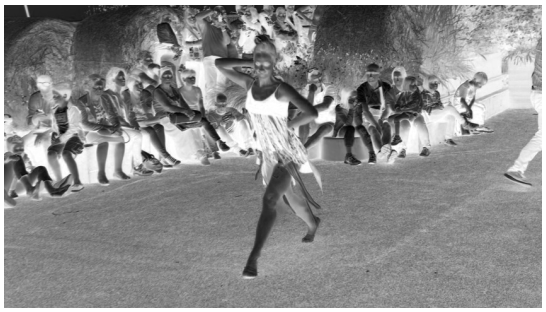
•2For each channel,inverse the intensity.

Result

platform:Visual Studio 2017, Debug and Release under x64,language C++

(a) *Cyan*

(b) *Magenta*



(a) *Cyan*

(b) *Magenta*

(c) *Yellow*

Figure 109: *CMK Color Space (Dance.raw)*

I.4 Image Resizing(Problem 1(c))

Motivation

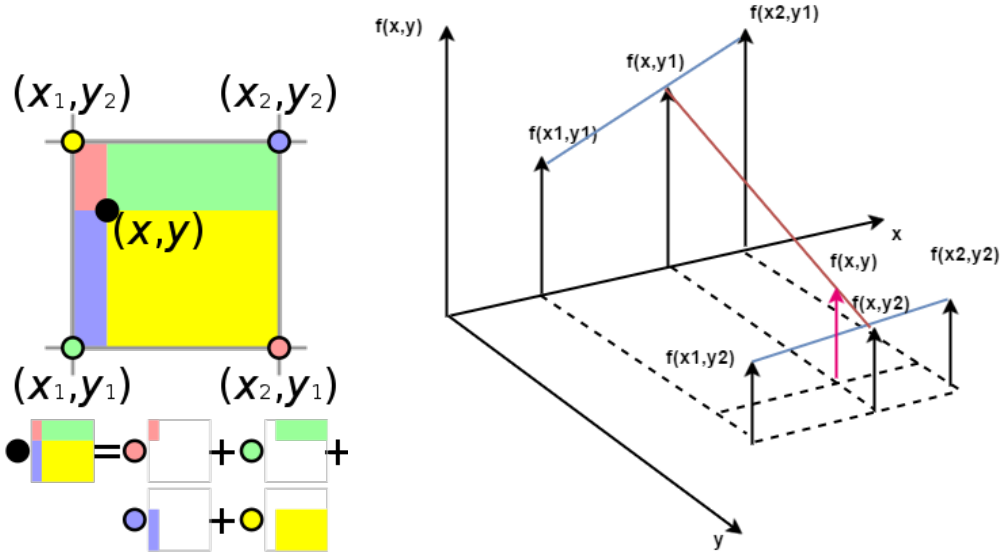
Most of time we need to zoom in an image to observe something small in the image, and sometimes we need to shrink an image when we want to arrange some images together. So it is important to study how to enlarge an image and how to shrink an image and this immediately inplies questions about how pixels are interpolated(for enlargement) or merged(for reduction).

To shrink an image, direct subsampling may lead to Gibbs effect, since we inject high frequency components into the image and most of the time, we smooth the image first and then do the subsampling.

Enlarge image is not easy because we need to add pixels which are not provided by the origin

image. There are several way to interpolate, one simple method is to find the pixel in the origin image that is closest to the pixel which is going to be interplate. This method is called the nearest neighbour. However we may have more methods and we are going to talk about bilinear interpolation.

Bilinear Interpolation



(a) *Find the intensity of (x,y)* **(b)** *bilinear interpolation*

Figure 110: *mathematical illustration for the bilinear interpolation*

Core Algorithm

```
1     for (int i = 0; i < newHeight - 1; i++) {
2       for (int j = 0; j < newWidth - 1; j++) {
3       //find the points for interpolation
4       int lux = floor(j*ratiow);  //the upper left corner coordinate x
5       int luy = floor(i*ratioh);  //the upper left corner coordinate y
6       double kx = j*ratiow - lux; //the slope for direction x ,ratiow is the scale ratio
            of width
7       double ky = i*ratioh - luy; //the slope for direction y ,ratioh is the scale ratio
            of height
8       //retrieve the data of four corner
9       unsigned char ulValue = data[luy*oldWidth + lux];
10      unsigned char urValue = data[luy*oldWidth + lux + 1];
11      unsigned char blValue = data[(luy + 1)*oldWidth + lux];
12      unsigned char brValue = data[(luy + 1)*oldWidth + lux + 1];
13      // linear calculation on direction x
14      double x1 = cal(ulValue, urValue, kx);
15      double x2 = cal(blValue, brValue, kx);
16      // linear calculation on direction y
```

```
17          double y = cal(x1, x2, ky);
18          dataResize[i*newWidth + j] = (unsigned char)y;
19      }
20  }
```

Result Result

platform:Visual Studio 2017, Debug and Release under x64,language C++

Figure 111: *Original Size(scale to 40% of original size to fit the paper)*

Figure 112: *Original Size(scale to 40% of original size to fit the paper)*

XIV. Histogram Equalization(Problem(2))

I. Backgroud and Motivation

Histogram equalization usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values. The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered. The calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal.

II. Method and Algorithm

II.1 Transfer-Function-Based Algorithm

This algorithm is kind of subjective because we can freely choose the transfer functions to transfer some intensity to another intensity. To illustrate such method, it is better to give two examples:

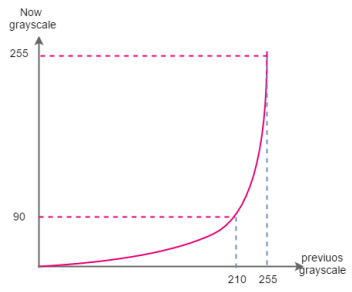


Figure 113: *One example of grayscale mapping*

first if an image consist of many bright part, for example , there are many intensity in range [210,255], we can build the transfer funciton as the left figure shows:
The figure shows that the intensity between 210 to 255 are mapped to a wider range, from 90 to 255, which means those close contrast values are now easily be observed. However we need to find suitable transfer functions for some specific image, because different images have different intensity distributions.

Algorithm
•study the histogram of the data
•select the transfer function subjectively
•map the data with the transfer function

Experiments and Results

Histogram

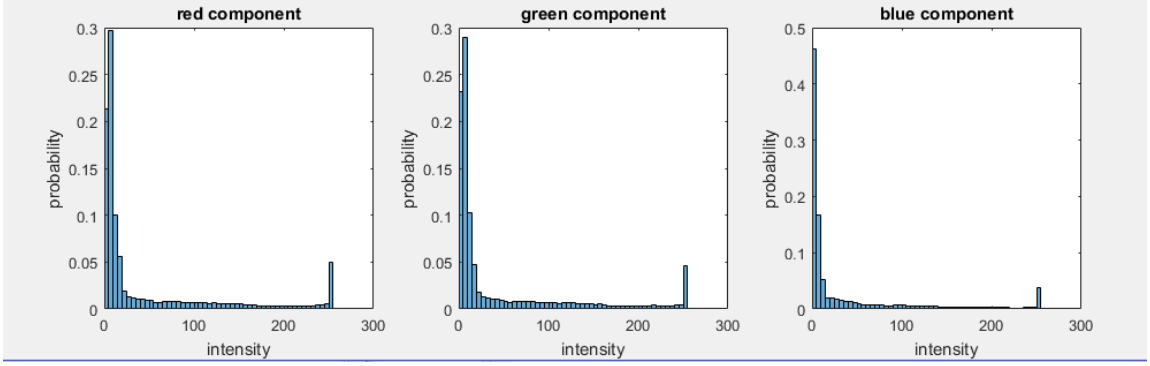


Figure 114: *Histogram of the Desk image*

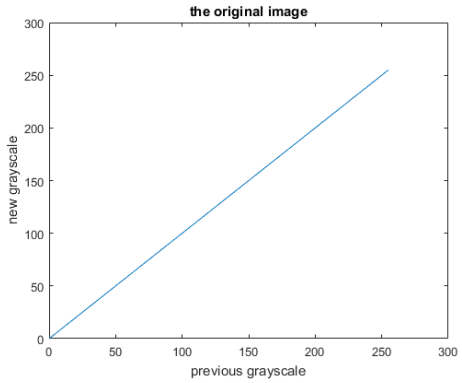
As we can see from the histogram, and the origin image, there are many pixels having the intensity from 0 to 50, therefore we need to choose transfer functions that can stretch this part.That is, we need to enlarge the range, using a line or curve which has a high slope at the begining. Two transfer functions are chosen, one is linear and the other one is non-linear. For the linear one, I decide to stretch [0-50] to [0,220], and compress [50 - 255] to [220-255] therefore the linear function I chose is:

$$h(v) = \begin{cases} \frac{50}{220}v & v \in [0, 50] \\ 220 + \frac{255-220}{255-50}v & v \in [50, 255] \end{cases}$$

and I use a log-like function for the nonlinear operation, the function is :
$h(v) = (log_{10}v)\frac{255}{log_{10}255}$

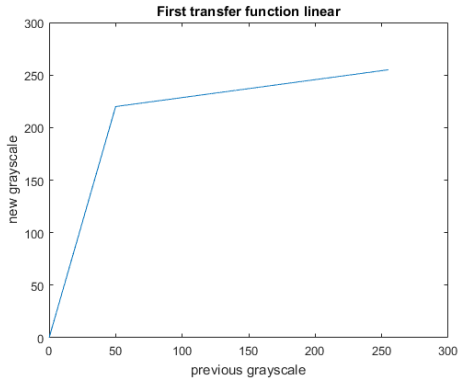
results are the following

platform:Visual Studio 2017, Debug and Release under x64,language C++



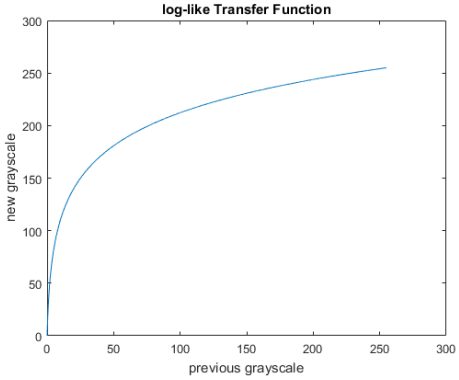
(a) *No change(plot with matlab)*

(b) *The original image*



(c) *The linear transfer function(plot with matlab)*

(d) *The image after transfer*



(e) *The nonlinear transfer function(plot with matlab)*

(f) *The image after transfer*

Figure 115: *Transfer function and the results*

II.2 The Cumulative-Probability-Based method

This is an objective method since we collect the data and try to find the best transfer function. As we all know the best result is to change the former distribution into a uniform distribution, and this can be done by manipulating the 'CDF'(Cumulative Distribution Function).

Algorithm

•Calculate the pdf of the intensity for each channel

•For each pdf, accumulate to get the cdf , $cdf(x) = cdf(x-1) + pdf(x)$

•Generate the mapping function $newIntensity(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$ where$M \times N$shoe the size of a one channel image and L is the number of grey level, here $L = 256$

Experiments and Results

The cdf of three channels are:

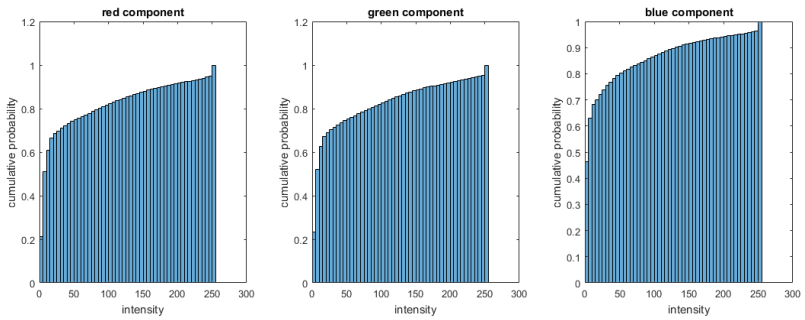
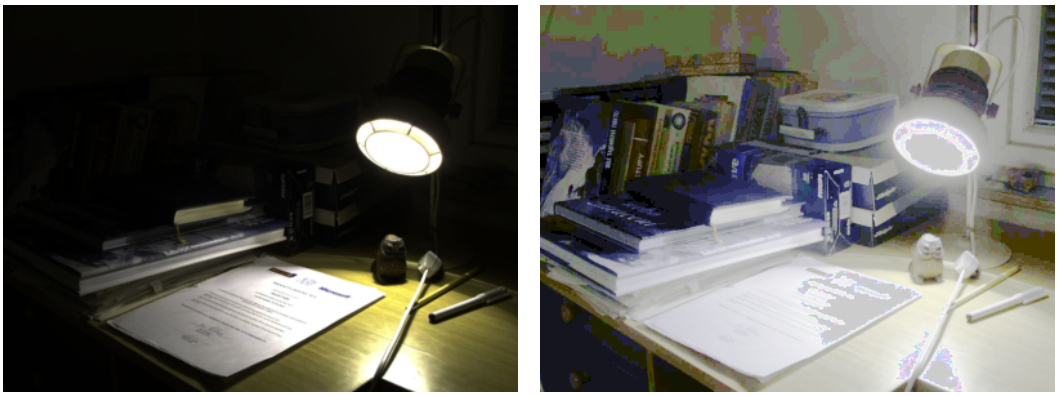


Figure 116: *cdf of intensity (plot with matlab)*

After histogram Equalization:



(a) *Original image* **(b)** *after histogram equalization*

Figure 117: *Histogram Equlization with cumulative-probability-based algorithm*

Discussion Compare the transfer function method and the cumulative method, we can find that the transfer function equalized images looks more smooth, because all the three channels we use the same functions. For the cumulative one, because three channels has different cdfs, we can see some distortion or 'color jump' in the images,makes the images not so realistic. Look at the

images, still the cumulative method brights the background the most, we can even observe the color of objects in the background. There is also one thing we need to notice, the impact to paper in the image is intersting. Before histogram equalization, we can see obvious contrast on the paper, black words on white paper. However, after the operations to the histogram, we contrast almost disappears. This is not good, the details of the image has been sabotaged.Therefore, maybe we can try some other algorithms in order to not only show the objects in the dark but preserve the contrast of some objects as well. So one way is to identify the background, or the dark part of an image. One way is to block the images,need a large window, and look into the histogram of the blocks. If some block has almost the same intensity , we can do equalizaiton. However this method has two shortcomings, one is how to decide the window size, and the other is how to decide the equalization parameters. The other method is to segment the image and identify the dark part.Then use the globla histogram to only equalize the dark part the preserve the bright part.

XV. Oil Painting Effect(Problem(2b))

I. Color Quantization

I.1 Method and Background

Like the closely related **k-means** clustering algorithm, Lloyd's algorithm repeatedly finds the centroid of each set in the partition and then re-partitions the input according to which of these centroids is closest.

I.2 Implementation

Initiation method:

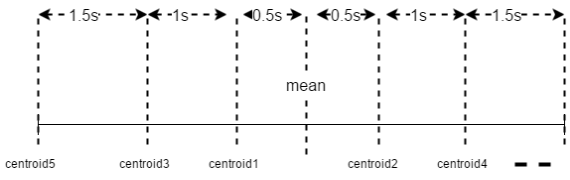


Figure 118: *Initilization of k-mean*

To initiate the centroids, I first calculate the mean and standard deviation of the datas of the given data, A class named lloydmax is created, which has public functions for users to call initilazation. The following figure show how I initilize the centroids.s is the standard deviation of all the training data.
Since if we were given a large amount of data, we may assume laws of large numbers, my assumption is all image may subject to some distribution, and we use the nonlinear standard deviation to get the initial centriods.
Update routine

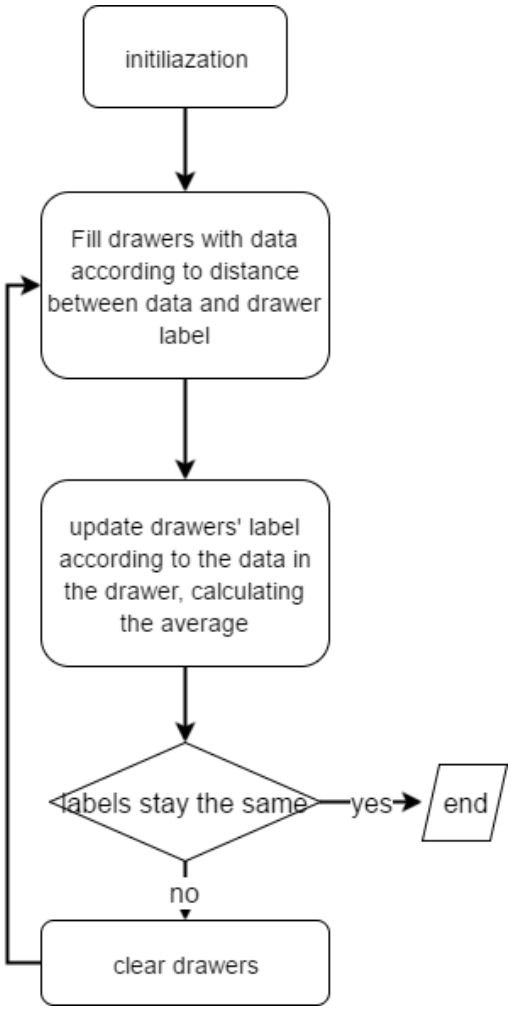


Figure 119: *update routine of k-mean*

In my code design, and in my head I create a desk with n drawers, n is the number of classification, each drawer has a label which is the centroid, and in the drawer are the data. the left shows the flowchart:

I.3 Experiment and Result

Take the 'Star_War.raw' for example, we perform the k mean algorithm and get the centroids:

platform:Visual Studio 2017, Debug and Release under x64,language C++

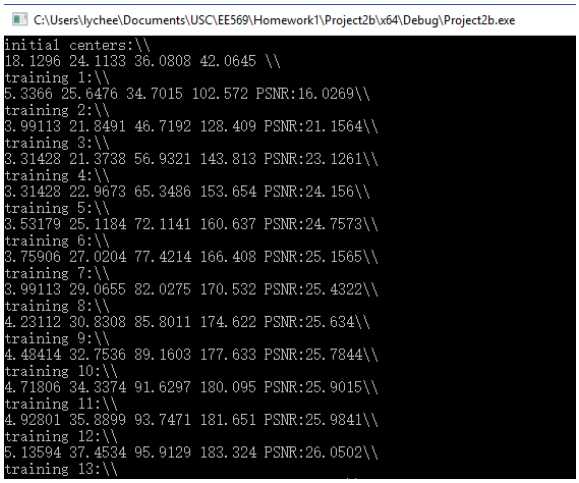


Figure 120: *training with Star_War image(choose 2 bit 4 color)*

After the training for each channel, we get the following representative intensity:
R: 6.47671 48.8416 112.551 197.249 PSNR:26.3835 (29 training)
G: 6.84428 46.9499 108.367 185.576 PSNR:25.6253 (24 training)
B: 8.74686 56.6232 120.926 202.265 PSNR:24.3243 (20 training)
Now we get the intervals for us to quantize the color.

(a) *Origin Image*

(b) *Color Quantization, 2-bit each channel*

Figure 121: *Quantization with 64 colors*

II. Grouping with Most Frequent Color

II.1 Implementation

Step1: Choose the block size, from 3 to 9, odd numbers
Step2: For each pixel in the image, select the most frequent color in its $N \times N$ neighborhood as the representative clor for the pixel

II.2 Result

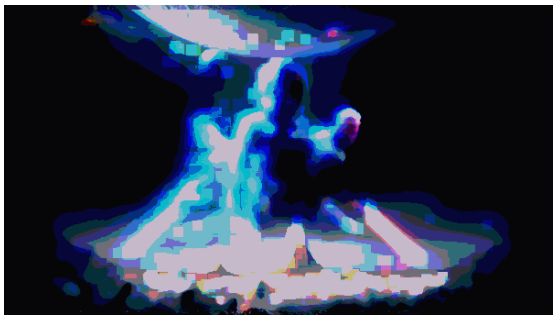
(a) *N = 3*

(b) *N = 5*

(c) *N=7*

Figure 122: *Grouping with N × N neighbours,N = 3,5,7*

(a) *N=9*



(b) *N = 11*

Figure 123: *Grouping with N × N neighbours,N = 9,11*

The next are results of image 'Trojans.raw'
Color Quantization Result

(a) *Origin image*

(b) *Color Quantization, 2-bit each channel*

Figure 124: *Quantization,64 Colors*

Grouping Results

(a) *N=3*

(b) *N = 5*

(c) *N=7*

Figure 125: *Grouping with N × N neighbours,N = 3,5,7*

(a) *N=9*

(b) *N = 11*

Figure 126: *Grouping with N × N neighbours,N = 9,11*

II.3 Discussion

clearly with N get larger the blocking effect is more and more obvious,

II.4 Requantization with 512 colors

Now each channel only has 3 bits to store the value of color, in total we can use 512 colors. Still we use k-means and the algorithm has already be written, so we change from 4 centroids to 8 centroids. Results for the final quantization step after training:

'Star_Wars'

R: 3.10445 19.3646 41.2774 68.8113 99.3464 130.918 165.307 205.431 PSNR:34.0704 (35 trainings)

G: 3.8476 24.3729 49.7842 78.4193 107.761 136.686 167.486 205.383 PSNR:33.4729 (26 trainings)

B: 5.4178 29.7428 56.3364 83.9135 112.424 142.123 174.159 206.258 PSNR:31.317 (33 trainings)

'Trojans.raw'

R: 20.4927 58.4051 91.8219 120.518 147.884 177.941 205.825 243.588 PSNR:28.9588 (23 trainings)

G: 11.521 44.7873 85.9311 113.058 143.939 177.498 206.024 243.464 PSNR:28.9369 (27 trainings)

B: 13.3346 46.8866 90.7046 117.031 146.69 178.803 205.134 244.687 PSNR:29.0593 (28 trainings)

(a) *The Origin Image*

(b) *Color Requantization, 512 colors*

Figure 127: *Requantization, 'Star_Wars'*

(a) *The Origin Image*

(b) *Color Requantization, 512 colors*

Figure 128: *Requantization, 'Trojans'*

Grouping Color results:

'Star_War'

(a) *Requantization with 512 colors*

(b) *N=3*

(c) *N=5*

Figure 129: *N = 1,3,5*

(a) *N = 7*

(b) *N = 9*

(c) *N = 11*

Figure 130: *N = 7,9,11*

Grouping Color results:

'Trojans'

(a) *Requantization with 512 colors*

(b) *N=3*

(c) *N=5*

Figure 131: *N = 1,3,5*

(a) *N = 7*

(b) *N = 9*

(c) *N = 11*

Figure 132: *N = 7,9,11*

II.5 Discussion

First let us compare images that use 64 colors instead of 2^24colors. Actually if we performed a very good training like what I did using the k-means, we can not see much difference if the images are shrinked as the star war image shows. But when we zoom in, we can clearly see the blocking effect of the image which is quantized. If we are allowed to use 512 colors, differences are smaller. Such method may used for color image compression however which is not very practical because each image do not has the same codebook and the training image' training result may not suitable for the testing image.
Secondly, let us talk about the grouping pixels in a block with the most frequent color. We can see clear blocking effect when the N(window size) goes larger . For the star war picture, I like N being 3 and for the trojan image, I like N being 7. Look at image trojan after quantize with 512 colors and set N =7, the sword is just white color but can still be seen different from the background and the background is some kind of painting skill using large brush, very artistic and still realistic.

III. Film Special Effect(Problem 2(c))

III.1 Motivation

Need tho design an algorithm to achieve teh film special effect, an example is shown below:

(a) *Original Image* **(b)** *Film effect*

Figure 133: *Film Special Effect example*

III.2 Method

As we can see from the example, there are two steps for the film effect. First is the mirroring operation, then is the color space transform.
•1 :For mirroring operation, for each pixels in each row, we can use stack to store the data one by

one and then pop the data out to achieve the mirroring effect.
•2 :For Color Space transformation, I use the color inversion.

III.3 Result

(a) *Original Image* **(b)** *Film Effect*

Figure 134: *Film effect of the Girl image*

XVI. Noise Removal(Problem 3)

I. Motivation and Background

All recording devices, both analog and digital, have traits that make them susceptible to noise. Noise can be random or white noise with no coherence, or coherent noise introduced by the device's mechanism or processing algorithms.
There are different noise types, like the white noise whose statistics shows that all noise samples are wide sense stationary with mean zero. And we have impulse noises which results from sensor-saturated or sensor-dead. Therefore we have some white and black dots in the images. This is also called the 'pepper and salt' noise.

II. Methods to remove noise

II.1 Linear smoothing filters

:This method remove noise by convolving(actually this is inner-product with a sliding window) the original image with a mask that represents a low-pass filter or smoothing operation. For the low-pass filter, we can choose the average filter or the Gaussian-mask filter.

Example:

average filter: $M = \frac{1}{W \times H} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & 1 & \cdots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$

Gaussian filter 2D:

where x and y is the relative distance from the pixel's neighborhood to the pixel.

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\frac{(x-\mu_X)^2}{\sigma_X^2} + \frac{(y-\mu_Y)^2}{\sigma_Y^2} - \frac{2\rho(x-\mu_X)(y-\mu_Y)}{\sigma_X\sigma_Y}\right]\right)$$

In our case, we let x and y become independent and they have no bias. the algorithm becomes:

$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y} \exp(-[\frac{x^2}{2\sigma_X^2} + \frac{y^2}{2\sigma_Y^2}])$

mask looks like $(5 \times 5, \sigma = 5) : M = \frac{1}{S} \begin{bmatrix} 0.0054 & 0.0058 & 0.0059 & 0.0058 & 0.0054 \\ 0.0058 & 0.0061 & 0.0062 & 0.0061 & 0.0058 \\ 0.0059 & 0.0062 & 0.0064 & 0.0062 & 0.0059 \\ 0.0058 & 0.0061 & 0.0062 & 0.0061 & 0.0058 \\ 0.0054 & 0.0058 & 0.0059 & 0.0058 & 0.0054 \end{bmatrix}$ S is the sum of elements in the matrix

II.2 Nonlinear filter

In this paper we provide the median filter,as the name implies, for each block(mask),we took the median as the representative intensity of that pixel.

steps:

•1.consider each pixel in the image

•2.sort the neighbouring pixels into order based upon their intensities.

•3.take the median value from the list as the representative value.

II.3 Non-local means

Another approach for removing noise is based on non-local averaging of all the pixels in an image. In particular, the amount of weighting for a pixel is based on the degree of similarity between a

small patch centered on that pixel and the small patch centered on the pixel being de-noised.
Steps:
•1.For each block, calculate all the similarities between other blocks and this block.
•2.add all the blocks togethers with the corresponding weight, and average the sum.

III. Promblems and Experiments

III.1 Problems

(1)Do all channels have the same noise type?
Compare the mixed noise image and the original image, we get the histogram of errors in each channel and get the following result:

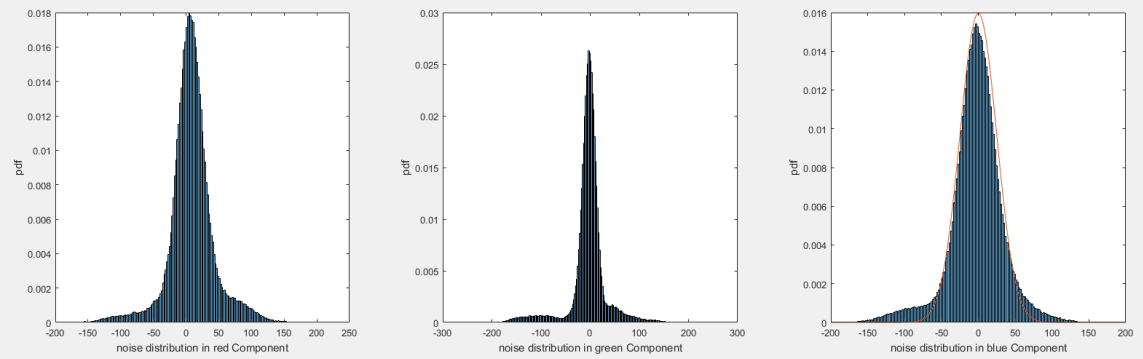


Figure 135: *The noise histogram*

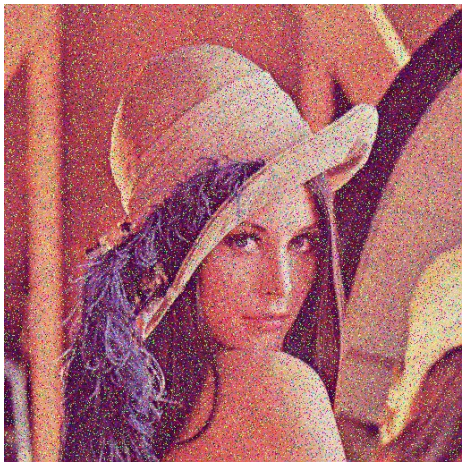
The histogram implies that the red channel and the blue channel have the same noise type :**Gaussian noise with sigma 25**, and the green channel has **Impulse noise**
(2)Perform filtering on individual channels?
Yes, because different channel has different noise type, I would perform filtering on individual channels seperately.
(3)What filters to choose to remove mixed noise?
the best noise reduction technique for red and green channel is the NLM(Non local mean) because the noise type is gaussian noise with mean 0. For green channel, it is best to use median filter to remove the salt and pepper noise.
(4)cascade filters in any order?
Yes, we can cascade filters, for example in this experiment I would use gaussian filter with small block size to filter the image first to prevent oversmooth but also smooth the image and then use the median filter with small block size because somehow median filter remain some sharp part of an image. The result will show below

(5)Discuss the effect of different window size

See the following figures and PSNRs:

III.2 results

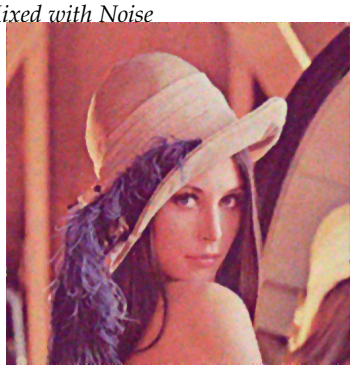
(a) *Origin Image*



(b) *Mixed with Noise*

(a) *Average Filter*

(b) *Gaussian Filter*

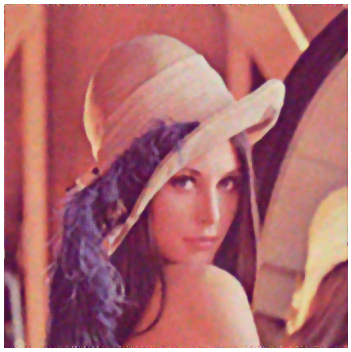


(c) *Median Filter*

Figure 137: *Window size 5×5*

(a) *Average Filter*

(b) *Gaussian Filter*



(c) *Median Filter*

Figure 138: *Window size 7×7*

(a) *Average Filter* **(b)** *Gaussian Filter* **(c)** *Median Filter*

Figure 139: *Window size 9×9*

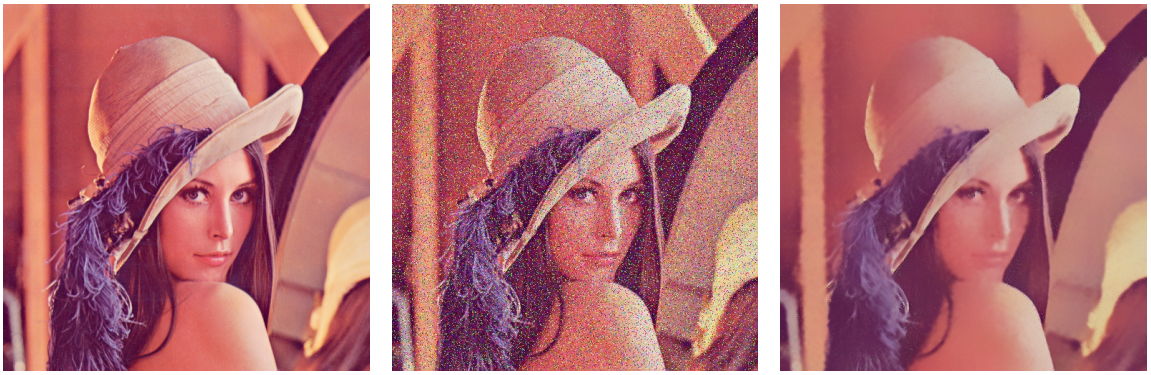
Table 29: *Performance of The filters*

window size	PSNR(dB)		
	average filter	gaussian filter(sigma = 15)	median filter
5	24.1378	24.1442	25.8006
7	23.9013	23.9109	25.641
9	23.42	23.4399	25.1805

From the figures and the corresponding PSNRs we can see that the larger window size provides more smoothed images. But the PSNRs becomes smaller since the images are over-smoothed.

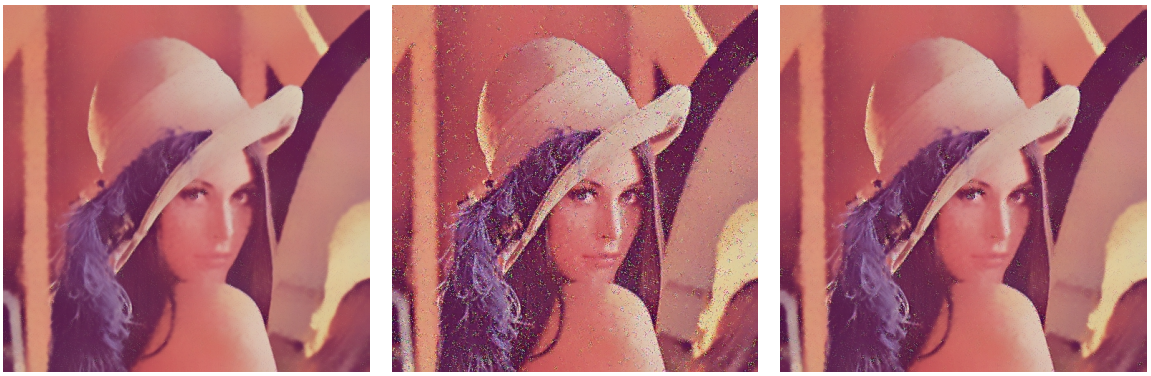
III.3 Best result in moving noise

(1)The methods and the results For the method, we also tried non-local mean, the algorithm has described above.The following figures show the results:



(a) *Original image* **(b)** *Mixed with noise* **(c)** *Denoise with NLM*

Figure 140: *Non-local Mean, Window size 7*



(a) *Big Filter Strength* **(b)** *small filter strength* **(c)** *median filter strength*

Figure 141: *More Non-local-mean experiments*

(a) *cascade filters, gaussian window 3 , median window 3* **(b)** *cascade filters, gaussian window 5, median window 5* **(c)** *cascade filters, gaussian window 5,median window 3*

Figure 142: *experiments with cascading filters*

the final cascading filters did not perform better than just one median filter whose PSNR is 25.4582, therefore , the best one is still the median filter with window size 5 whose PSNR is 25.8006

(2)ShortComings

These filters just use a low pass filter , linear or non linear in spatial to reduce noise, the image can be over smoothed.

(3)advises to improve the performance

Instead of just performing filter operation on the spatial domain, we should look into the data of the image or image blocks and find the data-driven algorithm to reduce noise just like the PLPCA and BM3D did.

IV. PLPCA

IV.1 PCA Introduction

PCA, principle component analysis is a procedure to convert a set of observations into a set of linearly uncorrelated variables called principal components.Why PCA can be used as a filtering approach for noisy data? Data has itself some covariance, and we can find pattern from it, by performing PCA ,we can find which dimension(direction) affects the data most, such direction is the eigen-vector of the covariance matrix, is the components. If we choose those components and disregard the others, the true signal can be effectively estimated. For the noisy data, perform PCA can help us find the main components in the data, and therefore, smooth the data.

Algorithm (data as row,column as feature)

$x_i = (f_1, f_2, \cdots, f_m)^t$

$X = (x_1^t, x_2^t, x_3^t, x_4^t, \cdots, x_n^t)^t \qquad X \in R^{n \times m}$

•1 Centralize by minus the mean,$X_0 = X - \overline{X}$

•2 Get the covariance matrix,$Cov = X_0^t X_0$

•3 Get the eigenvectors and eigenvalues of the covariance matrix, for instance if the data has two features, the covariance is an 2×2 matrix which usually has two eigenvectors, these two eigen-vectors is the direction for the main components.

•4 Sort the eigenvalues in descending order, the eigenvectors corresponding to the large eigenvalue are the main components.

•5 Set the threshold how much components are to kept.

•6$(optional)$ Project the original data with the selected eigenvectors

•7$(optional)$ Backproject the projected datas, the origin data would be modified by having eliminated some not-so-important component.

IV.2 PLPCA Algorithm

Algorithm

•1 Select the window size for the patch image, and select the range of neighbours in local.(sliding window size)

•2 Stack the images together, data as row, each row is a patch data

•3 Perform PCA to the data matrix, select the variance to preserve and backproject to original data.

IV.3 Experiments

Parameters

1.The parameters to tune are the window size, the size of image patch. Here I choose from 5 to 9.

2.The sliding window size, where to find the neighbour image patch. Here I choose from 9 to 21.
3.The retained variance, here I choose 0.2 or 0.6.
result

platform:Visual Studio 2017, Debug and Release under x64,language C++

Table 30: *Experiment Result(ws:window size, sws:sliding window size,rv:retained variance)*

exp	ws	sws	rv	PSNR
1	5	11	0.2	27.6722
2	5	11	0.6	25.336
3	5	13	0.2	25.9232
4	5	13	0.6	24.1632
5	5	15	0.2	27.7353
6	5	15	0.6	25.2205
7	5	17	0.2	26.4558
8	5	17	0.6	24.293
9	5	21	0.2	26.8716
10	5	21	0.6	24.4498
11	7	9	0.2	26.2333
12	7	9	0.6	25.2881
13	7	11	0.2	26.4422
14	7	11	0.6	24.8427
15	7	13	0.2	24.3169

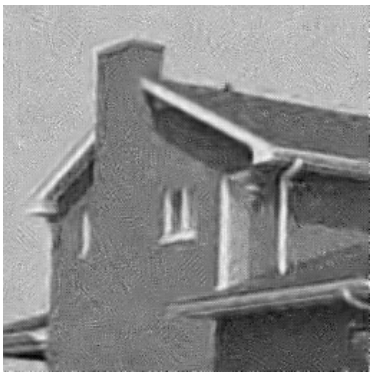
Table 31: *Experiment Result(ws:window size, sws:sliding window size,rv:retained variance)*

exp	ws	sws	rv	PSNR
16	7	13	0.6	23.446
17	7	15	0.2	26.6688
18	7	15	0.6	24.8347
19	7	17	0.2	26.3258
20	7	17	0.6	24.4883
21	7	21	0.2	26.5699
22	9	9	0.6	23.2218
23	9	11	0.2	26.0114
24	9	11	0.6	24.9287
25	9	13	0.2	25.5212
26	9	13	0.6	24.2199
27	9	15	0.2	25.5346
28	9	15	0.6	24.2139
29	9	17	0.2	23.7479
30	9	17	0.6	22.9598

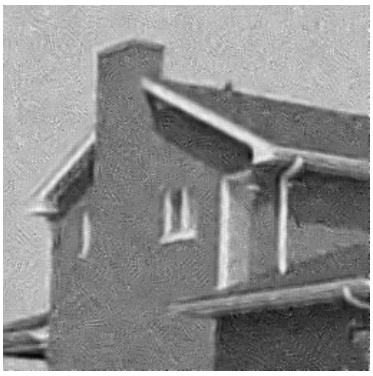
Accordint to the PSNR, the best four are: **window size 5,sliding window size 11, retained variance 0.2**
window size 5,sliding window size 15, retained variance 0.2
window size 5,sliding window size 21, retained variance 0.2
window size 7,sliding window size 15, retained variance 0.2

(a) *Origin Image*

(b) *Noisy Image*

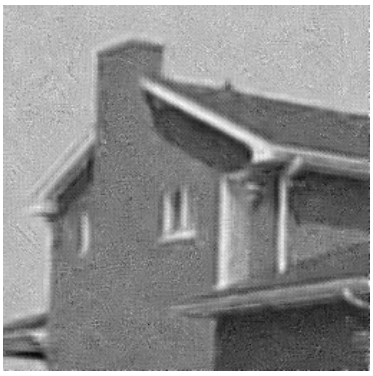


(c) *Best One*



(a) *Second Best*

(b) *Third Best*



(c) *Fourth Best*

4:Apply part(a)'s approach, the results as the following shows:

(a) *Average filter 5, PSNR:26.713*

(b) *gaussian ,sigma 10,PSNR:26.7394*

(c) *window size 5, PSNR:27.2776*

Figure 145: *Performance using filters*

Clearly from the performances we can find that the PLPCA looks better than the spatial domain filters. The house image mainly has a low frequency signal, and has some repeated patterns, the spatial filter can not find those patterns however the PLPCA can find the feature , and keep some useful signals. PCA is data-driven, from the algorithm of PCA, we find covariance from the data. Since each pixels in some position in a block stack can be seen as a random vector, and from what we know about random vector, according to the KL expansion, they can be expressed as a

weighted sum of orthogonal random variables and corresponding eigenvectors, and therefore we can doing linear estimation and affine estimation. That is better than just the experimental spatial filter.

V. BM3D transform filter

V.1 Abstraction

As far as I am concerned this is a method combining the thoughts of non-local-mean and PLPCA. The flowchart of the proposed image denoising algorithm shows as follows:

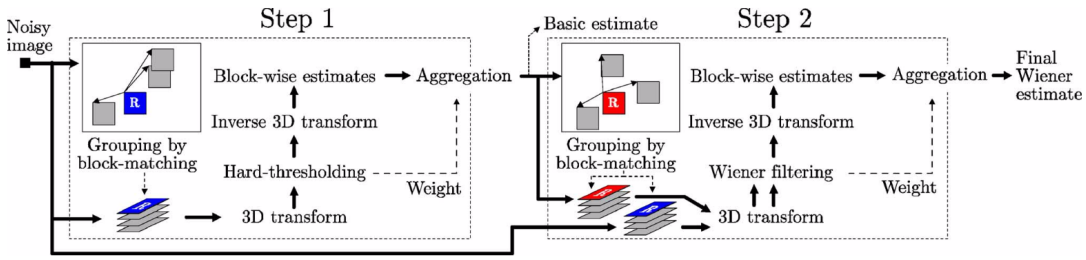


Figure 146: *flowchart of the proposed image denoising algorithm*

The first operation is image grouping, like the PLPCA and non-local-mean, we group block images. The method for grouping the images is group matching. The grouping method used in this algorithm is to find reference blocks first and then find the blocks that match the reference block. Then for the grouped data, we now have data of dimension 3 therefore we can perform 3-D transformation, there are a lot of transform basis we can choose . And the paper gives different performance of the transform basis's performance on different images. After this transform we can set the thresholding to keeping the main information in the image. For example ,if the basis is DCT or DFT, we set those threshold using similar matrix like the Q matrix used in JPEG. Then we perform the inverse 3D transform, and get a new 3D data. Then for this stack of images , we can perform block-wise estimate, that is estimate the true signal with all blocks in the images. Also we perform global aggregation. And with these two we can get a basic estimate of the true signal. Input this signal to a whole new system just like the previous one and we can form a Wiener filter.

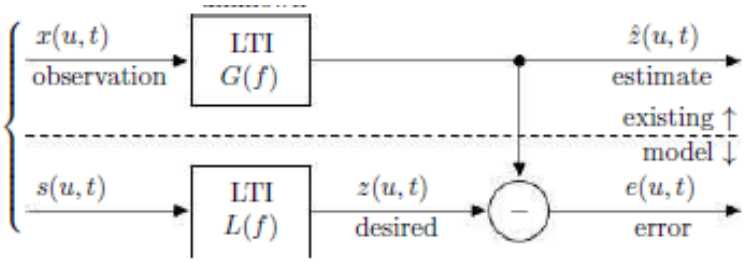


Figure 147: *Basic Structure of A Wiener Filter(EE 562 Robert Scholtz Course Note)*

And using the noisy image as one input and the desired(basic) as another input, we can get our system of this wiener filter.

V.2 Experiments and Result

Table 32: *Performance of different transform basis and block size (search window size: 39)*

	Transform Method					
		hadamard	**bior1.5**	**dct**	**DCrand**	**dst**
block size	**8**	**31.90**	**32.60**	**32.65**	**32.62**	**32.61**
	16	**32.54**	**32.55**	**32.54**	**32.56**	**32.29**
	32	**32.21**	**32.22**	**32.20**	**32.25**	**32.03**

Table 33: *Performance of window size and neighborhood search size(Transform : DCT)*

		Neighborhood Searching window size			
		39	**45**	**51**	**57**
block size	**8**	**31.90**	**32.64**	**32.66**	**32.66**
	16	**32.54**	**32.58**	**32.59**	**32.59**
	32	**32.20**	**32.21**	**32.22**	**32.23**

(a) *Origin Image* **(b)** *Noise image* **(c)** *BM3D method*

Figure 148: *Performance of a BM3D Method, block size 8, search window size 51, 'dct'*

V.3 Discussion

1. The effect of parameters First study the first table. For fixed block size 8, the best transform is DCT, however for the fixed block size 16 , the best transform method to choose is bior1.5. So for a fixed block size the best transform method can be different. But we can still see that when the performance goes down when the block size gets larger.
For the second table, we can see that the performance gets better when the neighborhood searching window size become larger, however it seems that there exists a convergency, you can find that searching window size 51 almost has the same performance as the searching window size 57.
2. Why using block matching instead of k-mean If we would like to use k-mean in this method, we have to first set the number of class, however we do not know the number of class we should choose to make sure we did a good block match. Instead if we use block matching, we select the reference block and for each reference block we can put similar blocks into that class, and if some blocks are different, just start another class and automatically we get those blocks clustered.
3.How would you classify BM3D denoising algorithm Clearly in the pipeline, there are spatial domain filter and frequency domain filter, the blocking and agrgregation are spatial domain filter and the hard-thresholding after the 3D transform is clearly the frequency domain filter.

Compare between PLPCA and BM3D
Clearly see the PSNR performance, the best one with PLPCA is just goes to 27.80, and the BM3D almost reconstructed the image. The better part of BM3D is good block matching which is not implemented in my PLPCA, in PLPCA I just use all the neighbors to form the data stack and perform operation.Moreover, although both methods use statistical analyse, the wiener estimation surely provides good result,since more it use the previous processed image and best estimation to estimate true signal, sometimes this has a function to sharpen a processed image.

XVII. Geometrix Image Modification

I. Geometrical Warping(1.a)

I.1 Motivation

We can apply some spatial warping technique to transform images into some special effects. In this problem we need to transform the input square image into an output image of a disk-shaped image.

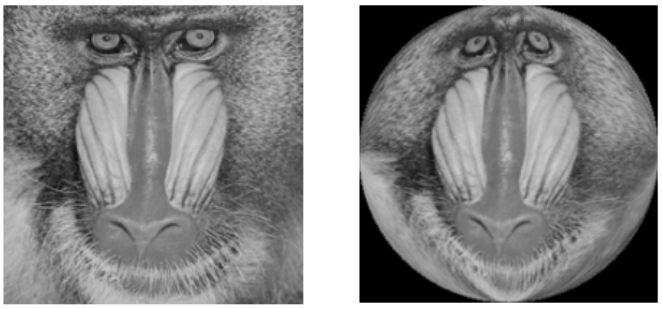
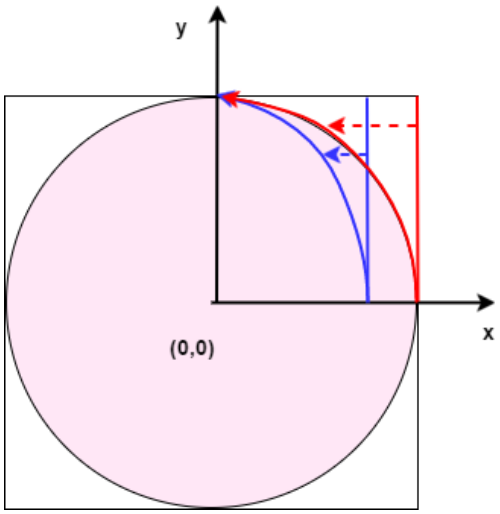


Figure 149: *Exmaple:warp to a disk-shaped image*

The wrapped image technique in this problem should satisfy three requirements: •Pixels that lie on the boundaries should still lie on the boundaries of the square should still lie on the boundaries of the circle.
•The center of the original images should be mapped to the center of warped images.
•The mapping should be reversible.

I.2 Routine

Analyze the warped image and the origin image. We can find the transfer function, denote the coordinate after transfer be (x', y') and the coordinate of the origin image be (x, y), let the origin$(0, 0)$ be the center of the image. Analyze the two images we can find the mapping method:



After analyzing we can find that all the

vertical lines are mapped to the curve and all the curves share the same curve function with one ellipse parameter(because they all pass(0,1) and (0,-1). We set the function to be $ax^2 + y^2 = 1$.

Routine

•1.For each pixel (x_0, y_0) in the square region, we first find the corresponding curve parameter. Because the pixel is on the line $x = x_0$ and so the curve pass $(x_0, 0)$. The parameter be $a = \frac{1}{x_0^2}$

•2.The corresponding mapping function is

$$\begin{cases} y_0' = y_0 \\ x_0' = (1 - y_0^)/a^2 \end{cases}$$

I.3 Experiments and Results

platform:Visual Studio 2017, Debug and Release under x64,language C++

Experiments

Perform the warping algorithm to the puppy,tiger and panda images. And restore the image with the warped images

Results

(a) *Original Image* **(b)** *Warped Image* **(c)** *Restore Image*

Figure 151: *puppy*

(a) *Original Image* **(b)** *Warped Image* **(c)** *Restore Image*

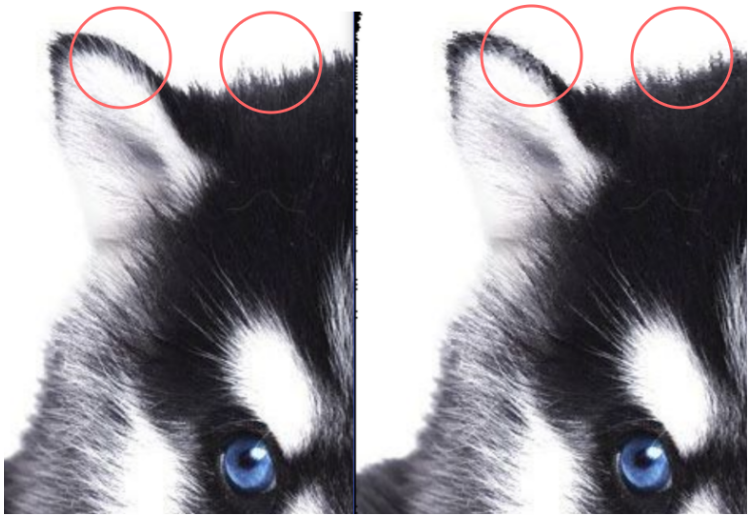
Figure 152: *tiger*

(a) *Original Image* **(b)** *Warped Image* **(c)** *Restore Image*

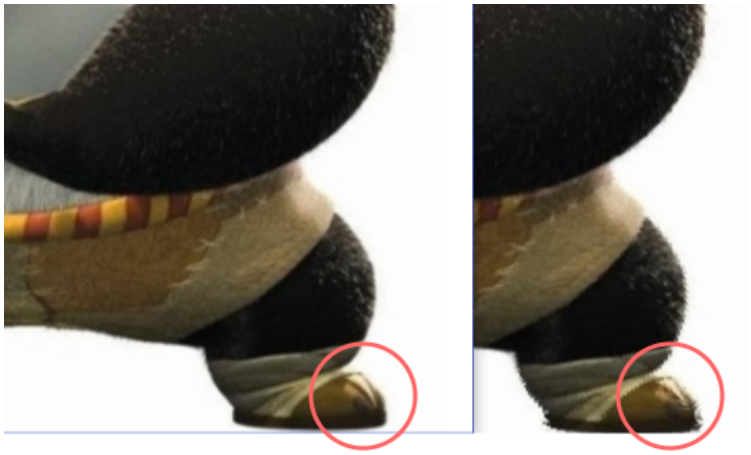
Figure 153: *panda*

I.4 Discussion

Is there any difference between the two images? explain the sources of distortion in detail.
Yes there is some distortion, basic the distortion happens near the edge of the image, the following shows some example, one is the puppy's ear and the other is the panda's shoes.



(a) *Distortion on puppy's ear*



(b) *Distortion on panda's shoes*

Figure 154: *distortion*

The distortion comes from little information provided when restore pixels at the corner, because a batch of corner pixels shrink into small region of pixels. Moreover because the shrink is operated in horizonal direction, there is an **eraser** effect, that those region look like being erased in horizonal direction.

II. Image Stitching (1.b)

II.1 Motivation

We can use homographic transformation and image stitching techniques to create panorama. There are different homographic transformation methods: translation, rotation and scaling. Each transform has its corresponding transform matrix, and we can use augmented coordinates to perform all three transformations together with one matrix. With such homographic transformation we can define some control points to stitch two images together.

II.2 Routine

•1.Define the size of the final plane. Define the width and the height.
•2.Locate the middle image at the center of the plane.
•3.Define the control points, 4 from the left image and 4 from the middle image, attention here, the coordintes of points of the middle image are relative to the plane origin.
•4.Solve the linear system with the 4 control points.
•5.Perform the homographic transform to the four corners of the left images and find the corresponding region in the plane to place the transformed left image.
•6.For each pixels in the region, perform the inverse transform to find the corresponding position of the left image and perform bilinear algorithm to find out the intensity.
•7.Redo step 3 to step 6 and this time use the right image.
•8.Adjust the side of the middle image to make good alignment.

Details in step 3

Figure 155: *The control points to choose between left and middle*

Figure 156: *The control points to choose between middle and right*

Details in step 4

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \implies \begin{cases} H_{11}x_0 + H_{12}y_0 + H_{13} = x'_0 \\ H_{21}x_0 + H_{22}y_0 + H_{23} = y'_0 \\ H_{31}x_0 + H_{32}y_0 + H_{33} = 1 \\ H_{11}x_1 + H_{12}y_1 + H_{13} = x'_1 \\ H_{21}x_1 + H_{22}y_1 + H_{23} = y'_1 \\ H_{31}x_1 + H_{32}y_1 + H_{33} = 1 \\ H_{11}x_2 + H_{12}y_2 + H_{13} = x'_2 \\ H_{21}x_2 + H_{22}y_2 + H_{23} = y'_2 \\ H_{31}x_2 + H_{32}y_2 + H_{33} = 1 \\ H_{11}x_3 + H_{12}y_3 + H_{13} = x'_3 \\ H_{21}x_3 + H_{22}y_3 + H_{23} = y'_3 \\ H_{31}x_3 + H_{32}y_3 + H_{33} = 1 \end{cases}$$

$$\implies \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = \begin{bmatrix} x'_0 \\ y'_0 \\ 1 \\ x'_1 \\ y'_1 \\ 1 \\ x'_2 \\ y'_2 \\ 1 \\ x'_3 \\ y'_3 \\ 1 \end{bmatrix}$$

Details in step 5

Denote coordinates of corner0, corner1, corner2, corner3 as P_0, P_1, P_2,P_3 and the transform matrix as H. find $P' = HP$, get $min(P'_0(x), P'_1(x), P'_2(x), P'_3(x))$ $P(x)$ means the x coordinate of the point. get $max(P'_0(x), P'_1(x), P'_2(x), P'_3(x))$, get $min(P'_0(y), P'_1(y), P'_2(y), P'_3(y))$ $max(P'_0(y), P'_1(y), P'_2(y), P'_3(y))$.

Details in step 6

for every pixel in the region, that is x from x_{min} to x_{max} and y from y_{min} to y_{max} , perform the inverse transform to find the corresponding pixel value.

II.3 result

Figure 157: *stitched image*

II.4 Discussion

What if we use more than four control points?
Clearly for stitching for points is already enough since we at most need to solve 9 parameters, and if we choose over 4 points , we would have more equations, and the **equation system is over-determined**, therefore , we may apply some techniques like MSE to solve the equation system. And there exist situation that all points are not correctly matched but in whole the effect looks good.
How to choose the control points?
After some experiments, I found the best strategy is to satisfy two conditions, first , we can not choose pixels that all have similar horizonal position value, we should choose pixels positions with horizonal postion and vertical postion vary a lot. for example the four points (x_1, y_1),(x_2, y_2),(x_3, y_3),(x_4, y_4), the $Var(\underline{x})$ and $Var(\underline{y})$ should not be small. Second, the points should be representive, like an corner of an object or a typical spot.

XVIII. Digital Half-toning

I. Dithering(2.a)

I.1 Motivation

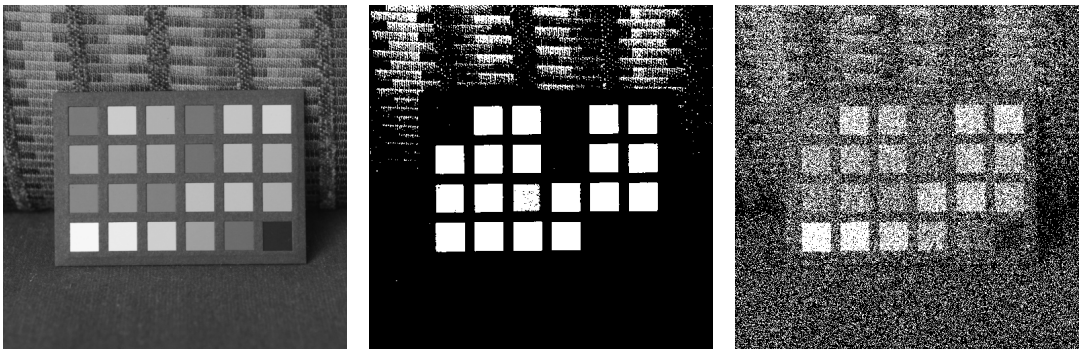
If we are going to print something, in black and white, for most of the printers we use, we only have black ink. We use how dense the dots are to show different gray. Therefore, for a given image, we need to decide how to arrang the dots. What if dpi (dots per pixel) is one, which means for each pixel we can only choose black or white, what is the strategy to make the image look just like the origin image, several algorithms were proposed.

I.2 Algorithms

1. Fixed thresholding
We can choose a threshold and for each pixel value larger than the threshold we assign white and else we assign black.
2. Random thresholding
For each pixel, we randomly select a threshold, if the pixel value is larger than the threshold , we assign white and else we assign black.
3.Dithering matrix
We can use dithering matrix which looks like a threshold matrix to decide which pixels to turn on. For example, the basic matrix is $I_2 = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$ and the corresponding threshold matrix becomes $T = \begin{bmatrix} \frac{1.5}{4} & \frac{2.5}{4} \\ \frac{3.5}{4} & \frac{0.5}{4} \end{bmatrix}$. And it is easy to see that a block with pixels that have larger intensity tend to turn on more postions(let that position have intensity 255)and a gray block tends to turn on less positions. There are also larger dithering matrices defined recursively using these formula:
$I_{2n} = \begin{bmatrix} 4 \cdot I_n(x,y) + 1 & 4 \cdot I_n(x,y) + 2 \\ 4 \cdot I_n(x,y) + 3 & 4 \cdot I_n(x,y) \end{bmatrix}$

I.3 Experiments and Results

platform:Visual Studio 2017, Debug and Release under x64,language C++



(a) *Original Image* **(b)** *Fixed Threshold* **(c)** *Random Threshold*

Figure 158: *Algorithm 1 and Algorithm2*

The I_4, I_8 matrices are:

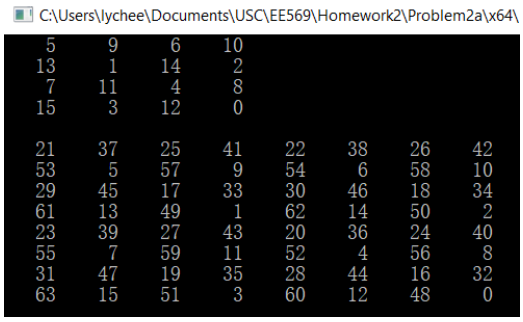
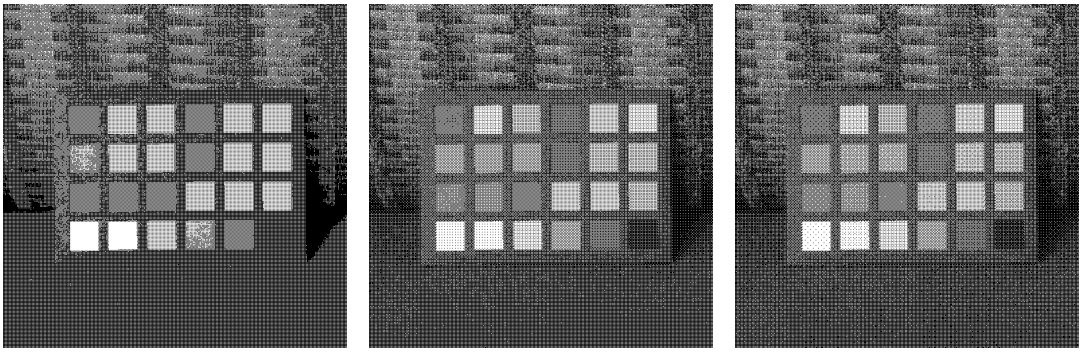


Figure 159: I_4, I_8 *matrices*

Results:



(a) I_2 **(b)** I_4 **(c)** I_8

Figure 160: *Results with dithering matrices*

Comment

As we can see from the figure, the larger matrix pattern provides more patterns and the result is better.

I.4 If four intensity can be displayed

Algorithm design

Step1: pattern choice Now since I have 4 intensities to choose, I can create more patterns than the I2,I4. The following figure shows the example patterns:

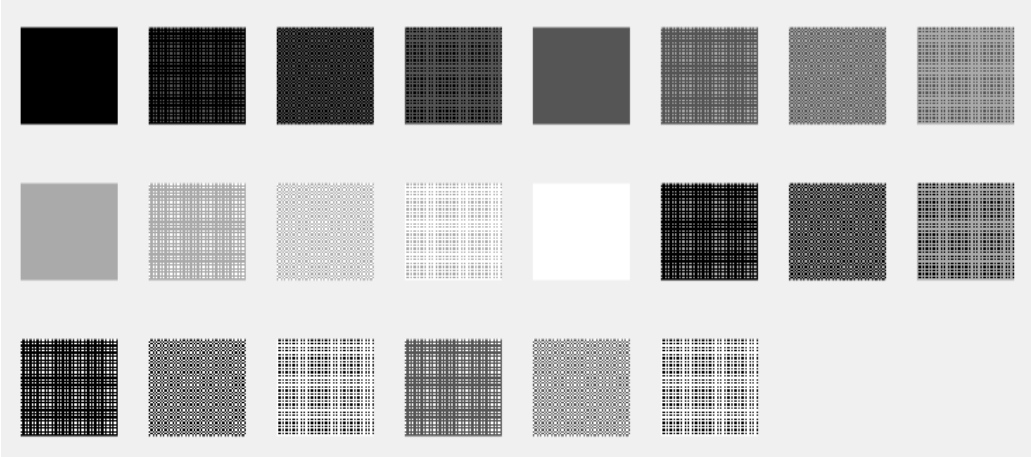


Figure 161: *Example of pattern*

In order to make the image seem smooth , I choose the following patterns, the combinations can be:

$$C_1\begin{bmatrix}0&0\\0&0\end{bmatrix},C_2=\begin{bmatrix}0&0\\0&85\end{bmatrix},C_3=\begin{bmatrix}85&0\\0&85\end{bmatrix},C_4=\begin{bmatrix}85&85\\0&85\end{bmatrix},C_5=\begin{bmatrix}85&85\\85&85\end{bmatrix},C_6=\begin{bmatrix}85&85\\85&170\end{bmatrix}$$

$$C_7=\begin{bmatrix}170&85\\0&170\end{bmatrix},C_8=\begin{bmatrix}170&85\\85&170\end{bmatrix},C_9=\begin{bmatrix}170&170\\0&170\end{bmatrix},C_{10}=\begin{bmatrix}170&170\\85&170\end{bmatrix},C_{11}=\begin{bmatrix}170&170\\170&170\end{bmatrix}$$

$$C_{12}=\begin{bmatrix}170&170\\85&255\end{bmatrix},C_{13}=\begin{bmatrix}170&170\\170&255\end{bmatrix},C_{14}=\begin{bmatrix}255&170\\85&255\end{bmatrix},C_{15}=\begin{bmatrix}255&170\\170&255\end{bmatrix},C_{16}=\begin{bmatrix}255&255\\85&255\end{bmatrix}$$

$$C_{17}=\begin{bmatrix}255&255\\170&255\end{bmatrix},C_{18}=\begin{bmatrix}255&255\\255&255\end{bmatrix}$$

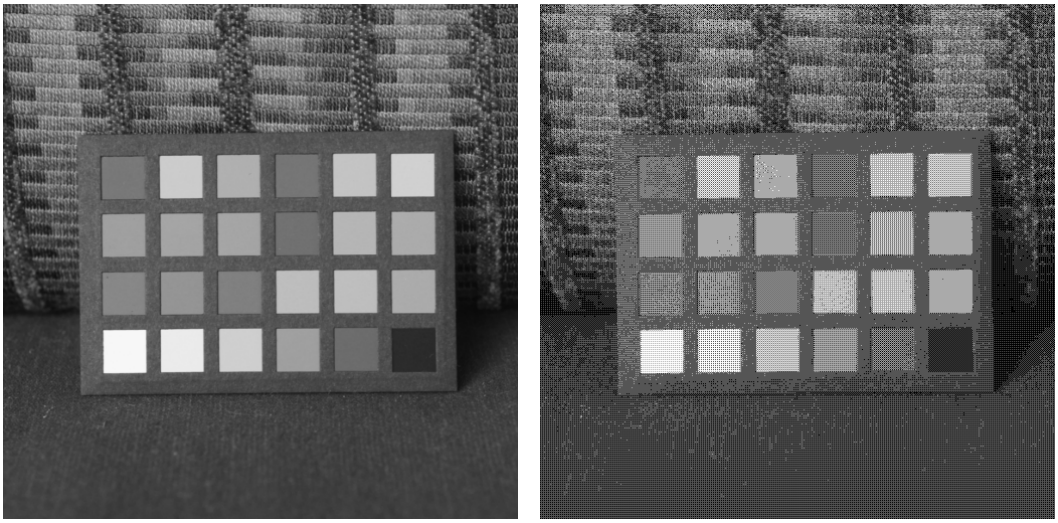
Because people are more sensive to changes in bright place, I assign more patterns to the bright part.

Step2: pattern decision

For each pixel in the image, we check the intensity v:

```
if v < 85 :
    index = round(v/(85/4.0))  // choose from C1 to C5
else if v < 170:
    index = round((v-85)/(85/6))+4 // choose from C5 to C11
else
    index = round((v-170)/(85/7))+10 // choose from C11 to C18
```

Step3:intensity decision
After choose the pattern , the result of that pixel(i,j) becomes $C_x[i\%2][j\%2]$.
Result



(a) *Original Image* **(b)** *The result of the algorithm*

Figure 162: *The result of the algorithm*

I.5 Discussion

Compare the algorithms above, using the dithering matrix can effectively improve the quality visually, however,we can see a lot of dot patterns, which lead to a lot of high frequency components, that some details may be subotaged.

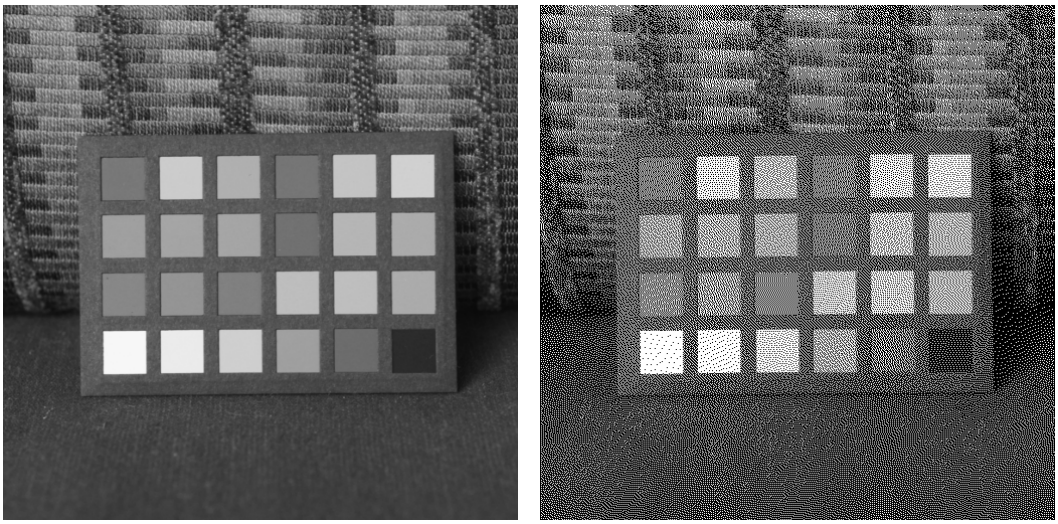
II. Error Diffusion 2(b)

II.1 Algorithm

Another method is error difusion, the idea is to distribute the error to the following neibourhood pixels. We get the quantization error of the current pixel, if the error is big, the surronding pixels' quantization error would be suppressed.
Algorithm
•1.For current pixel, calculate the quantization error
•2.Add the error on the neighbourhood pixels according to the error diffusion matrix.
•3.Move to the next pixel, repeat step1 and step2.
Error Diffsion Matrix

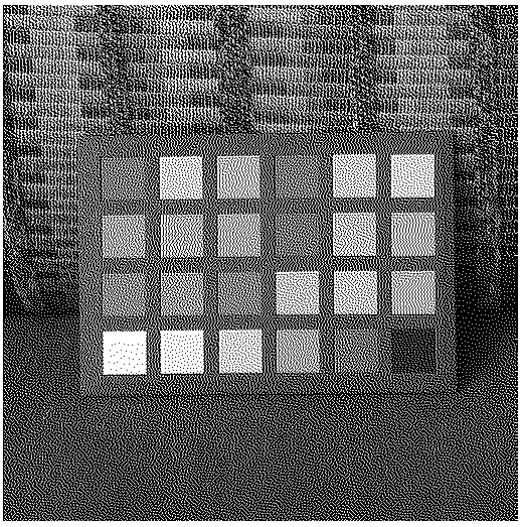
Floyd-Steinberg: $M_{error} = \frac{1}{16}\begin{bmatrix} 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$ JJN: $M_{error} = \frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$ Stucki: $M_{error} = \frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$

II.2 Results

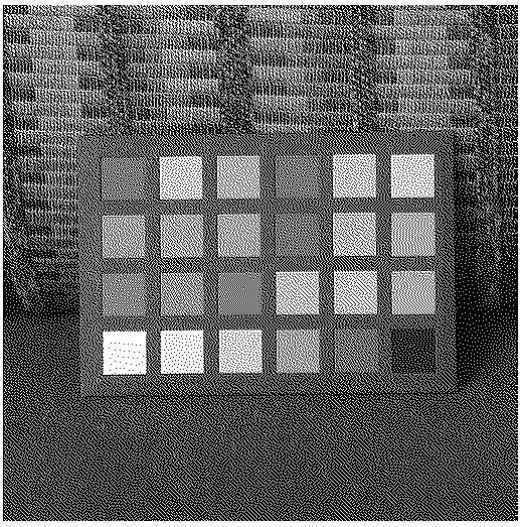


(a) *Original Colorchecker Board*

(b) *Floyd-Steinberg*



(c) *Jarvis,Judice,Ninke*



(d) *Stucki*

Figure 163: *Error Diffusion*

II.3 Discussion

We can that the three error diffusion matrix actually have similar visual effect, unlike the dithering matrix, they have patterns based on the origin data of the image. However, the dot effect kinds affect the visual result. That, the table texture is destroyed , and the total images looks brighter. My idea to improve the result is to use different error diffusion matrix to some level of pixel intensity.

my proposed routine

•1 segment image into several non-overlapping blocks using some window.

•2 cluster these clock into k class

•3 design k different error diffusion matrix.

•4 For each pixel location, gather neighbour to form a block and find which diffustion matrix to use.

III. Color Halftoning

III.1 Motivation and Background

When it comes to color printing, we do not have a lot of ink box of different colors, we also need to use techniques like digital half toning to use small amount of colors to mimic many different colors. This section discussed two method, one is separable error diffusion, and the other is MBVQ-based error diffusion.

III.2 Algorithm

Separable Error Diffusion

Each channel is a gray scale image, after halftoning, each channel we can have 2 values, 0 or 255, therefore we have eight colors.

$W = (0,0,0), Y = (0,0,1), C = (0,1,0), M = (1,0,0), G = (0,1,1), R = (1,0,1), B = (1,1,0), K = (1,1,1)$

Algorithm

•1.Convert the RGB image into CMY image.

•2.For each CMY channel, performing the error diffusion.

•3.Combine the three channels.

MBVQ-based Error Diffusion

Instead of performing quantization for each channel, we consider the true color of the pixel and trys to find the nearest color in the color set. We conclude that we can render every input color using one of six half tone quad ruples:RGBK,WCMY,MYGC,RGMY,RGBM,orCMGB,each with obviously minimal brightness variation. [1]

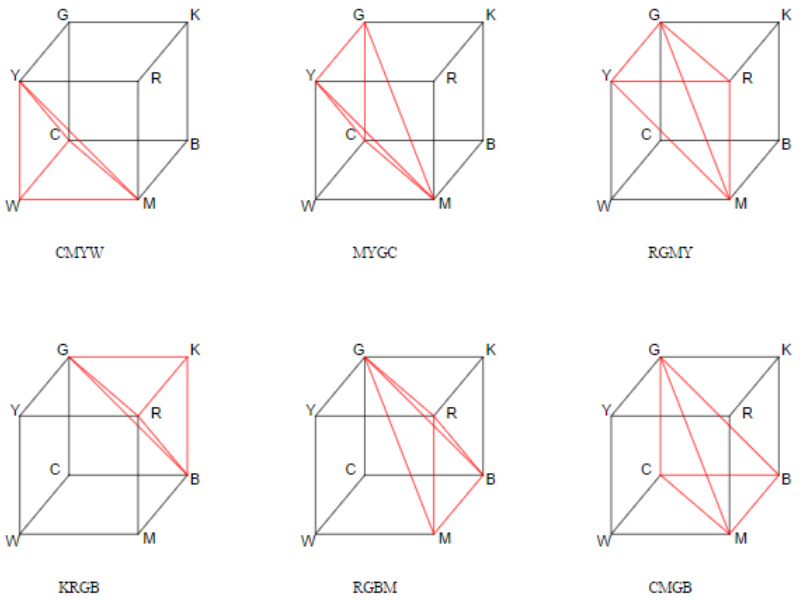


Figure 164: *Decompose the CMY cube into 6 different part*

pseudo code to find nearest color block

```
if ((r + g) > 255)
    if ((g + b) > 255)
        if ((r + g + b) > 510)
            return CMYW;
        else
            return MYGC;
    else
        return RGMY;
else
    if ((g + b) <= 255)
        if ((r + g + b) <= 255)
            return KRGB;
        else
            return RGBM;
    else
        return CMGB;
```

After finding the corresponding MBVQ quandrant, find the nearest vertex and then quantize each channel and distribute the error.

Algorithm

•1.For each pixel at position(x,y), determine its MBVQ quadrant based on RGB(x,y)

•2.Find the vertex V of the MBVQ tetrahedron to closest CMY(x,y) + e(x,y)

•3.Compute the new quantization error using CMY(x,y) + e(x,y) - V

•4.Distribute the quantized error to the future pixel error using a standard error diffusion process.

III.3 Results

(a) *Original Image* **(b)** *Separable Error Difusion* **(c)** *MBVQ-based Error Diffusion*

Figure 165: *Color Diffusion Results*

III.4 Discussion

It is not obvious to see the difference between the two techniques, but we can see differences when we zoomed in to the image.

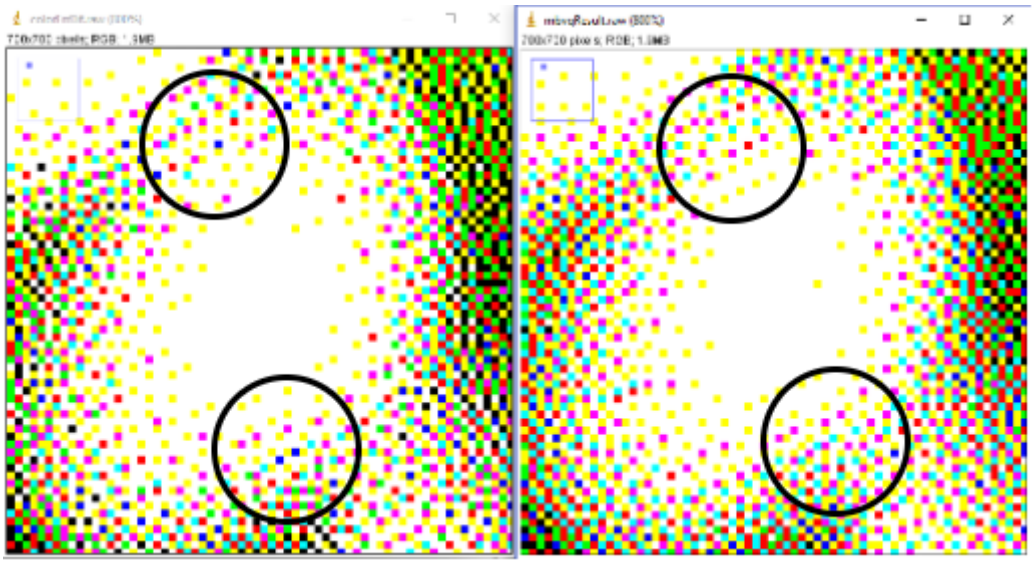


Figure 166: *Comparison between two techniques*

The left is seperable error diffusion, and the right is MBVQ-based error diffusion. We can see

that some dots that is actually wrong becasue of seperable error diffusion. The seperable error diffusion method relies on one channel too much that once that channel is actually vary from the actual value, the pixel value would be sabotaged. However, the right image elliminate the effect, as we can see from the figure above, the blue dot surrounded by the magenta becomes magenta using MBVQ-based error diffusion method .

XIX. Morphological Processing

I. Motivation

When dealing with some binary images, we can use some morphological operations to perform erosion or filling job so that it is easier to perform some advanced analysis like recognition or the counting job. For erosion, there are three methods, shrink , thining and skeletonizing. The method performes like filtering and uses a 3×3 size patterns to shift over all pixels. The binary value of a pixel is modified according to the pattern and structured image data.

II. Algorithm

•1 At each pixel location(j,k), if the pixel is foreground data, gathering the 8-connected neighbour pixels

•2 Use conditional pattern table to decide whether this pixel should be marked

•3 For marked pixels, using the unconditional pattern table to decide which mark should be modified, denote M(j,k)

•4 The final image $I(j,k)' = I(j,k) \cap \overline{M(j,k)}$

•5 Repeat step2 to step4 until the image stop change any more.

Example

1.First Iteration

image data: $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, conditional pattern $P1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$,$P2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$, unconditional pattern: $P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

Step2: pixels that hit is marked as M: $M_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & M & 0 & 0 & M & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, Step3: using unconditional $M_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & M & 0 & 0 & M & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, Step4: $I_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

2.Second Iteration

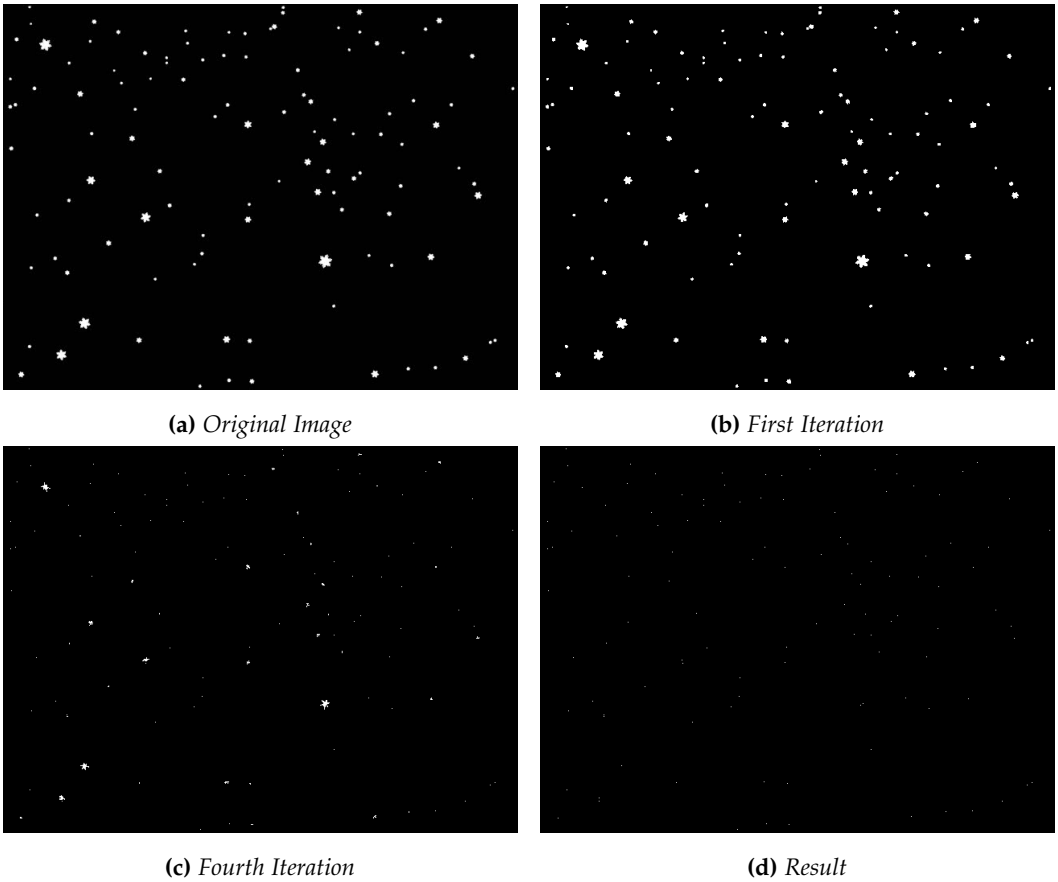
image data: $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ Step2: pixels that hit is marked as M: $M_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & M & M & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$,

Step3: using unconditional $M_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & M & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, Step4: $I_{j,k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

III. Problem3(a)

Using the shrinking filter to shrink the stars in the image. Count the number of the stars and the histogram of the star size.

III.1 Result



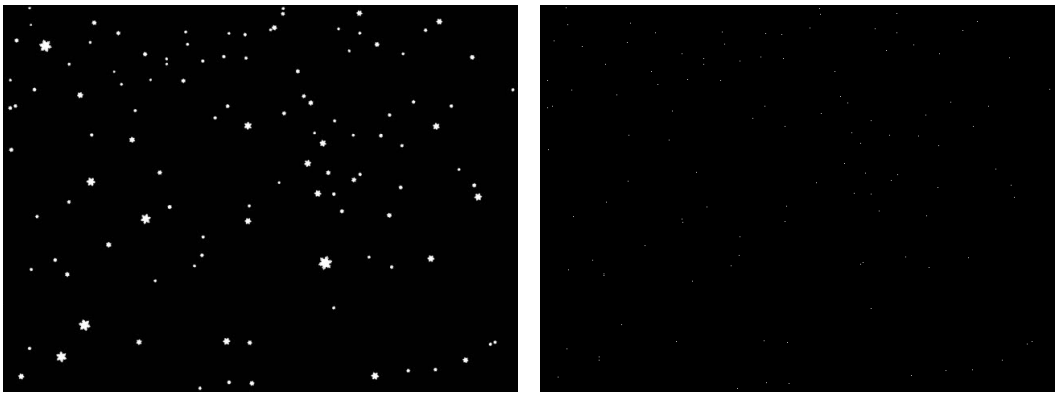
(a) *Original Image* **(b)** *First Iteration*

(c) *Fourth Iteration* **(d)** *Result*

Figure 167: *Shrink result*

III.2 Discussion

The total number of stars:



(a) *original Image* **(b)** *Result*

Figure 168: *some failure position*

```
star size: cdf
iter:0    one size star number:0
iter:1    one size star number:7
iter:2    one size star number:59
iter:3    one size star number:79
iter:4    one size star number:90
iter:5    one size star number:97
iter:6    one size star number:105
iter:7    one size star number:107
iter:8    one size star number:111
iter:9    one size star number:112
iter:10    one size star number:112
iter:11    one size star number:115
star size: pdf
size 1  star number: 7
size 2  star number: 52
size 3  star number: 20
size 4  star number: 11
size 5  star number: 7
size 6  star number: 8
size 7  star number: 2
size 8  star number: 4
size 9  star number: 1
size 10  star number: 0
size 11  star number: 3
complete
star number:115
```

Figure 169: *star size histogram*

We got the total number of stars 115. However, as the first two images shown, three large stars were shrinked into 2 dots. Therefore the total star number is 112.
And the figure above show the star size histogram. As the figure shows, the star size is defined as after how many iterations the star is shrinked into one dot.

IV. Problem3(b)

Using the thinning filter to the jigsaw image.

IV.1 Result

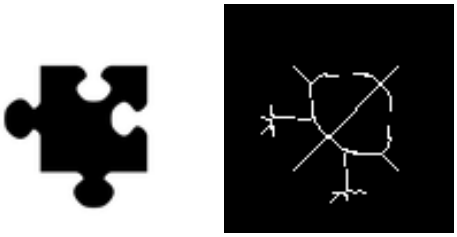
(a) *Original Image* **(b)** *Thinning*

Figure 170: *Thining Result (after inversing the origin image)*

V. Problem3(c)

Using the skeletonization to the jigsaw image.

V.1 Result



(a) *Original Image* **(b)** *Skeletonize*

Figure 171: *Skeletonize Result (after inversing the origin image)*

VI. Problem3(d)

VI.1 Problem Description

Counting game: there is a borad of different jigsaws, we would first like to find out how many jigsaws are in the board image, then to find out how many unique jigsaws.

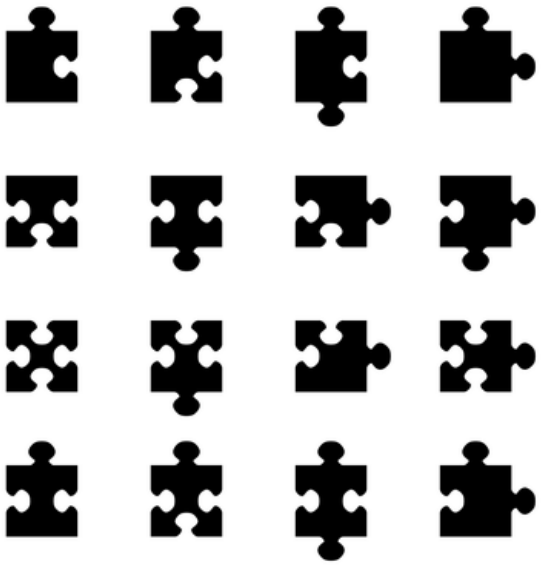


Figure 172: *Board Image*

VI.2 Routine

•*Step*1 Use component analysis to segment the jigsaws out.
•*Step*2 For each jigsaw treat it as a pattern, perform rotation and flip operation and compare it to the rest jigsaws.

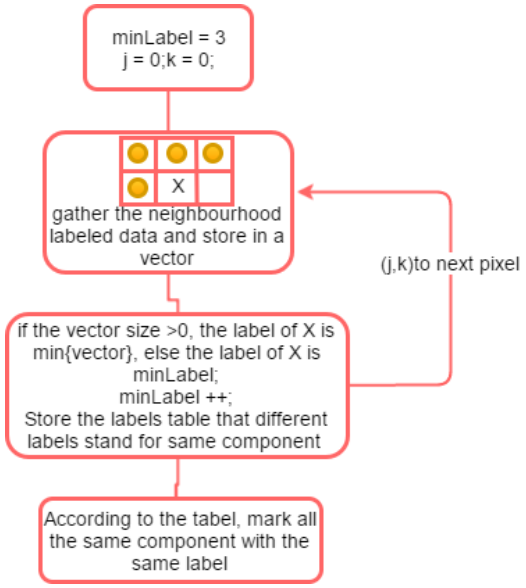


Figure 173: *Details in step 1*

Details in Step 2

```
for i = 1 to jigsaw number
    for j = i+1 to jigsaw number
        if jigsaw_i.width == jigsaw_j.width  && jigsaw_i.height == jigsaw_j.height
                v0 = jigsaw_i
                v1 = flip jigsaw_i
                v2 = rotate jigsaw_i 180 degree
                v3 = flip then rotate jigsaw_i
                if v0 == jigsaw_j  match and break;
                if v1 == jigsaw_j  match and break;
                if v2 == jigsaw_j  match and break;
                if v3 == jigsaw_j  match and break;
        else if jigsaw_i.width == jigsaw_j.height  && jigsaw_i.height == jigsaw_j.width
                v4 = rotate jigsaw_i 90 degree
                v5 = flip jigsaw_i then rotate 90 degree
                v6 = rotate jigsaw_i 270 degree
                v7 = flip jigsaw_i then rotate 270 degree
                if v4 == jigsaw_j  match and break;
                if v5 == jigsaw_j  match and break;
                if v6 == jigsaw_j  match and break;
                if v7 == jigsaw_j  match and break;
```

VI.3 Result

```
total number of jigsaws:16
start the segmentation job
different jigsaws are stored in the following file with dimensions:
jigsaw1.raw:    Width:46    Height:62
jigsaw2.raw:    Width:46    Height:62
jigsaw3.raw:    Width:46    Height:78
jigsaw4.raw:    Width:62    Height:62
jigsaw5.raw:    Width:46    Height:46
jigsaw6.raw:    Width:46    Height:62
jigsaw7.raw:    Width:62    Height:46
jigsaw8.raw:    Width:62    Height:62
jigsaw9.raw:    Width:46    Height:46
jigsaw10.raw:    Width:46    Height:62
jigsaw11.raw:    Width:62    Height:46
jigsaw12.raw:    Width:62    Height:46
jigsaw13.raw:    Width:46    Height:62
jigsaw14.raw:    Width:46    Height:62
jigsaw15.raw:    Width:46    Height:78
jigsaw16.raw:    Width:62    Height:62
complete gathering jigsaws
start find unique
find identical:   jigsaw2  and  jigsaw7    rotate 90 degrees
find identical:   jigsaw2  and  jigsaw11    rotate 90 degrees then flip
find identical:   jigsaw6  and  jigsaw13    rotate 180 degrees
find identical:   jigsaw8  and  jigsaw16    flip
find identical:   jigsaw10  and  jigsaw12    rotate 270 degrees
find identical:   jigsaw10  and  jigsaw14    rotate 180 degrees
number of unique:10
```

Figure 174: *Program Result*

In total there are 16 jigsaws, and 10 unique jigsaws.

XX. Texture Analysis and Segmentation

I. Motivation

In nature we have different textures, like rocks, grass, bricks etc. There are different ways to seperate and recognize different textures, like using the spacial filters or using some dataset for some learning. For the spacial filter methods, one of them is using Laws filters. And for the learining dataset, we can use Berkley Segmentation dataset.

II. Method

The two-dimension Laws filters are generated with one-dimension Laws filter, there are 5 basic one-dimension kernel.

Table 34: *1D Laws Filter*

Name	Kernel
L5 (Level)	**[1 4 6 4 1]**
E5(Edge)	**[-1 -2 0 2 1]**
S5(Spot)	**[-1 0 2 0 -1]**
W5(Wave)	**[-1 2 0 -2 1]**
R5(Ripple)	**[1 -4 6 -4 1]**

The 2D kernel is generated with outer-product. For example, we have two vectors : $\underline{v_1} = [v_{11}, v_{12}, v_{13}, v_{14}, v_{15}]^t$ and $\underline{v_2} = [v_{21}, v_{22}, v_{23}, v_{24}, v_{25}]^t$. The 2d kernel is $\underline{v_1} \cdot \underline{v_2}^t$. For example, the kernel $L5E5$ is generated:

$$\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}^t \cdot \begin{bmatrix}-1 & -2 & 0 & 2 & 1\end{bmatrix} = \begin{bmatrix}-1 & -2 & 0 & 2 & 1\\ -4 & -8 & 0 & 8 & 4\\ -6 & -12 & 0 & 12 & 6\\ -4 & -8 & 0 & 8 & 4\\ -1 & -2 & 0 & 2 & 1\end{bmatrix}$$

III. Problem 1(a) Texture Classification

III.1 Routine

•Step 1. Generate 9 Laws 2D filters with E5, S5 and W5.
•Step 2. For each image, generate 9 filter responses images.
•Step 3. Calculate the average energy. For each response image, calculate the average response and now each image has 9 average response.
•Step 4. Using K - means to cluster the $R^{12\times 9}$ dataset.

III.2 K-mean

Initiation method:

•Method 1 , initiate with mean and standard deviation

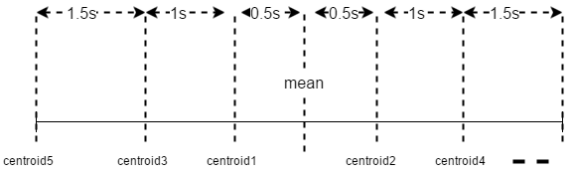


Figure 175: *Initilization of k-mean*

To initiate the centroids, I first calculate the mean and standard deviation of the datas of the given data.

•Method 2, initiate with random selection
Randomly select K data points from the data set as the initial centroids.

• Method 3, bootstrap
Randomly select points from the dataset, and generate new data points by combine several points, then randomly select nK points and average to get K initial points.

A class named VQ is created, which has public functions for users to call initilazation. The following figure show how I initilize the centroids.s is the standard deviation of all the training data.
Since if we were given a large amount of data, we may assume laws of large numbers, my assumption is all image may subject to some distribution, and we use the nonlinear standard deviation to get the initial centriods.

Update routine

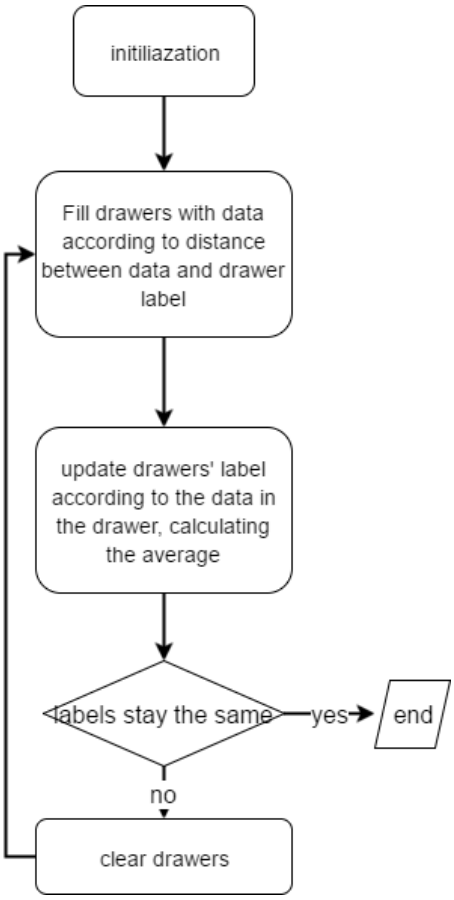


Figure 176: *update routine of k-mean*

III.3 Result

Perform the experiment on the 12 images, we get:

platform:Visual Studio 2017, Debug and Release under x64,language C++

```
the responses are:
texture1.raw:{14.9765, 11.021, 6.21116, 10.8715, 8.71145, 5.2311, 5.77893, 4.89916, 3.06536, },
texture2.raw:{5.3533, 4.33472, 2.36832, 3.71186, 3.16277, 1.83188, 1.83741, 1.64373, 1.02319, },
texture3.raw:{3.84403, 2.40382, 1.20675, 2.77056, 1.79193, 0.932069, 1.53716, 1.02052, 0.542256, },
texture4.raw:{15.9524, 11.7656, 6.49088, 12.1026, 9.40367, 5.39821, 6.3864, 5.26131, 3.16443, },
texture5.raw:{4.42875, 2.66357, 1.2325, 3.16456, 1.94945, 0.916155, 1.70395, 1.0757, 0.510028, },
texture6.raw:{15.4699, 11.5567, 6.50259, 11.451, 9.20069, 5.50043, 6.20529, 5.2801, 3.28889, },
texture7.raw:{1.29648, 1.21755, 0.869859, 1.2348, 1.21111, 0.883932, 0.877616, 0.885823, 0.657504, },
texture8.raw:{1.35875, 1.2271, 0.882662, 1.23929, 1.21809, 0.901754, 0.867971, 0.886731, 0.664761, },
texture9.raw:{4.13093, 2.93764, 1.40683, 2.78776, 2.11532, 1.07585, 1.43169, 1.1471, 0.610903, },
texture10.raw:{1.25172, 1.00096, 0.553072, 1.12624, 0.959056, 0.548048, 0.761286, 0.671151, 0.396981, },
texture11.raw:{3.24239, 2.05426, 1.07669, 2.13296, 1.37011, 0.737882, 1.00399, 0.653177, 0.362213, },
texture12.raw:{5.61105, 4.63431, 2.63311, 4.00521, 3.65101, 2.25234, 2.21327, 2.13359, 1.42372, },
start classifying
data size:      12
data dimension: 9
cluster into 4 sets
initial Centroids:
7.40741 5.43005 3.13741 5.72255 4.41544 2.59867 3.33541 2.79715 1.71855
8.58404 6.56946 3.77253 6.31444 5.28645 3.22759 3.45575 2.18796 0.968346
8.90514 4.89403 2.27857 4.20637 3.28687 1.86487 2.31778 1.80094 1.09931
10.091 7.48125 3.04267 5.3633 4.29048 3.51234 4.00191 2.77021 1.41327
training:1
5.61105 4.63431 2.63311 4.00521 3.65101 2.25234 2.21327 2.13359 1.42372
0 0 0 0 0 0 0 0 0
3.1133 2.22995 1.19959 2.271 1.72223 0.978446 1.25263 0.997992 0.595979
15.4663 11.4478 6.40154 11.475 9.10527 5.37658 6.12354 5.14686 3.17289
training:2
5.48218 4.48452 2.50072 3.85854 3.40689 2.04211 2.02534 1.88866 1.22346
1.25172 1.00096 0.553072 1.12624 0.959056 0.548048 0.761286 0.671151 0.396981
3.05022 2.08399 1.11255 2.22165 1.60934 0.90794 1.23706 0.944842 0.557944
15.4663 11.4478 6.40154 11.475 9.10527 5.37658 6.12354 5.14686 3.17289
training:3
5.48218 4.48452 2.50072 3.85854 3.40689 2.04211 2.02534 1.88866 1.22346
1.30232 1.14854 0.768531 1.20011 1.12942 0.777911 0.835624 0.814569 0.573082
3.91153 2.51482 1.23069 2.71396 1.80671 0.915489 1.4192 0.974125 0.50635
15.4663 11.4478 6.40154 11.475 9.10527 5.37658 6.12354 5.14686 3.17289
result
texture1.raw in class 3
texture2.raw in class 0
texture3.raw in class 2
texture4.raw in class 3
texture5.raw in class 2
texture6.raw in class 3
texture7.raw in class 1
texture8.raw in class 1
texture9.raw in class 2
texture10.raw in class 1
texture11.raw in class 2
texture12.raw in class 0
```

Figure 177: *Problem 1(a) experiment result*

III.4 Comment

Computer Result:

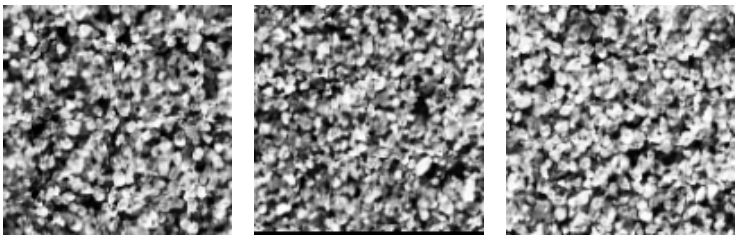
texture1, texture4, texture6 are in the same class.

texture2, texture12 are in the same class.

texture3, texture5, texture9, texture11 are in the same class.

texture7, texture8, texture10 are in the same class.

Visual Result:

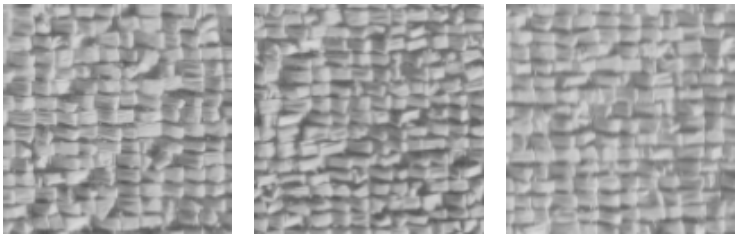


(a) *texture1.raw* **(b)** *texture4.raw* **(c)** *texture6.png*

Figure 178: *texture1,4,6 are in the same class*

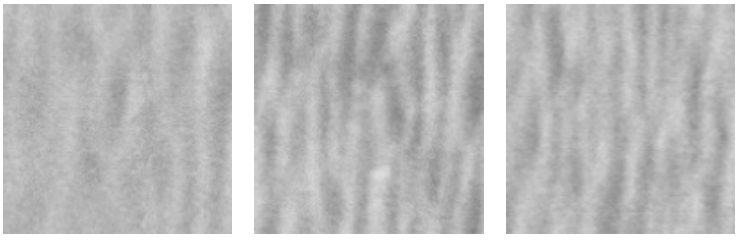
(a) *texture2.raw* **(b)** *texture9.raw* **(c)** *texture12.raw*

Figure 179: *texture2,9,12 are in the same class*



(a) *texture3.raw* **(b)** *texture5.raw* **(c)** *texture11.raw*

Figure 180: *texture3,5,11 are in the same class*



(a) *texture7.raw* **(b)** *texture8.raw* **(c)** *texture10.raw*

Figure 181: *texture1,4,6 are in the same class*

As the figures show that, only **texture9.raw** is misclassified. And we can see that this grass

image is very blur which could lead to such misclassification

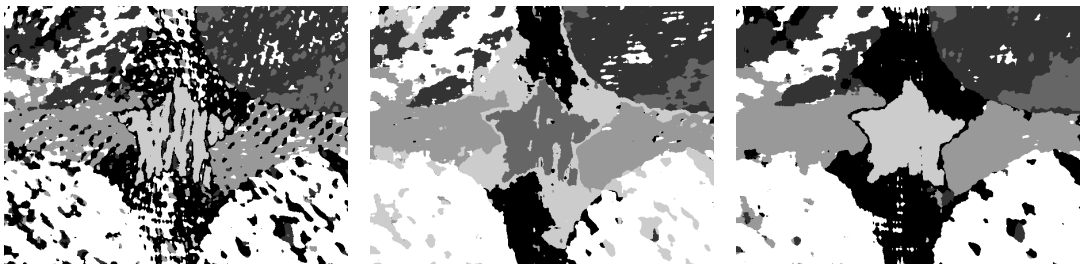
IV. Texture Segmentation

IV.1 Experiment Routine

According to the experiment above, for an image with different textures, we can extract different texture patch with laws filter, in this part we use Laws filter with 3×3 dimension, the $1d$ filters are $[1, 2, 1]^t$, $[2, 0, -2]^t$, $[1, -2, 1]^t$.
Step1: We apply all 9 law filters to the image and get 9 channel.
Step2: Define a window size and for each pixel location calculate the energy, do this for all 9 channels.
Step3: Now at each pixel location there is $9 - D$ feature vector and we use the first energy $L_3^t L_3$to normalize the rest feature.
Step4: Now the dataset is developed we use K-means to find the 6 centroids and assign each pixel a label. And a label image is generated. we use $(0, 51, 102, 153, 204, 255)$ to denote six segmented regions.

IV.2 Result

platform:Visual Studio 2017, Debug and Release under x64,language C++



(a) *Window Size 11* **(b)** *Window Size 17* **(c)** *Window Size 21*

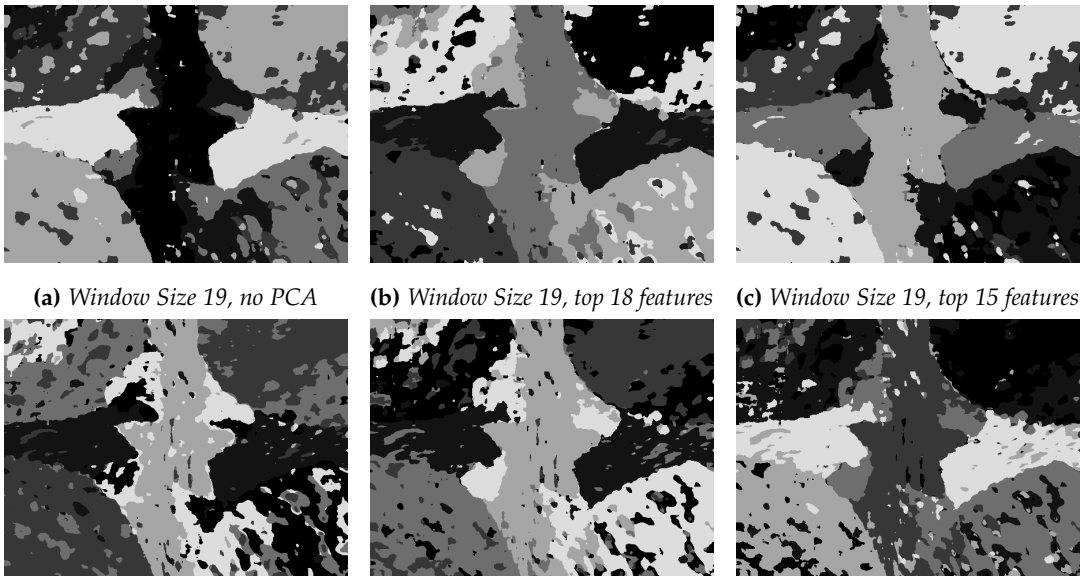
Figure 182: *Result with different window size*

IV.3 Improvement(1c)

Now we adopt all the 25 5×5 Laws filters to this image and try to get a better result, the experiment routine is the same as the above instead we change the filters. The following 6 experiment are performed.

Table 35: *Experiemnts*

Experiment index	window size	PCA
1	19	none
2	19	retain 18 features
3	19	retain 15 features
4	15	none
5	15	retain 18 features
6	15	retain 15 features



(a) *Window Size 19, no PCA* **(b)** *Window Size 19, top 18 features* **(c)** *Window Size 19, top 15 features*

(d) *Window Size 15, no PCA* **(e)** *Window Size 15, top 18 features* **(f)** *Window Size 15, top 15 features*

Figure 183: *Experiments Results*

Results

IV.4 Disussion

This sigmentation method has some inner problem, if we choose a larger window size, the pixels at the border would contain two textures. In this situation a lot of not clean data points would be used in k-mean method. However, if we choose a small window size, some information would be lost, and this could lead to mis-classification.

Basicly in this problem PCA does not give us a better result, since the PCA just changes the place and direction and correlation of the corrdinate, and I think to further improve the performance , we should take some **'clean' patches** out and train the classifier, then we can get a better result to

perform the segmentation job.

XXI. Edge Detection

I. Motivation

Edge detection is basic and important in computer vision and image segmentation. An edge is a set of connected pixels that lie on the boundary between two regions. An ideal edge is a set of connected pixels, each of which is located at an step transition in gray level($0 \rightarrow 1$). In practise, edges sometimes are blurred so the grayscale gradually from $0 \rightarrow 1$, as a result, edges are more closely modeled as having a ramplike profile. With such model, we can construct detectors and kernels to detect edge. Basiclly all the spatial filter that filters an edge out in frequency domain they are the high-pass filter.

II. Basic Edge Detector(Problem 2(a))

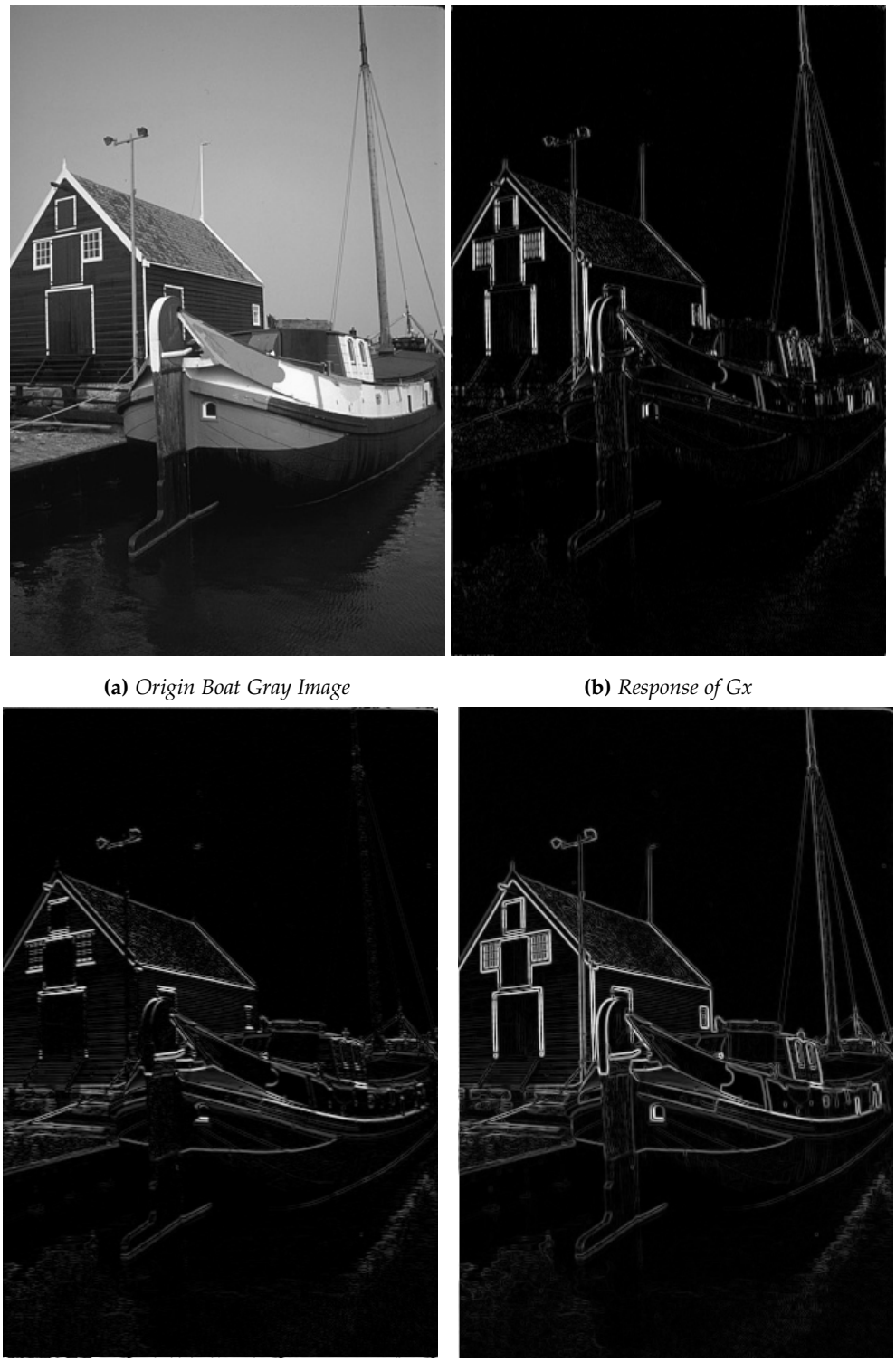
II.1 Sobel Detector

Kernel The kernel of Sobel Detector is:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, magnitude = \sqrt{g_y^2 + g_x^2}, \theta = arctan(g_y/g_x)$$

θ can be used in the Non-Maximun suppression.

Filter Result **Clean Image** platform:Visual Studio 2017, Debug and Release under x64,language C++



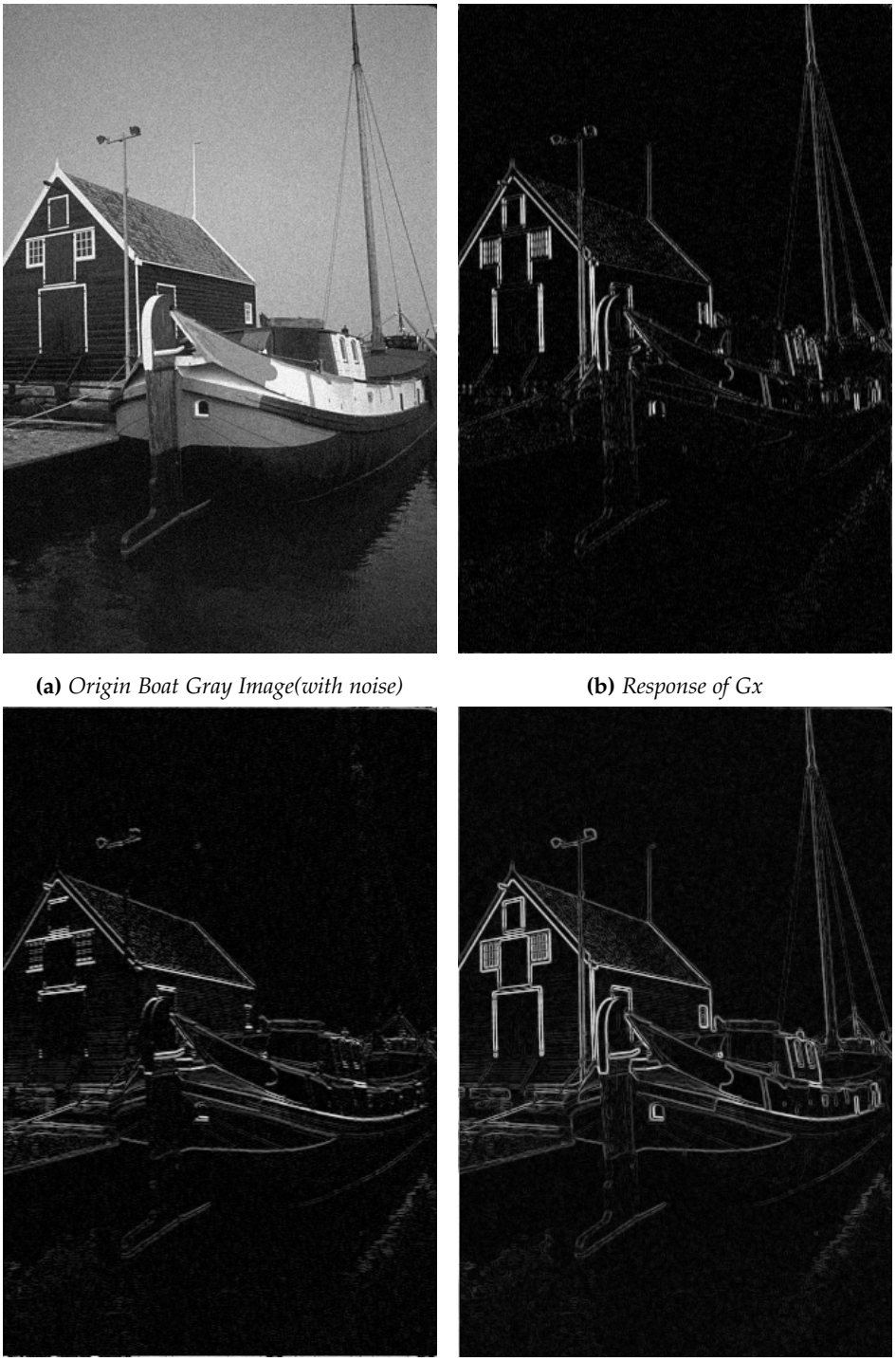
(a) *Origin Boat Gray Image* **(b)** *Response of Gx*

(c) *Response of Gy* **(d)** *Magnitude of Response*

Figure 184: *Sobel Detector of Boat Result*

Noisy Image

platform:Visual Studio 2017, Debug and Release under x64,language C++



(a) *Origin Boat Gray Image(with noise)*

(b) *Response of Gx*

(c) *Response of Gy*

(d) *Magnitude of Response*

Figure 185: *Sobel Detector of Boat Result(Noisy)*

Edge Map The cdf of the magnitude response is : As the figure shows, to keep the top 5% we should set the threshould to be:71 , and to keep the top 10% we should set the threshold to be 37.

clean image

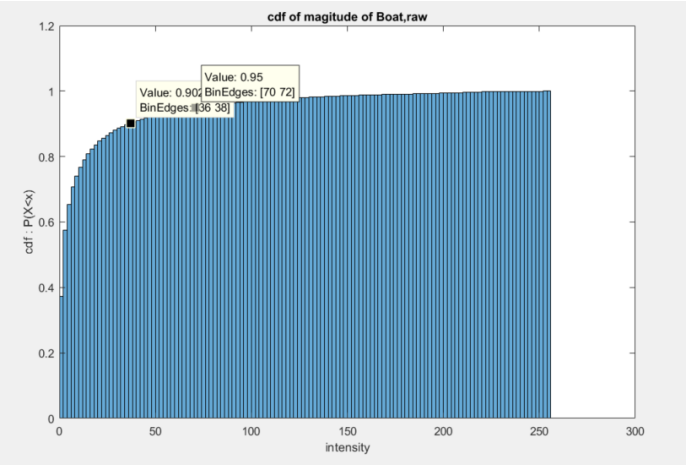
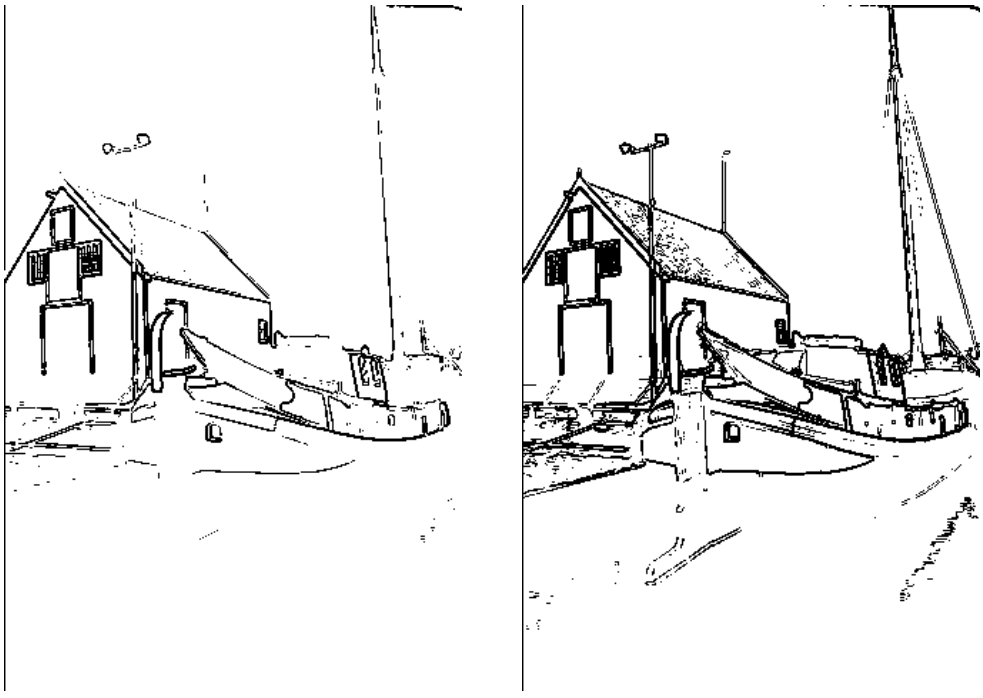


Figure 186: *CMF of magnitude response*

Result with threshold



(a) *Keep top 5% of the magnitude* **(b)** *Keep top 10 % of the magnitude*

Figure 187: *Edge map*

noisy image

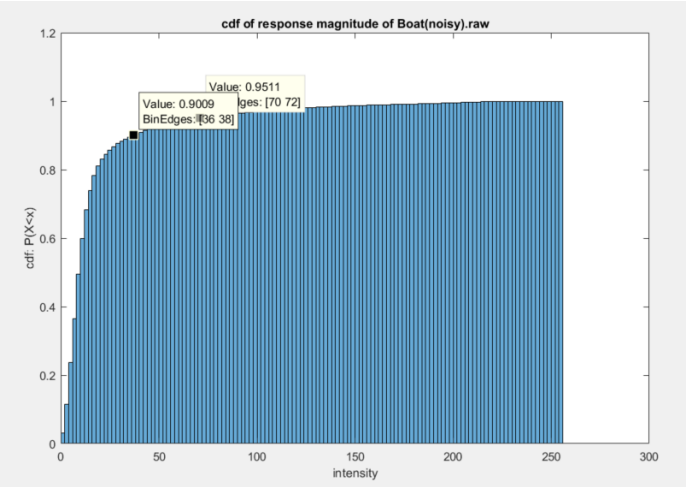
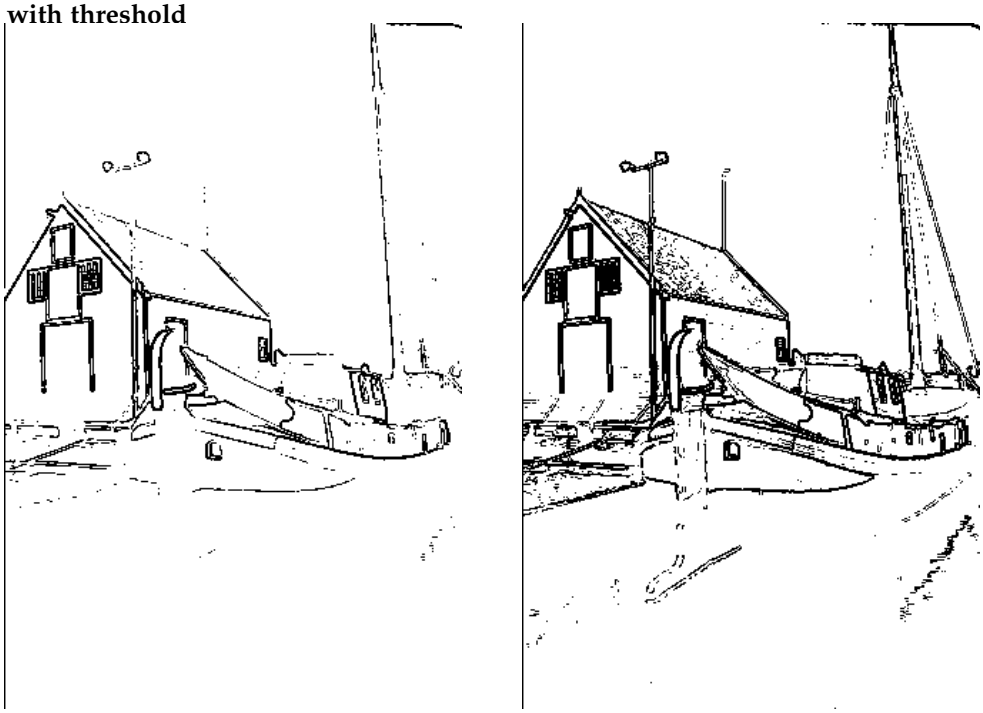


Figure 188: *CMF of magnitude response*

Result with threshold



(a) *Keep top 5% of the magnitude* **(b)** *Keep top 10 % of the magnitude*

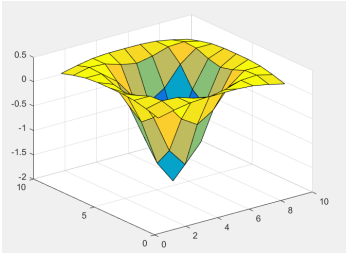
Figure 189: *Edge map*

II.2 Zero Crossing Detector

Kernel $G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$ and the second derivative of gaussian kernel is :
$\bigtriangledown^2 G(x,y) = \frac{1}{\pi\sigma^4}(\frac{x^2+y^2}{2\sigma^2} - 1)e^{-\frac{x^2+y^2}{2\sigma^2}}$
Discrete LoG kernel example, window size 9, $\sigma = 2$

0.193536	0.269968	0.329177	0.351527	0.353464	0.351527	0.329177	0.269968	0.193536
0.269968	0.346329	0.329062	0.226718	0.164957	0.226718	0.329062	0.346329	0.269968
0.329177	0.329062	0.0842225	-0.315103	-0.519105	-0.315103	0.0842225	0.329062	0.329177
0.351527	0.226718	-0.315103	-1.07781	-1.45199	-1.07781	-0.315103	0.226718	0.351527
0.353464	0.164957	-0.519105	-1.45199	-1.90521	-1.45199	-0.519105	0.164957	0.353464
0.351527	0.226718	-0.315103	-1.07781	-1.45199	-1.07781	-0.315103	0.226718	0.351527
0.329177	0.329062	0.0842225	-0.315103	-0.519105	-0.315103	0.0842225	0.329062	0.329177
0.269968	0.346329	0.329062	0.226718	0.164957	0.226718	0.329062	0.346329	0.269968
0.193536	0.269968	0.329177	0.351527	0.353464	0.351527	0.329177	0.269968	0.193536

(a) *Numerical Kernel,window size 9,* $\sigma = 2$



(b) *LoG kernel 3d plot,window size 9,* $\sigma = 2$

Result platform:Visual Studio 2017, Debug and Release under x64,language C++

LoG Response

(a) *Window Size 9,* $\sigma = 2$

(b) *Window Size 9,* $\sigma = 4$

(c) *Window Size9,* $\sigma = 8$

Figure 191: *LoG responses*

ternary map

if the result responses is $\underline{r}$,the threshold I choose is $\mathbb{E}[\underline{r}] \pm Var(\underline{r})/2$

mapping to gray level: $-1 \rightarrow 0$, $0 \rightarrow 127$, $1 \rightarrow 255$. After checking the histogram of the three response images above, the statisticas are:

$\mathbb{E}_{\sigma=2} = 140, Var_{\sigma=2} = 17.0$,

$\mathbb{E}_{\sigma=4} = 132, Var_{\sigma=4} = 16.1$,

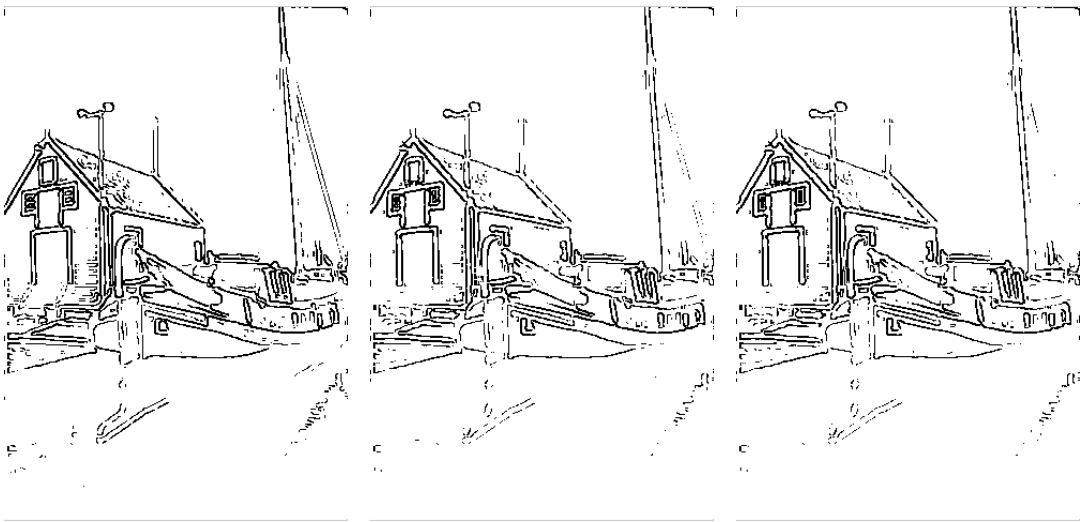
$\mathbb{E}_{\sigma=8} = 130, Var_{\sigma=8} = 15.8$

(a) *Ternary* $\sigma = 2$ **(b)** *Ternary* $\sigma = 4$ **(c)** *Ternary* $\sigma = 8$

Figure 192: *Ternary Map*

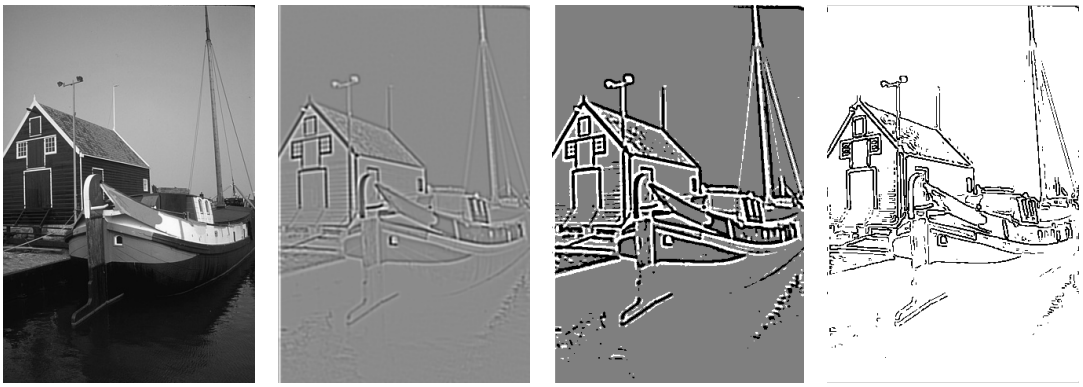
Zero Crossing(edge map)

For pixel at location i, if there is a zero crossing at this location, mark it as edge.



(a) *Edge Map* , $\sigma = 2$ **(b)** *Edge Map* , $\sigma = 4$ **(c)** *Edge Map* , $\sigma = 8$

Figure 193: *Edge Map*



(a) *Boat gray scale origin* **(b)** *LoG response* $\sigma = 2$ **(c)** *Ternary Map* $\sigma = 2$ **(d)** *Edge Map* $\sigma = 2$

Figure 194: *Routine of LoG edge detector*

Summary Results on noisy image

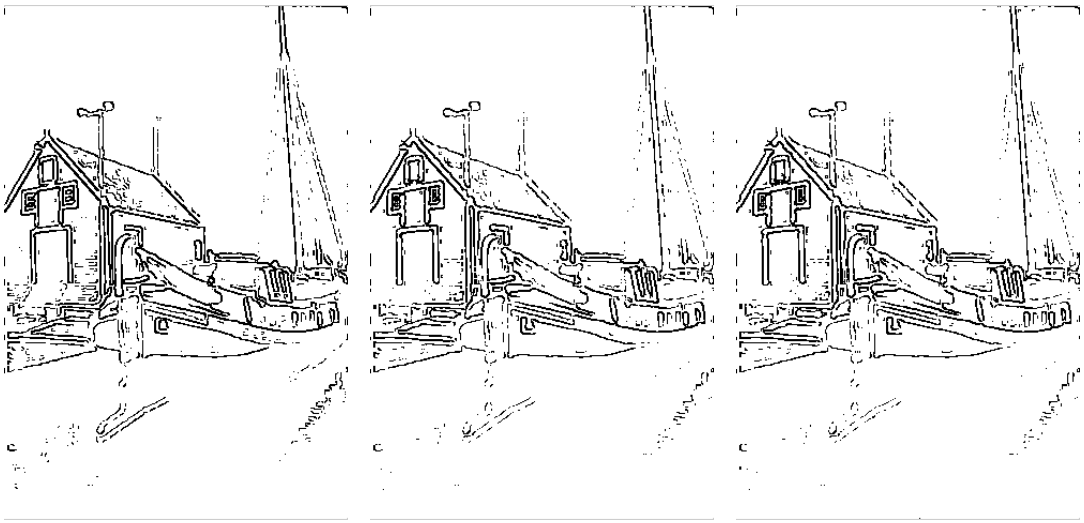
platform:Visual Studio 2017, Debug and Release under x64,language C++

(a) $\sigma = 2, mean = 138.661, Var = 16.810$ **(b)** $\sigma = 4, mean = 132.017, Var = 15.934$ **(c)** $\sigma = 8, mean = 130.677, Var = 15.821$

Figure 195: *LoG Responses*

(a) *ternary map*, $\sigma = 2$ **(b)** *ternary map*, $\sigma = 4$ **(c)** *ternary map*, $\sigma = 8$

Figure 196: *Ternery Map*



(a) *edge map*, $\sigma = 2$ **(b)** *edge map*, $\sigma = 4$ **(c)** *edge map*, $\sigma = 8$

Figure 197: *Edge Map*

Discussion According to the figures shown above we can find the following disadvantages of spatial filter edge detector. 1. for sobel detector, they are sensitive to noise, however, for LoG filter, because of the Gaussian filter, the noise can be suppressed. 2. for both sobel detector and LoG edge detector, they are sensitive to texture, as we can see from the roof of the house, sometimes such texture is not useful for image segmentation since we only need contour when performing segmentation job.

XXII. Structured Edge

I. Motivation And Machine Learning Tools

Traditional edge detectors compute gradients, and state of art edge detectors use multiple features, like brightness, color, texture and depth gradients computed over multiple scales. Visually salient edges correspond to a variety of visual phenomena, so it is hard to find an edge that meets everyones requirement. So now in structed edge, edges are structurely learned according to the segmentation dataset.

I.1 Decision Tree

1.Terminology

• a decision tree is a flow-chart-like structure.
• each **internal nodes** represents a test on an **attribute**.
• each branch represents outcome of a test.
• each leaf node represents a class label.
• **Splitting** is a process of dividing a node into two or more sub-nodes.

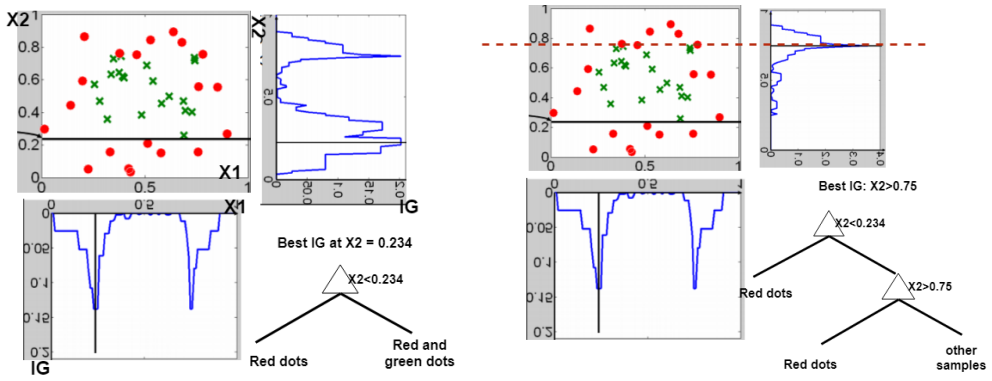
A decision tree is learned by keep splitting the tree node according to the samples at current tree node.

• **Pruning** is a process to remove sub-nodes of a decision node.

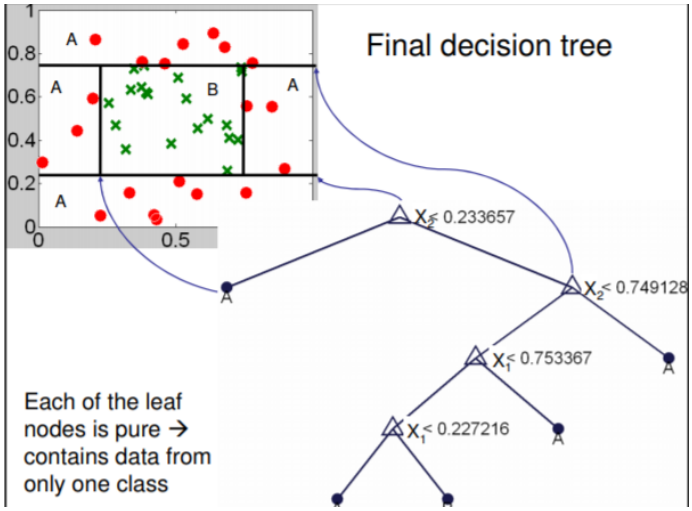
2.Details in splitting

There are many ways to decide where to split the tree. for example using **Gini Index, Chi-Square,Information Gain,Reduction in Variance**. It is easy to explain with the following example.(Here I use IG(information Gain to split the tree)).

Suppose we have samples with 2 features



(a) *First split of the tree* **(b)** *Second split of the tree*



(c) *final decision tree*

Figure 198: *0.40*

3.Why pruning

We use pruning to avoid overfit, We can set constrains on tree size, the number of trees and the depth of a tree.

I.2 Random Forest

To know what is a random forest, we should know what is tree bagging first. Tree bagging means generate a bag of trees. We can first generate different sample set by using some techniques like the bootstrapping. Then for each dataset, we can construct a decision tree. This process is called tree bagging.

Random Forest perform tree bagging, a lot of dataset are generated, the difference is , instead of develop a decision tree, it develop a 'random tree'. In the process of splitting, instead of find the best feature to split, the random forest randomly select some features from all features and pick the best feature from the selected features.

II. Structured Edge Detector

II.1 Overview

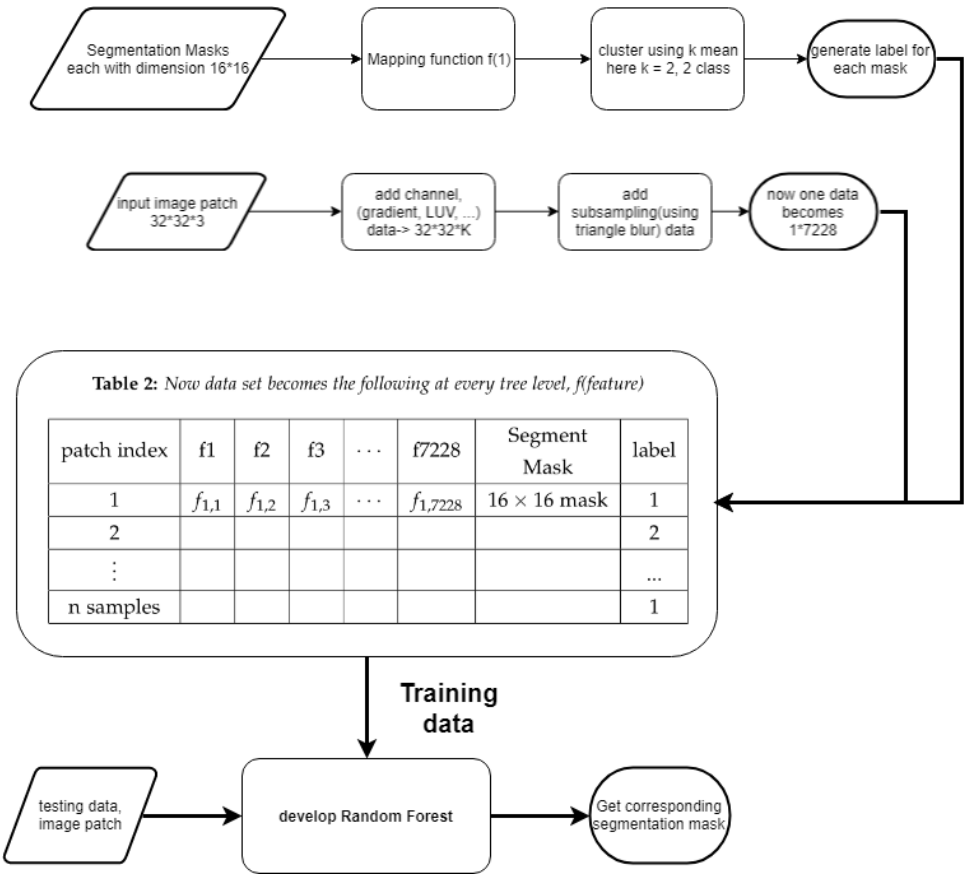


Figure 199: *flowChart of how structured edge detector works*

II.2 Details

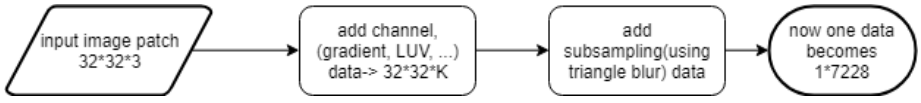


Figure 200: *processing on the input*

Preprocessing on Input training Data for the input image patch($R^{32\times32\times3}$)

we add information to transfer the data from $R^{32\times32\times3}$ to R^{7228}, we use 3 color channels, 2 magnitude channel, and 8 orientation channels. So now we have $32 \times 32 \times 13 features$. . Then we subsample each channel data with triangle blurring first. So now we have $32 \times 32 \times 13/4 = 3328$

features. We further downsample all channel images to 5×5 size image , and get $\begin{pmatrix} 25 \\ 2 \end{pmatrix} = 300$ every channel. So now each image patch we have $3328 + 13 * 300 = 7228$ features.

Preprocessing on the segmentation mask

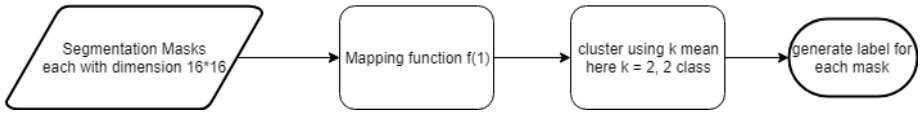


Figure 201: *processing on the segmatation mask*

For each 16×16 segmentation mask , we first create a long binary vector indicating whether each pair of segmentation index are same or not. For example if we have a 2×2 segmentation mask s is $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 & 2 \\ \hline \end{array}$, the corresponding vector would be $s[1] \neq s[2], s[1] = s[3], s[1] \neq s[4], s[2] \neq s[3], s[2] = s[4], s[3] \neq s[4]$, So the vector is 0,1,0,0,1,0. So the 16×16 segmentation mask, the vector would be of length $\begin{pmatrix} 256 \\ 2 \end{pmatrix} = 32640$ features, then we randomly choose 256 dimensions from it. For each of the segmentation mask we perform the above preprocessing operation, and after that we can perform k-means on the segmentation mask to be processed and label them with different labels.

Now at each splitting node, we have the dataset in the form as the following table shows. And we can use this label to split the tree.

Table 36: *Now data set becomes the following at every tree level, f(feature)*

patch index	f1	f2	f3	$\cdots$	f7228	Segment Mask	label
1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$\cdots$	$x_{1,7228}$	$16 \times 16 mask_1$	1
2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$\cdots$	$x_{2,7228}$	$16 \times 16 mask_2$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n samples	$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	$\cdots$	$x_{n,7228}$	$16 \times 16 mask_n$	1

Ensamble Model

With teh dataset at each tree node , it is easy for us to develop a random forest. After training the random forest, For every image patch, we can find the corresponding segmentation masks from different tree, and use that masks, we develop a probabilty to indicate whether there are edges or not.

II.3 Performance Analysis

Given the Ground truth and the generated edge map, we can develop the four terms: True Positive, True Negative, False Positive, False Negative. The definition can be shown in the following table.

Table 37: *Performance Analysis*

	edge pixels in Ground Truth	edge pixels in Edge Map	
True Positive	**Yes**	**Yes**	Precision: P = $\frac{TP}{TP+FP}$
True Negative	**No**	**No**	Recall: R = $\frac{TP}{TP+FN}$
False Positive	**No**	**Yes**	$F = \frac{2PR}{P+R}$
False Negative	**Yes**	**No**	

Precision shows how many selected items are relevant, and recall shows how many relavant items are selected. The higher F implies a better edge detector.

II.4 Result

Table 38: ***House Image, SE + SH****, thrs: Threshold, R: Recall, P: Precision, GT : Ground Truth, no NMS*

thrs		0.0500	0.1000	0.1500	0.2000	0.2500	0.3000	0.3500	0.4000	0.4500	0.5000	0.5500	0.6000	0.6500	0.7000	0.7500	0.8000	0.8500	0.9000	0.9500
GT1	R	0.8820	0.9541	0.9402	0.9309	0.9146	0.8600	0.7757	0.6856	0.5851	0.4823	0.4224	0.3492	0.2865	0.2435	0.2080	0.1685	0.1383	0.1011	0.0761
	P	0.1777	0.2600	0.3544	0.4638	0.5705	0.6511	0.7224	0.7712	0.8030	0.8113	0.8051	0.8089	0.8203	0.8380	0.8606	0.8923	0.9482	1.0000	1.0000
GT2	R	0.8370	0.8517	0.7974	0.7423	0.6959	0.6455	0.5786	0.5061	0.4357	0.3631	0.3227	0.2655	0.2148	0.1787	0.1487	0.1162	0.0897	0.0622	0.0468
	P	0.2741	0.3773	0.4887	0.6013	0.7057	0.7945	0.8761	0.9255	0.9721	0.9932	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
GT3	R	0.8872	0.8700	0.8103	0.7629	0.7240	0.6811	0.6177	0.5421	0.4661	0.3929	0.3500	0.2923	0.2420	0.2036	0.1701	0.1329	0.1026	0.0711	0.0536
	P	0.2540	0.3369	0.4342	0.5402	0.6419	0.7330	0.8176	0.8667	0.9091	0.9394	0.9480	0.9623	0.9850	0.9960	1.0000	1.0000	1.0000	1.0000	1.0000
GT4	R	0.8332	0.8533	0.7947	0.7329	0.6630	0.5773	0.4992	0.4398	0.3796	0.3190	0.2889	0.2462	0.2057	0.1752	0.1497	0.1212	0.0991	0.0727	0.0548
	P	0.2333	0.3231	0.4164	0.5075	0.5748	0.6076	0.6461	0.6876	0.7241	0.7458	0.7652	0.7927	0.8186	0.8380	0.8606	0.8923	0.9442	1.0000	1.0000
GT5	R	0.8503	0.8660	0.8093	0.7355	0.6421	0.5526	0.4771	0.4278	0.3754	0.3226	0.2959	0.2545	0.2108	0.1838	0.1563	0.1266	0.1039	0.0759	0.0572
	P	0.2280	0.3141	0.4061	0.4878	0.5332	0.5570	0.5915	0.6405	0.6858	0.7224	0.7508	0.7847	0.8037	0.8420	0.8606	0.8923	0.9482	1.0000	1.0000
mean	R	0.8579	0.8790	0.8304	0.7809	0.7279	0.6633	0.5897	0.5203	0.4484	0.3760	0.3360	0.2816	0.2320	0.1969	0.1665	0.1331	0.1067	0.0766	0.0577
	P	0.2334	0.3223	0.4200	0.5202	0.6052	0.6686	0.7307	0.7783	0.8188	0.8424	0.8538	0.8697	0.8855	0.9028	0.9163	0.9354	0.9681	1.0000	1.0000
F		0.3670	0.4717	0.5578	0.6244	0.6609	0.6660	0.6527	0.6237	0.5794	0.5199	0.4822	0.4254	0.3676	0.3233	0.2819	0.2330	0.1922	0.1423	0.1091

Table 39: ***Animal Image, SE + SH****, thrs: Threshold, R: Recall, P: Precision, GT : Ground Truth, no NMS*

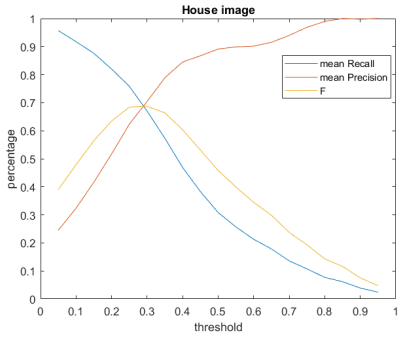
thrs		0.0500	0.1000	0.1500	0.2000	0.2500	0.3000	0.3500	0.4000	0.4500	0.5000	0.5500	0.6000	0.6500	0.7000	0.7500	0.8000	0.8500	0.9000	0.9500
GT1	R	0.7543	0.7519	0.6115	0.5003	0.3748	0.2522	0.1713	0.1362	0.1077	0.0803	0.0613	0.0488	0.0381	0.0256	0.0173	0.0065	0.0024	0	0
	P	0.2828	0.4854	0.6872	0.8048	0.8836	0.9298	0.9863	0.9957	1.0000	1.0000	0.9904	1.0000	0.9846	1.0000	1.0000	1.0000	1.0000	NaN	NaN
GT2	R	0.7074	0.6952	0.5418	0.4465	0.3242	0.2151	0.1422	0.1157	0.0923	0.0688	0.0530	0.0418	0.0331	0.0219	0.0148	0.0056	0.0020	0	0
	P	0.3096	0.5238	0.7106	0.8383	0.8920	0.9254	0.9555	0.9870	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN
GT3	R	0.4993	0.4842	0.4432	0.3734	0.3108	0.2194	0.1518	0.1237	0.0978	0.0698	0.0532	0.0374	0.0273	0.0129	0.0086	0.0043	0.0022	0	0
	P	0.1548	0.2584	0.4118	0.4967	0.6059	0.6689	0.7226	0.7478	0.7514	0.7185	0.7115	0.6341	0.5846	0.4186	0.4138	0.5455	0.7500	NaN	NaN
GT4	R	0.7944	0.7874	0.6191	0.4896	0.3532	0.2328	0.1540	0.1225	0.0959	0.0719	0.0554	0.0437	0.0346	0.0229	0.0155	0.0059	0.0021	0	0
	P	0.3326	0.5676	0.7767	0.8794	0.9299	0.9583	0.9897	1.0000	0.9945	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN
GT5	R	0.6792	0.6637	0.5546	0.4576	0.3402	0.2310	0.1579	0.1269	0.1003	0.0748	0.0576	0.0454	0.0360	0.0238	0.0161	0.0061	0.0022	0	0
	P	0.2735	0.4601	0.6691	0.7904	0.8612	0.9145	0.9760	0.9957	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN
mean	R	0.6869	0.6765	0.5540	0.4535	0.3406	0.2301	0.1554	0.1250	0.0988	0.0731	0.0561	0.0434	0.0338	0.0214	0.0144	0.0057	0.0022	0	0
	P	0.2707	0.4591	0.6511	0.7619	0.8345	0.8794	0.9260	0.9452	0.9492	0.9437	0.9404	0.9268	0.9138	0.8837	0.8828	0.9091	0.9500	NaN	NaN
F		0.3883	0.5470	0.5986	0.5686	0.4838	0.3648	0.2662	0.2208	0.1790	0.1357	0.1059	0.0830	0.0653	0.0419	0.0284	0.0113	0.0044	NaN	NaN

Table 40: ***House Image, SE + SH + MS****, thrs: Threshold, R: Recall, P: Precision, GT : Ground Truth, NMS*

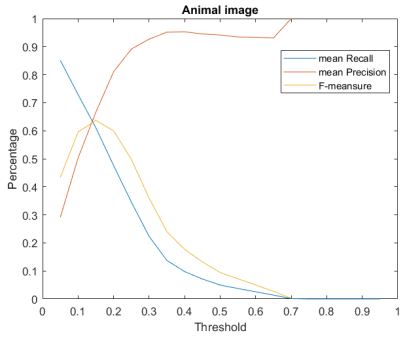
thrs		0.0500	0.1000	0.1500	0.2000	0.2500	0.3000	0.3500	0.4000	0.4500	0.5000	0.5500	0.6000	0.6500	0.7000	0.7500	0.8000	0.8500	0.9000	0.9500
GT1	R	0.9872	0.9802	0.9750	0.9651	0.9320	0.8542	0.7565	0.6171	0.4927	0.3870	0.3178	0.2632	0.2225	0.1720	0.1389	0.1011	0.0813	0.0517	0.0320
	P	0.1871	0.2569	0.3452	0.4557	0.5749	0.6749	0.7843	0.8369	0.8388	0.8409	0.8351	0.8373	0.8587	0.8970	0.9484	0.9831	1.0000	1.0000	1.0000
GT2	R	0.9546	0.8942	0.8267	0.7580	0.7026	0.6279	0.5436	0.4446	0.3588	0.2823	0.2341	0.1930	0.1594	0.1179	0.0901	0.0633	0.0500	0.0318	0.0197
	P	0.2941	0.3809	0.4758	0.5819	0.7047	0.8067	0.9163	0.9803	0.9931	0.9975	1.0000	0.9982	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
GT3	R	0.9599	0.8855	0.8307	0.7756	0.7363	0.6635	0.5818	0.4820	0.3945	0.3177	0.2661	0.2212	0.1823	0.1349	0.1030	0.0724	0.0572	0.0364	0.0225
	P	0.2585	0.3298	0.4180	0.5204	0.6455	0.7452	0.8572	0.9291	0.9545	0.9811	0.9939	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
GT4	R	0.9247	0.9072	0.8696	0.8085	0.7262	0.6191	0.5075	0.4143	0.3378	0.2751	0.2287	0.1894	0.1605	0.1242	0.0999	0.0727	0.0581	0.0372	0.0230
	P	0.2436	0.3304	0.4279	0.5306	0.6226	0.6800	0.7313	0.7809	0.7992	0.8308	0.8351	0.8373	0.8610	0.9000	0.9484	0.9831	0.9929	1.0000	1.0000
GT5	R	0.9590	0.9197	0.8773	0.7940	0.6940	0.5849	0.4749	0.3859	0.3313	0.2789	0.2379	0.1977	0.1676	0.1296	0.1043	0.0759	0.0611	0.0388	0.0240
	P	0.2419	0.3208	0.4135	0.4990	0.5699	0.6152	0.6554	0.6966	0.7507	0.8068	0.8321	0.8373	0.8610	0.9000	0.9484	0.9831	1.0000	1.0000	1.0000
mean	R	0.9571	0.9174	0.8759	0.8202	0.7582	0.6699	0.5729	0.4688	0.3830	0.3082	0.2569	0.2129	0.1785	0.1357	0.1072	0.0771	0.0616	0.0392	0.0242
	P	0.2450	0.3238	0.4161	0.5175	0.6235	0.7044	0.7889	0.8448	0.8673	0.8914	0.8992	0.9020	0.9161	0.9394	0.9690	0.9898	0.9986	1.0000	1.0000
F		0.3902	0.4786	0.5642	0.6346	0.6843	0.6867	0.6638	0.6030	0.5314	0.4580	0.3997	0.3445	0.2988	0.2372	0.1931	0.1430	0.1160	0.0754	0.0473

Table 41: ***Animal Image, SE + SH + MS****, thrs: Threshold, R: Recall, P: Precision, GT : Ground Truth, NMS*

thrs		0.0500	0.1000	0.1500	0.2000	0.2500	0.3000	0.3500	0.4000	0.4500	0.5000	0.5500	0.6000	0.6500	0.7000	0.7500	0.8000	0.8500	0.9000	0.9500
GT1	R	0.9322	0.8055	0.6871	0.5312	0.3795	0.2445	0.1493	0.1059	0.0779	0.0547	0.0416	0.0286	0.0155	0.0012	0	0	0	0	0
	P	0.3046	0.5291	0.7161	0.8678	0.9424	0.9786	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN	NaN	NaN	NaN
GT2	R	0.8547	0.7446	0.6101	0.4623	0.3303	0.2105	0.1274	0.0907	0.0668	0.0469	0.0357	0.0245	0.0133	0.0010	0	0	0	0	0
	P	0.3259	0.5709	0.7421	0.8814	0.9572	0.9833	0.9960	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN	NaN	NaN	NaN
GT3	R	0.6626	0.5353	0.4827	0.4094	0.3158	0.2122	0.1381	0.0978	0.0683	0.0468	0.0338	0.0230	0.0122	0.0014	0	0	0	0	0
	P	0.1790	0.2907	0.4160	0.5530	0.6484	0.7024	0.7649	0.7640	0.7252	0.7065	0.6714	0.6667	0.6538	1.0000	NaN	NaN	NaN	NaN	NaN
GT4	R	0.9531	0.8290	0.6734	0.5024	0.3548	0.2232	0.1337	0.0948	0.0698	0.0490	0.0373	0.0256	0.0139	0.0011	0	0	0	0	0
	P	0.3477	0.6081	0.7836	0.9164	0.9838	0.9976	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN	NaN	NaN	NaN
GT5	R	0.8526	0.7330	0.5994	0.4748	0.3479	0.2260	0.1391	0.0986	0.0726	0.0510	0.0388	0.0266	0.0144	0.0011	0	0	0	0	0
	P	0.2991	0.5170	0.6708	0.8328	0.9276	0.9714	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	NaN	NaN	NaN	NaN	NaN
mean	R	0.8511	0.7295	0.6106	0.4760	0.3457	0.2233	0.1375	0.0976	0.0711	0.0497	0.0374	0.0256	0.0138	0.0012	0	0	0	0	0
	P	0.2913	0.5032	0.6657	0.8103	0.8919	0.9267	0.9522	0.9528	0.9450	0.9413	0.9343	0.9333	0.9308	1.0000	NaN	NaN	NaN	NaN	NaN
F		0.4340	0.5955	0.6369	0.5997	0.4982	0.3599	0.2403	0.1770	0.1322	0.0944	0.0720	0.0499	0.0273	0.0023	NaN	NaN	NaN	NaN	NaN

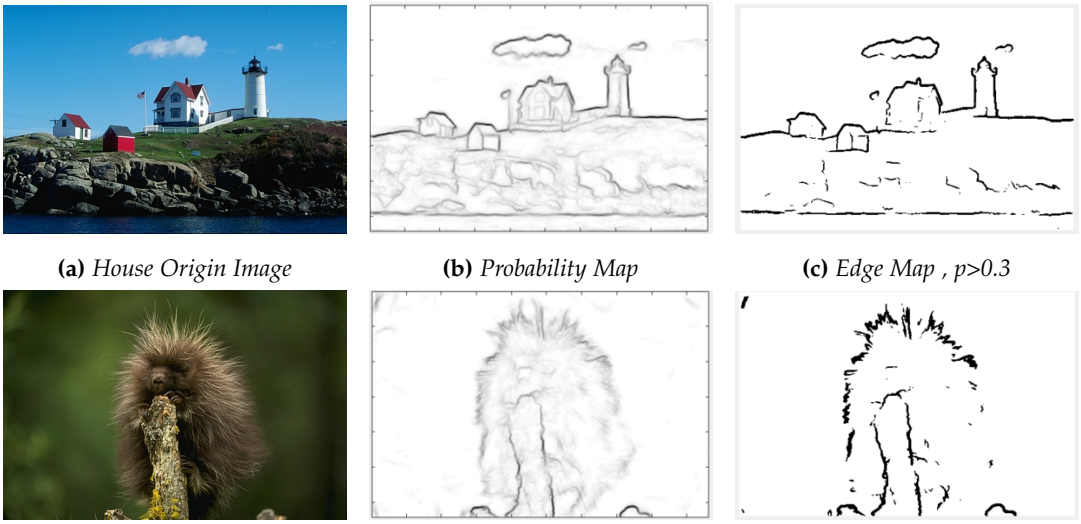


(a) *House data result*



(b) *Animal data result*

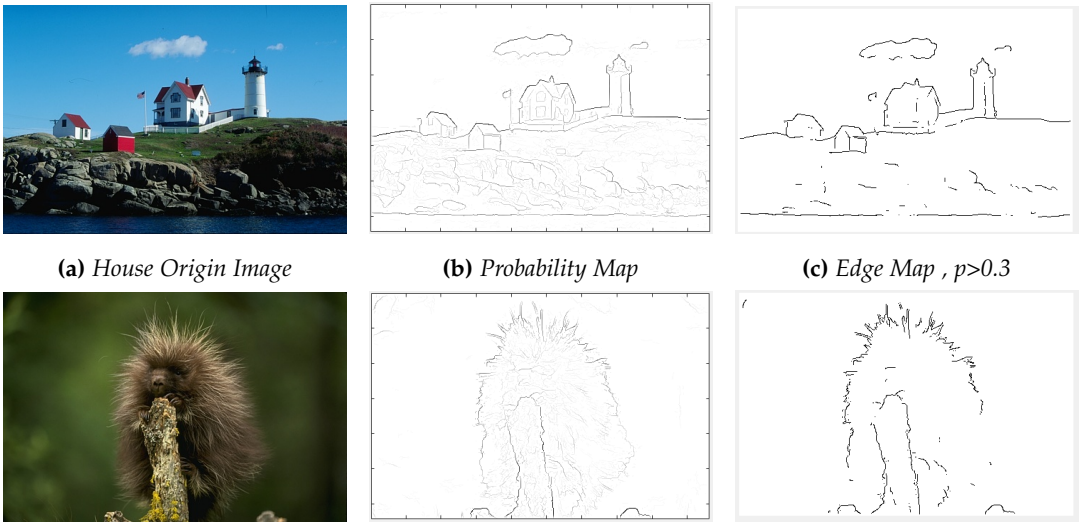
According to the table, for house image, the best threshold is **0.3** under both parameter settings, for animal image, the best threshold is **0.15** under both parameter settings, the following shows the probability map and edge maps under both settings.



(a) *House Origin Image* **(b)** *Probability Map* **(c)** *Edge Map , p>0.3*

(d) *Animal Origin Image* **(e)** *Probability Map* **(f)** *Edge Map, p>0.15*

Figure 203: *SE+SH*



(a) *House Origin Image* **(b)** *Probability Map* **(c)** *Edge Map , p>0.3*

(d) *Animal Origin Image* **(e)** *Probability Map* **(f)** *Edge Map, p>0.15*

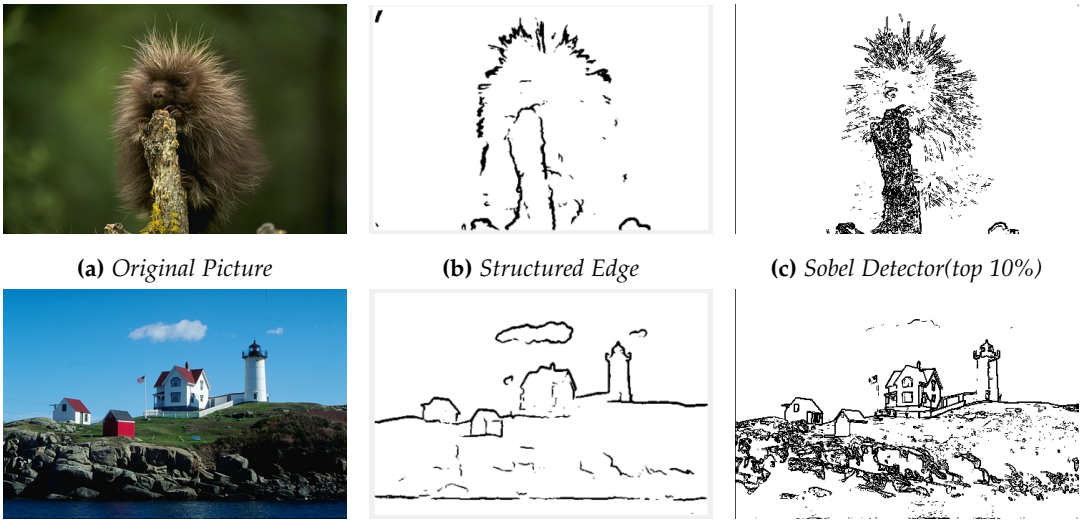
Figure 204: *SE+SH+MS+NMS*

II.5 Discussion

Which image is easier to get high F measure

According to the table above, **House** image is easier to get high F measure and the reason is obvious. The contour of the house is easier to observe, however, because the contour of the animal is formed with texture, it is hard to observe the contour.

Comparison between Sobel and Structured Edge



(a) *Original Picture* **(b)** *Structured Edge* **(c)** *Sobel Detector(top 10%)*

(d) *Original Picture* **(e)** *Structured Edge* **(f)** *Sobel Edge detector(top 10%)*

Figure 205: *Result of Sobel Detector and the Structured Edge*

Clearly the structured edge detector can effectly avoid the textures effect and try to find a better contour. This is good for image segmentation and structured edge detector is less sensitive to noise. If use the F-meansure, the F for Sobel detector for image animal is 0.3424, and the for house image, the F is 0.3631, therefore the structured edge detector is more satisfying.

XXIII. Salient Point Descriptors and Image Matching

I. Motivation and Background

One of the most important topic in computer vision is image matching. There are many operators to find feature points in an image.

operator 1: $(\bigtriangledown^2_{norm} L)^2$: Laplacian operator, with the previous gaussian operation, we can assume this the LoG operator.

operator 2: $(traceH_{norm}L)^2$: the trace of the hessian matrix and $trace(H_{norm}) = t(L_{xx} + L_{xy})$ (1 - normalized) [1]

operator 3: $(detH_{norm}L)^2$: the determinant of the hessian and $det(H_{norm}L) = t^2(L_{xx}L_{yy} - L^2_{xy})$ (1-normalized) [1]

operator 4: $\tilde{k} = L^2_y L_{xx} - 2L_x L_y L_x y + L^2_x L_{yy}$: junction detection and $\tilde{k}_{norm} = t^2\tilde{k}$ [1]

operator 5: DoG operator used in SIFT[2]: the difference of gaussian $L(\cdot; k\sigma) - L(\cdot; \sigma) = (G(\cdot; k\sigma) - G(\cdot; \sigma)) * I(\cdot)$ and the principle behind this method is $G(\cdot; k\sigma) - G(\cdot; \sigma) \approx (k\sigma - \sigma)\frac{\partial G}{\partial \sigma} = (k-1)\sigma \cdot \sigma \bigtriangledown^2 G = (k-1)\sigma^2 \bigtriangledown^2 G$ and the $\sigma^2 \bigtriangledown^2 G$ is the LoG operator [2]

II. Summary of SIFT and SURF

II.1 SIFT

Step 1 Use *DoG* to approximate the *LoG*, for one octave, first generate several images with gaussian filter (here comes first parameterσ), then get the substraction between two gaussian filtered images to get DoG.

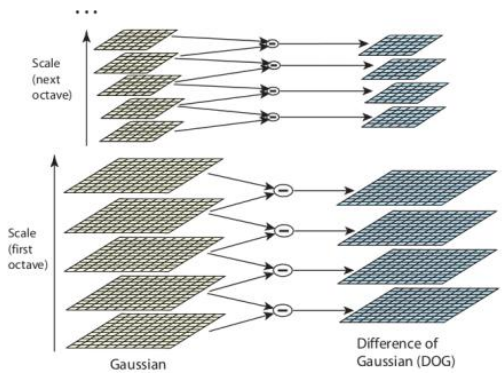


Figure 206: *Difference of Gaussian*

When the DoG image set is formed , we can find the locak extreme with the neighbour 26 pixels' values,

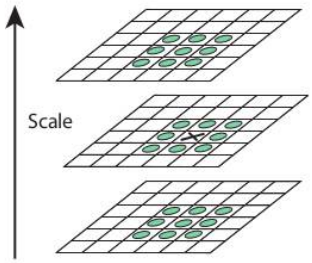


Figure 207: *Find local extrema*

The above operation is performed at each octave(here comes second parameter **octave number**), and we can find several local extrema.

Step 2 Localize each keypoints, because each keypoint is find at different octave, and we have to decide the precise location of an extrema, since the above is in discrete model and we can form a continuous function and find a precise location. Moreover, if some extrema are small, we ignore the extrema (here comes third parameter**threshold for contrast**) , also we use Hessian matrix to filter out edge keypoints(here comes fourth parameter **threshold for edge**).

Step 3 Assign Orientation, an orientation histogram with 36 bins covering 360 degrees is created, the highest peak in the histogram is taken.

Step 4 Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

II.2 SURF

Step 1 Approximate LoG , SURF goes a little further and approximates LoG with Box Filter. Below image shows a demonstration of such an approximation. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location.

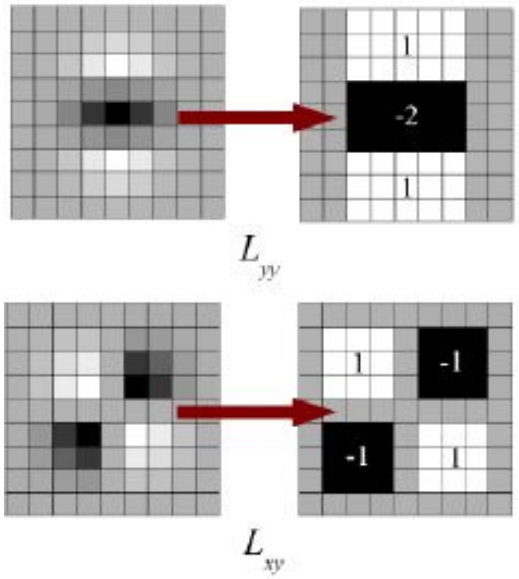


Figure 208: *box filter for approximate LoG*

Step 2 Localization is the same as SIFT. **Step 3** Orientation assignment is a bit different: SURF uses wavelet responses in horizontal and vertical direction for a neighbourhood of size 6s. Adequate guassian weights are also applied to it. Then they are plotted in a space as given in below image. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. Interesting thing is that, wavelet response can be found out using integral images very easily at any scale. For many applications, rotation invariance is not required, so no need of finding this orientation, which speeds up the process. It improves speed and is robust upto $\pm 15^{\circ}$.

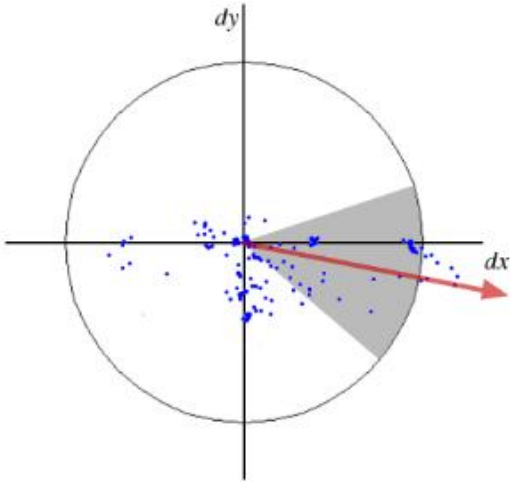


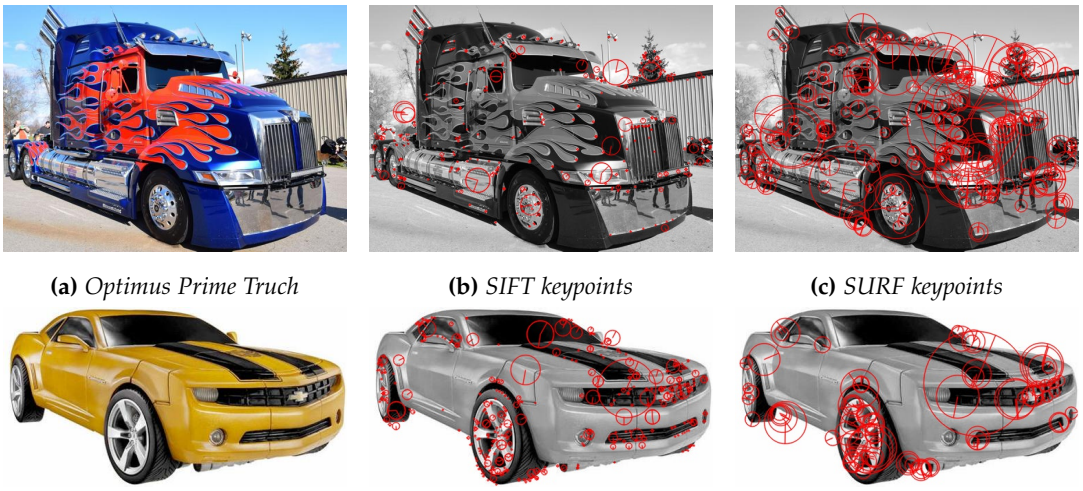
Figure 209: *Orientation Assignment*

III. Result

platform:Visual Studio 2017, Debug and Release under x64,language C++,API: OpenCV 3.4.0

SIFT Parameter: top 400 keypoints, octave layers = 4, contrast threshold 0.04, edge threshold 20, $\sigma = 1.6$.

SURF Parameter: hessian threshold 7000, octave layers = 3. octave = 4



(a) *Optimus Prime Truch* **(b)** *SIFT keypoints* **(c)** *SURF keypoints*

(d) *Bumblebee Car* **(e)** *SIFT Keypoints* **(f)** *SURF Keypoints*

Figure 210: *Keypoints Detection,SIFT Parameter: top 400 keypoints, octave layers = 4, contrast threshold 0.04, edge threshold 20, $\sigma = 1.6$. SURF Parameter: hessian threshold 7000, octave layers = 3. octave = 4*

III.1 Discussion

If we set the default parameter, the SIFT detector took 0.3415s and the SURF detector took 0.2322s to gather all the keypoints. This is still not efficient enough for realtime operation. According to the truck figure, we can see that the SIFT found all the sharp corners in different scale, like the corner of waterdrop shape toppings on the surface, or the end of air intake grid, also some blob like the holes on the tire. For the SURF detector, we can see that a lot of blobs are detected, even the whole tire are detected. There are similarityies between SIFT detector and SURF detector, like the blobs, and there are also some differences, as we can see from the bumbleebee car, the corners of the mirror are found, more sharp corner are more eaisly found with lower octave. But corners are not found with the setting above in SURF keypoints detector.

IV. Image Matching

SIFT and SURF both provide keypoints descriptors as described in the introduction, we can use the features to mathc the key points between two images.

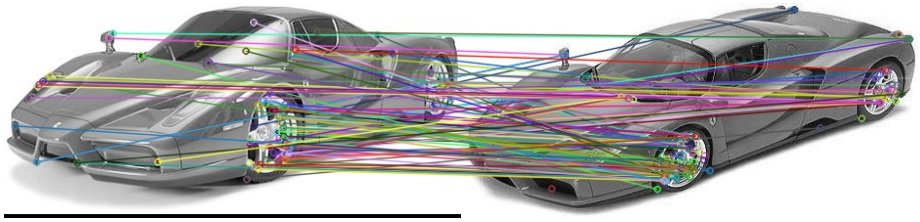
Experiment Apply the image matching method to the two ferrari car images.

SIFT Parameter: top 100 keypoints, octave layers = 4, contrast threshold 0.04, edge threshold 20, $\sigma = 1.6$.

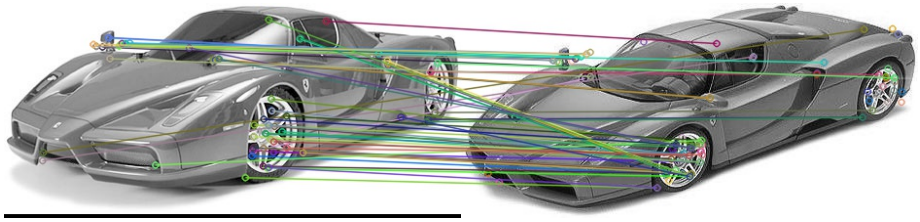
SURF Parameter: hessian threshold 5000, octave = 4, octave layers = 3.

IV.1 Result

platform:Visual Studio 2017, Debug and Release under x64,language C++,API: OpenCV 3.4.0

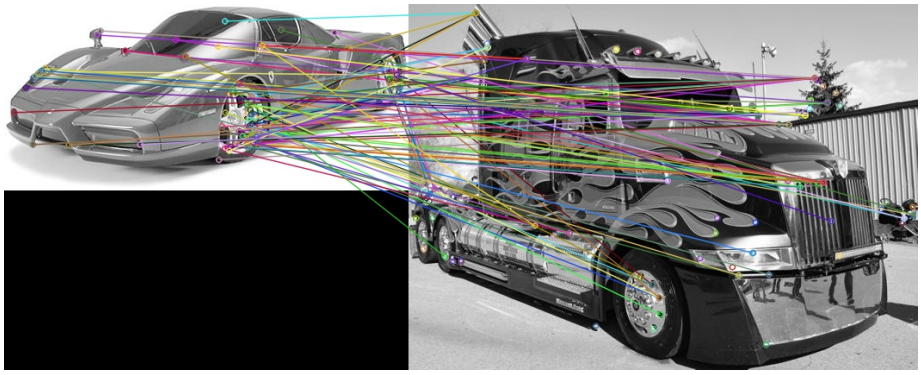


(a) *SIFT matching*

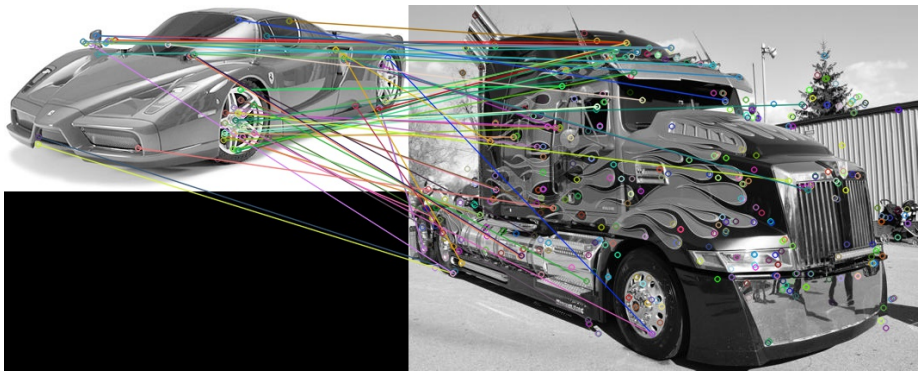


(b) *SURF matching*

Figure 211: *Matching between two ferrari images*

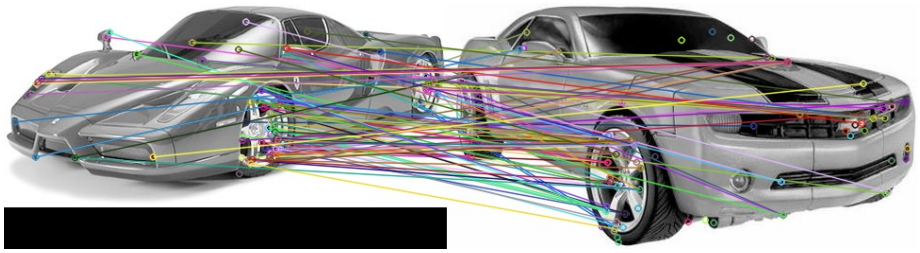


(a) *SIFT matching*

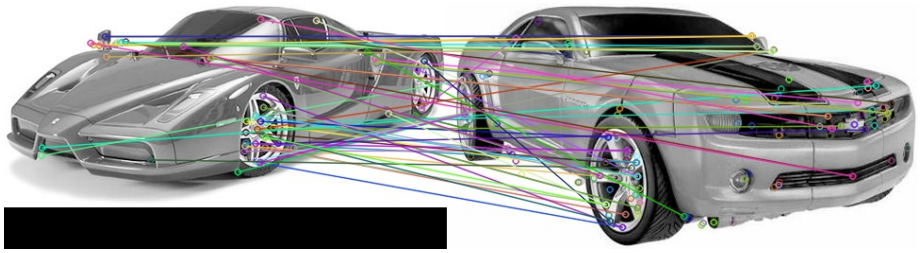


(b) *SURF matching*

Figure 212: *Matching between Ferrari and Optimus Prime Truck*



(a) *SIFT matching*



(b) *SURF matching*

Figure 213: *Matching between Ferrari and Bumblebee Car*

IV.2 Discussion

For three images, there are a lot of mismatches, for the first image, there are also some matches like the tires of both ferrari images, and the tires of ferrari image and the bumblebee car images. For the failure cases, since both image and objects are different, if for the task of car matching, the low ratio of matching may be satisfying. However, for the task that whether these two objects of the images is the same class, such mismatching may be not satisfying. Look at the bumblee car and the ferrari, both have keypoints on the mirror, but they did not match, and clearly the reason is the luminance, the mirror of the bumblee car is darker while the luminance of the ferrari car is bright there is large reflecition. Therefore, matching may not work well between different objects and against the same object but with a large viewing angle difference. But for part of the object with small viewing angle difference, the SIFT and SURF matching can still perform well, like the tire and the rearview mirror.

V. Bag Of Words

Bag of words can be used for image recognition, in this problem we use bag of words to match the test image to some code.

Routine

Step 1: We extract SIFT keypoints from the training images, and gather all the training keypoints descriptors into the dataset. Suppose we have generated n keypoints, then we have a dataset with dimension $n \times 128$.

Step 2: Perform k-means on the dataset, in this problem we choose k = 8, that is we cluster all the

data points into 8 classes.

Step 3: After get the 8 centroids, for each image, we cluster each keypoint to the label class, and get a histogram of the frequency of keypoints in each class. For example, we have 100 keypoints in image1 and 10 keypoints in class1, 20 in class 2,.....10 keypoints in class 8. We have the code for that image is (0.1,0.2,...0.1). So now for images Ferrari, Optimus Truck and Bumblebee, we have three code each with dimension 8.

Step 4: For test image, we generate the code, and matches it to the three code, to recognize, what the test image is.

V.1 Result

platform:Visual Studio 2017, Debug and Release under x64,language C++,API: OpenCV 3.4.0

```
the codebook is:
      0.118         0.16        0.162         0.07        0.118        0.168         0.08        0.124
  0.0518962     0.163673    0.0299401     0.165669     0.151697    0.0578842     0.141717    0.237525
       0.06          0.1         0.23         0.05        0.106        0.238        0.106         0.11
the code for test image is:
  0.0931818     0.152273     0.172727    0.0727273    0.0977273     0.218182          0.1    0.0931818
the result is :
it is a Ferrari
```

Figure 214: *Bag of Words Experiment*

V.2 Discussion

According to experiment above, bag of words can be used for object recognition. But we should also pay attention to the training dataset and the classification method. In this case, we have training samples that suit the testing case, what above a ferrari that dressed like the bumblebee car, we have no experiment data and therefore I can not tell the performance. And when it comes to the classification, we use kmeans to cluster the keypoints descriptors and I think first, classified into 8 features is not enough, we can try 16 or 24 features, secondy, I think we can use other classification method like the EM algorithm(kmeans is one kind of EM algorithm), because one can anlyse the probability distribution for one descriptor, each histogram can be seem as a density function, and with all the density functions we can use EM to perform the classification to avoid overfit.

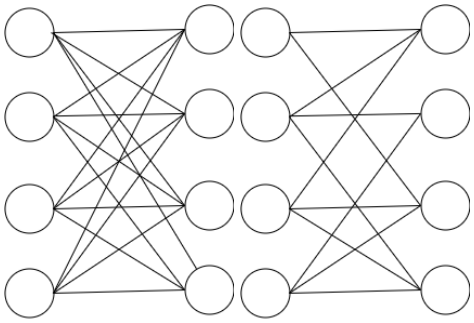
XXIV. CNN Training and Its Application to the MINST Dataset

I. CNN architecture and Training(a)

I.1 CNN Components

Fully connected Layer Fully connected layers connect ecery neuron in one layer to every neu-

ron in another layer, Example below show this concept:



(a) *fully connected* **(b)** *not fully connected*

Figure 215: *Example of fully connected layer*

Convolutional layer Convolutional layer apply 'convoltuion' operations with different filters. The network tunes the parameters of these filters. We actually perform the inner product operation when some filter overlay some image block. Belows shows an example:

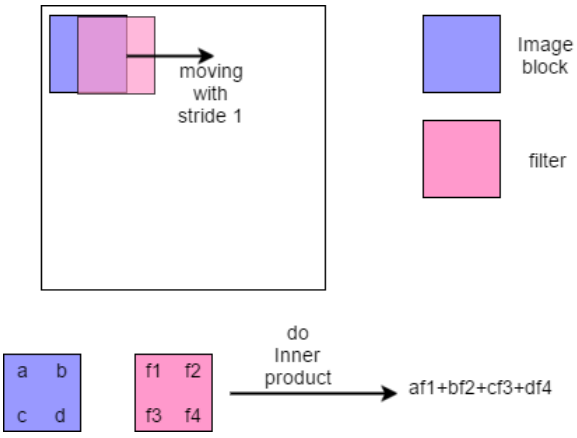


Figure 216: *Convolutional Layer*

another example:

$$Imageblock = \begin{bmatrix} 123 & 234 & 25 \\ 59 & 89 & 163 \\ 23 & 26 & 254 \end{bmatrix}$$

$$filter = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix} \quad result = \begin{bmatrix} 147 & 57 \\ 56 & 95 \end{bmatrix}$$

max pooling layer

The max pooling layer outputs the max element in each block.The following figure shows the process:

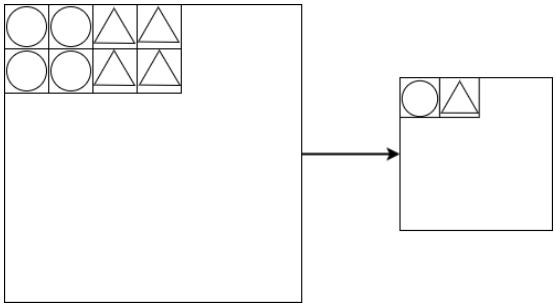


Figure 217: *Max polling layer*

The activation function First is the structure of one single neuron:

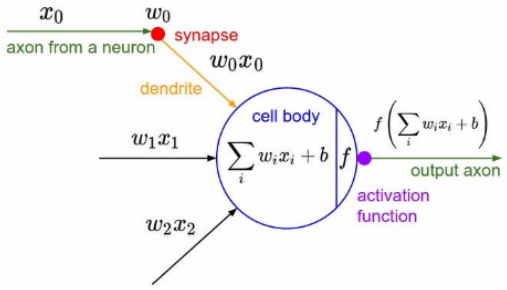


Figure 218: *The structure of one neuron*

The activation function deals with the weighted sum from the synapse. There are many different types of the activation function. The most frequently used activation functions are $tanh$,$ReLU(RectifiedLinearUnit)$,$Logistic$ etc.

the softmax function

the softmax function, or normalized exponential function, is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range [0, 1] that add up to 1. The function is given by $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} for j = 1,,K$. Here for the classification project we usually use softmax function to represent how much similar the object is belong to the class.

I.2 Over-fitting

Over-fitting happens when the trained network fit the training network so well while leading to the inefficiency to the testing data. For example, a model may learn the detail and the noise in the training data to the extent that it negatively impacts the performance of the model on new data. To overcome overfitting, in CNN, we can reduce the number fo filters, or assign a reasonable ratio of the number of training image to number of the testing image. Also, we set a parameter of dropout to drop some element in the network while traing. For exmaple in tensorflow we use arguments like keep_prob to denote the probability that each element is kept.

I.3 Why CNN

Instead of traditional feature extraction kernels, CNN uses the network to keep tring to find the best suitable feature extration kernel. In the image classification project, because CNN did well in feature extraction, therefore it works much better than traditional methods. The handcraft feature extraction kernel has too much experience from human's subjective thought, those filter parameters are most time fixed. However the CNN make the feature extraction one important step of the whole system, that the kernel are adapted according to the classification result and most of the time the extraction features are strange and can not be simply found by mathematical principle or human experience.

I.4 Loss Function and Backpropagation

First, we identify this is a classification problem, and the last stage is the softmax layer. The softmax function is given by
$\sigma : \mathbb{R}^K \to (0,1)^K$
$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} for \;\; j = 1,,K.$
The ideal output is **one-hot vector**, lets denote the label for sample i as $\underline{l_i}$ and the classification result is $\underline{a_i}$, therefore, the loss function can be defined.
There are many ways to define a loss function, first well-known is the **mean square error**. $MSE = \frac{1}{n}\sum_{i=1}^{n}(\underline{a_i} - \underline{l_i})^2$,here the square operation means the inner product of two vectors. The other is the **cross entropy**, the formula is $H_{y'}(y) := -\sum_i y_i' \log(y_i)$Where $y_i^{'}$ is the ground truth label of ith training instance and y_i is the prediction result of your classifier for the ith training instance.
Now consider the loss function as E, given a training input, we know the result is a funciton of weight vectors and convolution layer feature vector, we denote all of them as $\underline{\omega}$, therefore the loss function can be expressed as $E(\underline{\omega})$, the above process is called **propagation**. Then we can use the loss function to learn all the parameters with **Backpropagation**. We compute $\frac{\partial E}{\partial \omega_i}$.

Consider an input feature map of dimension $H \times W$ and the weight kernel of dimension $k_1 \times k_2$, we have the output feature map of size $(H - k_1 + 1)$ by $(W - k_2 + 1)$. And the partial learning step is $\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}}$, for all tuning parameters we perform the same process, first analyse the structure, then perform partial derivative. Now we can update the weight, first the weight's output delta and input activation are multiplied to find the gradient of the weight. And the ratio of the weight's gradient is substracted from the weight. $\omega_{ij_{new}} = \omega ij - \eta \Delta w_{ij} = \omega ij - \eta \frac{\partial E}{\partial w_{ij}}$, here η is the learning rate which measures how much step should a weight vector change each updating time. The above shows the rule of gradient decsent, there are many other ways to perform the learning optimization algorithm. In my report, I choose between GradientDescentOptimizer and AdamOptimizer(Adaptive moment).Adaptive Moment Estimation is another method that computes adaptive learning rates for each parameter. Here the moments means the first and second moment, we also compute the decaying average of past and past squared gradients m_t and v_t respectively as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{1}$$

and the update rule is :
$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$
So to conclude, the parameters we can tune are the learning rate, the optimize method, the filter dimension of different layer, the depth of the network, the initial weight, the decay rate of last hidden layer and the training image amount in each epoch.However in the following sections, I would vary the parameters of learning rate, decay rate and the architecture of the whole network to try to obtain a better result.

II. Train LeNet-5 on MNIST Dataset(b)

II.1 Architecture of LeNet-5

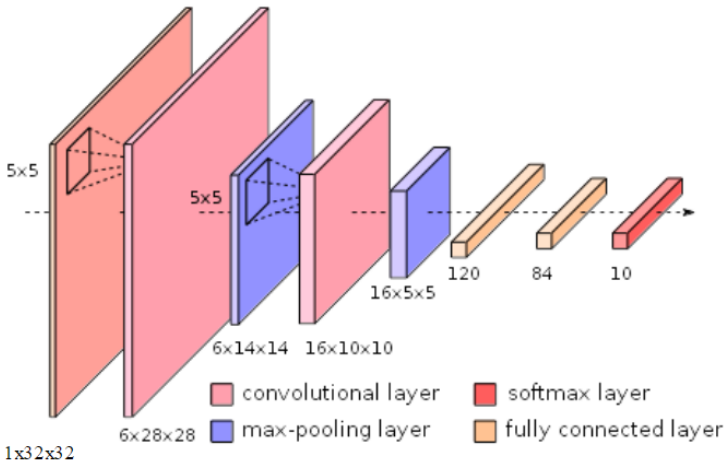


Figure 219: *LeNet-5 Architecture*

The table below more precisely shows the parameters.

Table 42: *My caption*

	layer in($w \times h \times channel$)	**filter**($w \times h \times channel \times amount$)	**layer out**($w \times h \times channel$)
1	$32 \times 32 \times 1$	$5 \times 5 \times 1 \times 6$	$28 \times 28 \times 6$
2	$28 \times 28 \times 6$	$2 \times 2 \times 1 \times 1$	$14 \times 14 \times 6$
3	$14 \times 14 \times 6$	$5 \times 5 \times 6 \times 16$	$10 \times 10 \times 16$
4	$10 \times 10 \times 16$	$2 \times 2 \times 1 \times 1$	$5 \times 5 \times 16$
5	$5 \times 5 \times 16$	$5 \times 5 \times 16 \times 120$	$1 \times 1 \times 120$
6	$1 \times 1 \times 120$	$1 \times 1 \times 120 \times 84$	$1 \times 1 \times 84$
7	$1 \times 1 \times 84$	$1 \times 1 \times 84 \times 10$	$1 \times 1 \times 10$
8	$1 \times 1 \times 10$	softmax	$1 \times 1 \times 10$

II.2 Parameters Initialization

There are many parameters to be tuned, like mentioned in the theory above, we can tune learing rate, the dropout(decay) rate, the initial filters,etc.

weight Initialization We use the Lenet-5 as the basic structure, therefore the first convolutional layer we use 6 filters, I choose the following 6 filters as the initial filter:(0:black, 1:white)

Figure 220: *6 initial filters*

For the next $5 \times 5 \times 6$ filters, we randomly generate numbers and assign them as the filter weights.Because it is hard for human to image, or to visualize the high dimension filters. The random method is using the gaussian distribution with mean 0 and dev 0.1.

learning rate learning rate changes how much steps should go along the deviation, a large learning rate would greatly affect the performance since the weight change does not dig deeper and sometimes we may miss the local optimum. However a very small learning rate affects the learning time, that it took a long convergence to find the optimal. For the following experiments I choose 5 different learning rates : 0.1, 0.05, 0.01, 0.005, 0.001.

decay To avoid overfitting, use drop out for some layer we want, we use a parameter called decay to mean the probability that a neuron's output is dropped. For the LeNet-5 training, I specify a dropout layer before the last layer, and the decay rate I choose is 0,0.1,0.2,0.3,0.4.

Optimization Method As mentioned above, we have **First Order Optimization Algorithms**.These algorithms minimize or maximize a Loss function E(x) using its Gradient values with respect to the parameters. Most widely used First order optimization algorithm is Gradient Descent.The First order derivative tells us whether the function is decreasing or increasing at a particular point. First order Derivative basically give us a line which is **Tangential** to a point on its Error Surface.The high variance oscillations in SGD makes it hard to reach convergence , so a technique called **Momentum** was invented which accelerates SGD by navigating along the relevant direction and softens the oscillations in irrelevant directions. For the experiments all through this report, the optimizer to use is Adaptive moment estimator.

II.3 Experiments

Weight initialization and Optimization Method keeps the same for all the experiments, the parameters to tune are learning rate and decay, in total 20 experiments are performed. The epoch number is 10000, at each epoch, the batch size is 60 which means 60 images are used at each epoch.

platform:Pycharm Community, Compiler: Python 3.6.1, Tensorflow-GPU with GTX 1060

Table 43: *Experiment results*

epoch:10000 , batch size:60 , Adam Optimizer,Lenet5 structure			
learning rate	**decay**	**training accuracy**	**testing accuracy**
0.001	**1.0**	**1.0**	**0.991**
0.005	**1.0**	**0.993**	**0.987**
0.01	**1.0**	**0.982**	**0.978**
0.05	**1.0**	**0.151**	**0.11**
0.1	**1.0**	**0.134**	**0.105**
0.001	**1.0**	**1.0**	**0.991**
0.001	**0.9**	**1.0**	**0.9899**
0.001	**0.8**	**0.993**	**0.9897**
0.001	**0.7**	**1.0**	**0.9873**
0.001	**0.6**	**1.0**	**0.9898**

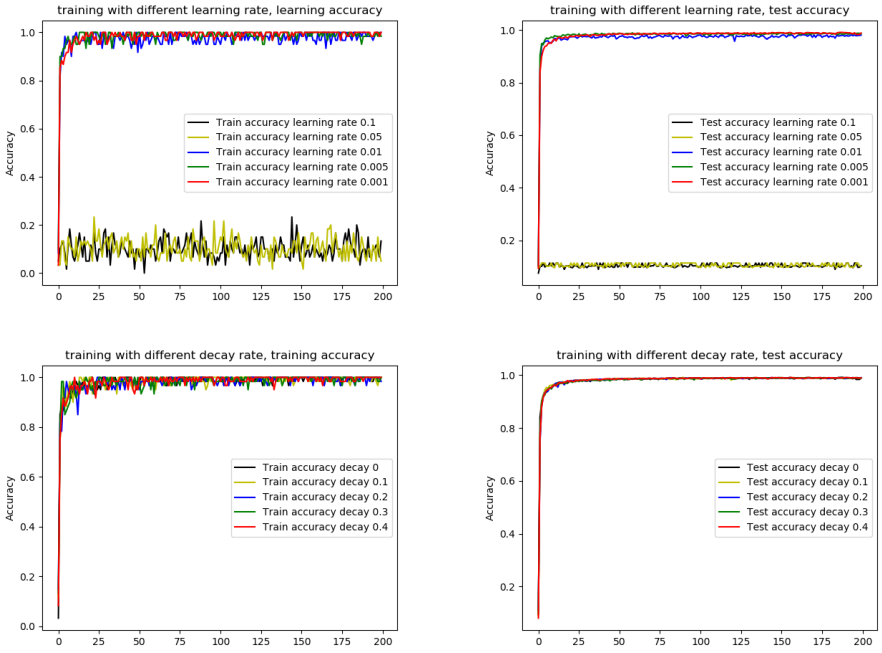


Figure 221: *Performance on different settings*

I also extract the trained 6 5×5 filters in first convolution layer and 16 $5 \times 5 \times 6$ filters in second convolution layer. To visualize the $5 \times 5 \times 6$ filters, I only takes the first 3 channel , and make it an RGB image to try to get a taste of the filter pattern.

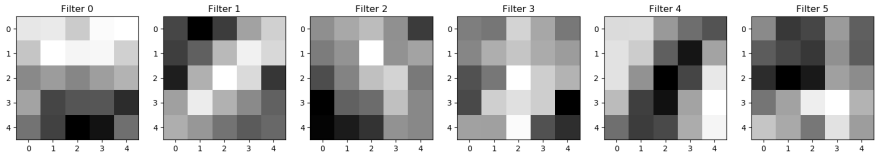


Figure 222: *trained filters in first convolution layer*

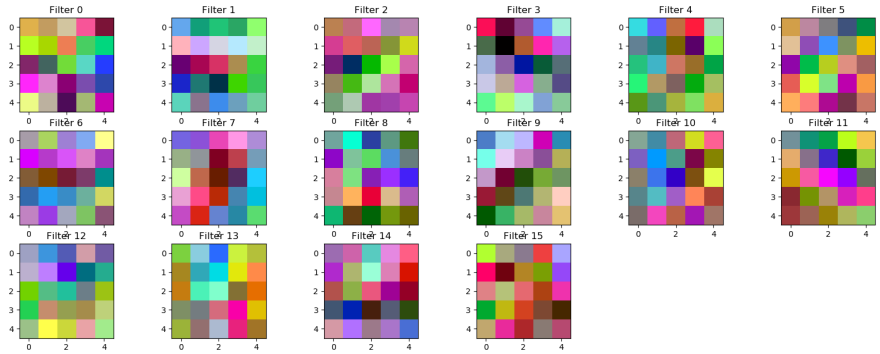


Figure 223: *trained filters in second convolution layer*

II.4 Best parameter for LeNet-5 and the experiment result

According to the experiments above, I choose learning rate to be 0.0001, the decay rate to be 0.1, and the epoch munber to be 15000, each epoch the batch size is 60. **Best Accuracy: 0.989**

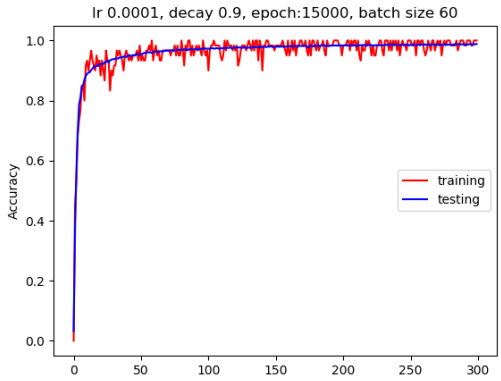


Figure 224: *performance curve, final testing accuracy:0.989*

III. Improve the LeNet-5 for MINST dataset

III.1 Setting 1

First I give more convolution filters in the first and second layer, and for the MLP, I used 5 hidden layers which make the network deep.

Setting:

Table 44: *Setting 1*

	layer in($w \times h \times channel$)	**filter**($w \times h \times channel \times amount$)	**layer out**($w \times h \times channel$)
1 Conv	$32 \times 32 \times 1$	$5 \times 5 \times 1 \times 32$	$28 \times 28 \times 32$
2 Pool	$28 \times 28 \times 32$	$2 \times 2 \times 1 \times 1$	$14 \times 14 \times 32$
3 Conv	$14 \times 14 \times 32$	$5 \times 5 \times 32 \times 64$	$10 \times 10 \times 64$
4 Pool	$10 \times 10 \times 64$	$2 \times 2 \times 1 \times 1$	$5 \times 5 \times 64$
5 Conv	$5 \times 5 \times 64$	$5 \times 5 \times 64 \times 1024$	1×1024
6 FC	1×1024	1024×512	1×512
7 FC	1×512	512×256	1×256
8 FC	1×256	256×128	1×128
9 FC	1×128	128×64	1×64
10 FC	1×64	64×10	1×10
11	$1 \times 1 \times 10$	softmax	$1 \times 1 \times 10$

learning rate:1×10^{-4}**, Optimization Method: Adaptive Moment Estimate, epoch number: 20000, batch size each epoch: 60, decay : 0.5, Activate function ReLU for all conv and all fc layer**
The above settings gives a result of testing accuracy: 0.993

```
step 19950, training accuracy 1
2018-04-16 14:26:43.771663: W C:\tf_jenkins\workspace\rel-win\M\windows-gpu\PY\36\1
2018-04-16 14:26:43.951918: W C:\tf_jenkins\workspace\rel-win\M\windows-gpu\PY\36\1
test accuracy 0.9929
```

Figure 225: *Best Accuracy I can get*

III.2 Setting 2

Next is the setting of VGG, since the VGG Net is performed at first to deal with the ILSVRC, it has more layers, but I still migrate it to the MNIST dataset with dimension 28×28 images. VGGNet consists of 16 convolutional layers and is very appealing because of its very uniform architecture. It only performs 3×3 convolutions and 2×2 pooling all the way through. For our image, we can at most do 3 pooling, so architecture is in the following table, $Conv'$ means convolution without padding zero.

Table 45: *Setting 2,VGG-like*

	layer in($w \times h \times channel$)	**filter**($w \times h \times channel \times amount$)	**layer out**($w \times h \times channel$)
1 Conv	$28 \times 28 \times 1$	$3 \times 3 \times 1 \times 32$	$28 \times 28 \times 32$
2 Conv	$28 \times 28 \times 32$	$3 \times 3 \times 32 \times 32$	$28 \times 28 \times 32$
3 Pool	$28 \times 28 \times 32$	$2 \times 2 \times 1 \times 1$	$14 \times 14 \times 32$
4 Conv	$14 \times 14 \times 32$	$3 \times 3 \times 32 \times 64$	$14 \times 14 \times 64$
5 Conv	$14 \times 14 \times 64$	$3 \times 3 \times 64 \times 64$	$14 \times 14 \times 64$
6 Pool	$14 \times 14 \times 64$	$2 \times 2 \times 1 \times 1$	$7 \times 7 \times 64$
7 Conv'	$7 \times 7 \times 64$	$3 \times 3 \times 64 \times 128$	$5 \times 5 \times 128$
8 FC	$5 \times 5 \times 128$	$5 \times 5 \times 128 \times 1024$	1×1024
9 FC	1×1024	1024×512	1×512
10 FC	1×512	512×256	1×256
11 FC	1×256	256×128	1×128
12 FC	1×128	128×64	1×64
13 FC	1×64	64×10	1×10
14	$1 \times 1 \times 10$	softmax	$1 \times 1 \times 10$

learning rate:1×10^{-4}**, Optimization Method: Adaptive Moment Estimate, epoch number: 25000, batch size each epoch: 60, decay : 0.6, Activate function ReLU for all conv and all fc layer,**
The above settings gives a result of testing accuracy: 0.9953

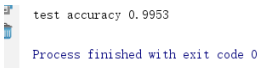


Figure 226: *VGG final test performance*

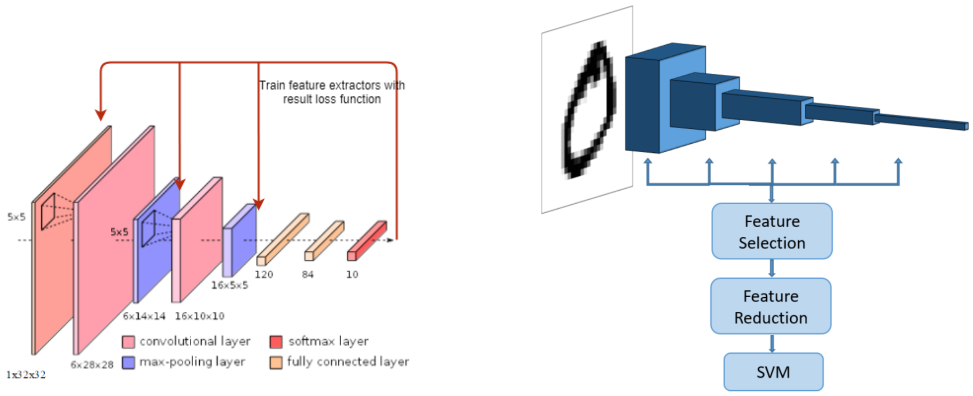
IV. Discussion

When I was doing experiment on VGG structure, I found out that if the decay I choose is small, the training would soon stopped at accuracy around 0.101, which mean failure of that setting. So when a network is really sophosticate, the drop rate should be large, at least 50% of the neurons should be dropped in the final fully connected layers.
So what would happen to a very complicated structure, one thing comes up to me is the gradient explosion and gradient vanishing. But we use activation functions ReLU, and this solves the gradient vanishing and explosion problem. So obvious not all the neurons and learned weight vectors is useful. So it should not be reasonable to design a very complicated system. The following saak transform shows a system with very limited transform kernels but still perform a good job.

XXV. Saak Tranform and Its Appliation to The MNIST Dataset

I. Saak Tranform and its Comparison with CNN

Unlike the CNN which uses lots of supervised data and BP algorithms to train the filters and find the image feature, Saak(Subspace approximation with augmented kernels) transform just performs forward operations. And use the learned features to train the classifier. The basic forward Saak transform can be illustrated in the following image:



(a) *CNN basic architecture* **(b)** *Saak basic architecture*

Figure 227: *CNN and Saak Transform*

I.1 Comparison between feature extraction technique

What is the 'filter' in Saak

We use lossless Saak tranform to illustrate the feature extraction. First is to explain the one stage Saak Transform.

Suppose we have N training images each with channel C, we first seperate each image into small cuboid (or cube if channel is 1) patches, the paper first adjust the image size into $L \times L \times C$ and let $log_2(L)$ to be the integer. Then uses the non-overlapping patches mask with size $2 \times 2 \times C = D$ to build the patch dataset. And actually I think any size can be chosen according to the training images. But here still I'll use the papers method to illustrate.Then we flatten the patches data and form a data matrix, We remove patches data with small variances and then perform KLT transform to get the eigenvectors. **These eigenvectors forms the transform kernel.**Then the resulting cuboid data are augmented and then perform another stage of Saak Transform.

For example, we have N training image, each with size $L \times L \times C$, suppose at the first stage, $L = 32, C = 1, N = 10000$. And the patch mask we choose is $2 \times 2 \times C = 2 \times 2 \times 1$. Then each image would generate $(32/2) \times (32/2) = 256$ patches, and in total we would get $N \times 256 = 2560000$ image patches with dimension2×2. Next, we flatten all the patches data into dimension1×4 and form a data matrix $B$with dimension 2560000×4. Then for each data in the dataset, remove

data with small variance and form a data matrix B' with size $N' \times 4$. Now we can perform the KLT(Karhunen-Loeve tranformation) and get the eigenvectors matrix, Since the dimension of the data matrix is $N' \times 4$, we get 4 eigenvectors and form a matrix K$[\underline{e_1}, \underline{e_2}, \underline{e_3}, \underline{e_4}]$ with dimension 4×4. With this kernel, an image with size $32 \times 32 \times 1$ can be transformed into $16 \times 16 \times 4$(each patch with size $2 \times 2 \rightarrow (flatten) 1 \times 4 \cdot K_{(4\times4)} = S_{(1\times4)}$).Next we augment the data by applying another kernel $-K$ and get the $16 \times 16 \times 8$ cuboid. Relu the resulting cuboid to get a new cuboid. So for the next stage, each image now becomes size $L = 16, C = 8 \rightarrow 16 \times 16 \times 8$, now the patch size becomes $2 \times 2 \times 8 = 32$, gathering all the patches from the training dataset to train a kernel matrix of size 32×32.

To conclude, the saak used the data-driven filter, formed with KLT transform in each stage.

what is 'filters' in CNN

As mentioned above, the filters are randomly initiated or manually designed with some traditional filter at first.Then the filters are modified according to the performance of a certain task. There are similarities between the Lenet-5 network and the Saak Transform, that they all use multistage to transform an image into a long vector using filters and convolution operation. The difference is the choice of the filter, one uses the training data instead of knowing the label of each image, the CNN uses the label of each training image and the classification performance to train the feature extractor.

I.2 Comparison Between Pipeline

As discuused above, the feature extraction part in Saak transform is independent of the classifier job. Therefore, after getting the feature vector which is interested, we can use the classifiers like SVM, random forest, to perform the classification job, or to perform the regression job. This part can be chosen freely. Since there is no BP algorithm, the classifier is trained once and the job is done.

However, in CNN, the whole job is one system that we can not take any part seperately out. The filters are trained again and again, which can be less efficient than Saak.The CNN also provides many activation functions for choice and the design of different layers and filters can be very different. However in Saak transform, the filter maximum amount is fixed(lossless saak transform). And the choice of filter is driven by data.

I.3 Comparison Between Performance

If a CNN structure is well defined, the performance can be perfect, however, we always have no idea how many layers and neuron numbers are suitable for a certain task. So to design a CNN structure can be painful and a lot of experiments have to be done. However, deep neural networks are easily fooled, the CNN might have some problems in robustness. Secondly, if the dataset is changed the CNN has to be training again for the classification problem which is not efficient

enough. In real life we always just use the trained network for the image classification problem. The saak can perform some state-of-art correnctness in classification problem and it is robust. The adding noise experiment showed that kind of feature. Moreover, Saak has scalability agianst object classes. When one CNN is trained, the input test image should fit the input neuron amount of the trained network.

I.4 Comparison Between Mathematical Transparency

Clearly, Saak transform is transparent in math , each kernel, and the classifier can be explained in math and calculated.
However, the filter weights in CNN is trained through much iterations and can not simpliy calculated and observed. Many stochastic process in the system.

II. Application of Saak transform to the MNIST Dataset

II.1 Experiment Routine

Settings: each stage retain 3,4,7,6,8 principle components

•1The first step is the Karhunen-Loeve tranformation. That is for a set of datas with some dimension, we can reduce the dimension by finding subspace that can represent these datas. In image case, we take the first step as an example. We first split the image into none overlap 2×2 patches, if we have an image of size 32×32 than we have 256 patches in a space with four dimensions. We can perform the KLT transform and find the 4 directions(basic functions) in the hyperspace. After getting the four basic functions , we can get the 4 responses with one dc responce. We take first 3 response as the transform kernel $K(4 \times 3)$.
•2Then is the augmented part, we just get $-K$, and a dc extraction kernel $DC = \sqrt{N}[1, 1, \cdots, 1]$ Therefore , the transform kernel becomes $[K, -K, DC]_{4\times 7}$ Then add the Relu function on it.
•3 After the first Saak tranform stage, we have a cuboid, with 7 channels, we still split the $16 \times 16 \times 7$ with $2 \times 2 \times 7$ masks, now we have a haperspace with dimensions $4 \times 7 = 28$. Then we keep doing the KLT. However it is easy to see that the dimension of the space would become larger and larger. So we perform the truncated KLT also known as PCA. We keep dimensions that we are interested in,4 in second stage.
•4 all the five stage data shape.

Table 46: *MultiSaak Transform Stage Ouptput*

	input	**truncated trained kernel**	**output**
stage 1	$32 \times 32 \times 1$	$2 \times 2 \times 7 \times 1$ **stride = (2,2)**	$16 \times 16 \times 7$
stage 2	$16 \times 16 \times 7$	$2 \times 2 \times 7 \times 9$ **stride = (2,2)**	$8 \times 8 \times 9$
stage 3	$8 \times 8 \times 9$	$2 \times 2 \times 9 \times 15$ **stride = (2,2)**	$4 \times 4 \times 15$
stage 4	$4 \times 4 \times 15$	$2 \times 2 \times 15 \times 13$ **stride = (2,2)**	$2 \times 2 \times 13$
stage 2	$2 \times 2 \times 13$	$2 \times 2 \times 13 \times 17$ **stride = (2,2)**	$1 \times 1 \times 17$

•5 Now we get an feature vector(with augmented features) with dimension $16 \times 16 \times 7 + 8 \times 8 \times 9 + 4 \times 4 \times 15 + 2 \times 2 \times 13 + 1 \times 1 \times 17 = 2677$,we stack all the feautures from all stages together.
•6 We use the F-score to select (1000,1500,2000) features from the all 2677 features. F-score: choose the best features that makes variance between class as large as possible, and in-class variance as small as possible. the following formula shows the F-score measurement for each feature.

$$F = \frac{between-groupvariability(BGV)}{within-groupvariability(WGV)} = \frac{\sum_{c=1}^{C} n_c (S_c - S)^2/(C-1)}{\sum_{c=1}^{C}\sum_{d=1}^{n_c}(S_{c,d} - S_c)/(T-C)}$$

where C is the class number, n_c is the number of Saak coefficients of the c_{th} class, S_C is the sample mean of the c_{th} class, S is the mean of all samples, $S_{c,d}$, is the d_{th} sample of the c_{th} class, and T is the total number of samples.
Now we get data with feature size 1000 or 1500 or 2000. **The learned feature selection indexes should be returned**
•7 Perform another round PCA, (32,64,128) **The pca components should also be returned**
•8 with the trained data features, train the classifier (SVM and Random Forest).
•9 Perform Saak transform on the testing set, with the same setting, use the learned feature selection indexes and PCA components to get feature data for the testing images.
•10 classify the testing image feature data with the trained classifier. And calculate the performance.

III. Brief introduction to SVM

S(**S**upport)V(**V**ector)M(**M**achine) are supervised learning models with associated learning algorithms that analyze data used for classification and regression. The first step is the nonlinear mapping which we can choose the kernel, in this experiments, we use RBF , radial basis function, which is a kernel formed with all dimensions and can let the new space with more dimensions. The second step is perform learning algorithm to linearly classify samples in the new space with loss function associated with support vectors. Support vectors are those data points that are closest to the training boundary.

IV. Code to Explain process

I designed the following steps to conduct experiemnts:

1: for training data get Saak Transform outputs

```
dataset, filters, outputs, datalabel = multi_stage_saak_trans()
```

stack all stage responses

```
    features = np.zeros((image_amount, 2677))
     for i in range(image_amount):
         feature = []
         for j in range(len(outputs)):
             stagej = outputs[j].data.numpy()
             feature = np.concatenate([feature, stagej[i, :].flatten()])
         features[i, :] = feature
```

F-score

```
    model = feature_selection.SelectKBest(score_func=feature_selection.ff, k=
        k_best_fscore)
    topFeatures = model.fit_transform(features, datalabel)
    indexs = model.get_support()
```

PCA

```
    pca = PCA(n_components=pcaComNumber)
    pcaTopFeatures = pca.fit_transform(topFeatures)
    return pca
```

2. for the testing data

get Saak coefficients

```
    datatestSet, filters, outputsTest, datalabeltest = multi_stage_saak_trans_test()
```

stack all features

```
    features = np.zeros((image_amount, 2677))
     for i in range(image_amount):
         feature = []
         for j in range(len(outputs)):
             stagej = outputs[j].data.numpy()
             feature = np.concatenate([feature, stagej[i, :].flatten()])
         features[i, :] = feature
```

F-score and PCA(use the learned models)

```
topFeatures_test = features[:, learnedIndex]
pcaTopFeatures_test = pca.transform(topFeatures_test)
```

Finally train the classifier and get the error

```
clf = svm.SVC()
clf.fit(pcaTopFeatures_train,datalabel)
error = 0
for i in range(1000):
    if(clf.predict([pcaTopFeatures_test[i]]) != datalabeltest[i]):
        error = error +1
print(error/1000)
```

V. Result

number of components in each stage:3,4,7,6,8
For feature selection choose from 1000.1500.2000
For last round PCA choose from 32,64,128

Table 47: *Test images accuracy(Random Forest)*

	last PCA dimension		
Dimension after F-test	32	64	128
2000	92.96	92.29	90.76
1500	92.71	92.30	91.14
1000	92.79	92.27	91.14

Table 48: *Test images accuracy(SVM)*

	last PCA dimension		
Dimension after F-test	32	64	128
2000	92.22	96.90	98.13
1500	92.28	96.97	98.13
1000	92.92	97.18	98.17

Therefore, the best accuracy to get is using SVM classifier and with final feature size 128. The acurracy reaches 98.17.
comment:
1.Compared to **CNN** Obvious the result is not better than the CNN, but almost the same performance regarding to the LeNet-5 setting to the MNIST dataset. But the most important feature of Saak Tranform is that it is an one-pass operation.And here for each stage we choose very small amount of retained features, so this accuracy already did well. I believe if more components are

retained at each stage, the performance can be better .
2. The performance fof Saak transform also depends on how the classifier is chosen, as we can see from the result, SVM playes much better than the random forest,(both use default settings). And the random forest gives a strange outcome that more PCA left, left accuracy for the training forest. For SVM more features left, accuracy becomes better.
3. I asked myself a question, whether the Saak transform with SVM would perform better if we change 3,4,7,6,8, to another setting that retains more information. Can it reach 99%? According to the paper, the accuracy is all between 98% to 99%, and there is a ceiling for Saak. Is there ceiling for the CNN? As shown in the following topic, actually some digit images can not be recognized by man, and it is reasonable to misclassify those images, so maybe a 98.5% for saak is more satisfy than a 99.6% by CNN.

VI. Error Analysis

To be fair, I compare the Lenet-5 result and the Saak result. One with the accuracy 98.9 and the other with the accuracy 98.17. In total we have 10000 test images, we collect the wrong index in Lenet5 and wrong indexs in Saak experiments.
Wrong image index predict by CNN and Saak

```
In [2]: print(wrongIndexarray)

[247, 259, 321, 340, 449, 495, 582, 583, 613, 619, 674, 684, 717, 740, 938, 947, 965, 1014, 1039, 1112, 1178, 1182, 1226, 1232,
1247, 1299, 1319, 1393, 1414, 1527, 1530, 1549, 1609, 1621, 1681, 1709, 1878, 1901, 2018, 2035, 2098, 2109, 2118, 2130, 2135, 2
293, 2369, 2387, 2414, 2447, 2454, 2462, 2597, 2654, 2742, 2743, 2771, 2896, 2921, 2939, 2995, 3073, 3330, 3337, 3422, 3503, 35
20, 3534, 3559, 3597, 3767, 3796, 3808, 3853, 3869, 3906, 4063, 4075, 4078, 4176, 4205, 4211, 4224, 4248, 4256, 4265, 4284, 436
0, 4497, 4500, 4536, 4571, 4639, 4740, 4761, 4807, 4814, 4956, 5600, 5642, 5842, 5937, 5955, 5973, 6091, 6166, 6172, 6505, 657
1, 6572, 6576, 6597, 6651, 6755, 7233, 8059, 8094, 8246, 8408, 8520, 9009, 9530, 9634, 9642, 9664, 9669, 9679, 9692, 9729, 976
8, 9770, 9779, 9811, 9856, 9888]
```

(a) *wrong index of CNN*

```
In [5]: print(errorindex)

[42, 68, 109, 110, 131, 200, 210, 245, 261, 353, 428, 517, 523, 546, 576, 640, 646, 666, 843, 851, 870, 886, 992, 1000, 1044, 1
174, 1205, 1346, 1545, 1551, 1563, 1679, 1701, 1748, 1771, 1896, 1916, 1961, 1965, 2069, 2091, 2151, 2237, 2306, 2447, 2455, 26
00, 2639, 2664, 2686, 2693, 2729, 2773, 3014, 3015, 3056, 3094, 3117, 3175, 3242, 3254, 3260, 3287, 3321, 3425, 3433, 3439, 344
3, 3494, 3503, 3628, 3888, 4211, 4236, 4271, 4310, 4361, 4420, 4458, 4483, 4554, 4566, 4569, 4578, 4597, 4605, 4701, 4740, 478
1, 4782, 4786, 4798, 4860, 4878, 4901, 4907, 4915, 5018, 5023, 5045, 5320, 5354, 5372, 5453, 5483, 5506, 5555, 5636, 5704, 578
9, 5798, 5994, 6018, 6038, 6096, 6179, 6182, 6224, 6242, 6323, 6368, 6428, 6519, 6559, 6608, 6621, 6686, 6699, 6776, 6782, 680
0, 6819, 6886, 6943, 7035, 7157, 7182, 7212, 7241, 7379, 7424, 7428, 7468, 7530, 7568, 7585, 7653, 7754, 7757, 7816, 7884, 790
3, 7937, 7947, 7970, 8234, 8297, 8416, 8465, 8501, 8550, 8573, 8629, 8633, 8655, 8782, 8790, 8966, 8981, 9097, 9144, 9163, 920
7, 9283, 9292, 9474, 9570, 9598, 9649, 9870, 9886]
```

(b) *wrong index of Saak*

Figure 228: *wrong indexs*

intersect the two indexs and find the same images taht both failed to pass the test case. Show the images, to find their similarities.

```
In [11]: cnnErrorIndex = np.asarray(cnnErrorIndex)
         saakerrorIndex = np.asarray(errorindex)
         sameindex = np.intersect1d(cnnErrorIndex, saakerrorIndex)
         print(sameindex)

         [1527 2135 3767]

In [14]: for i in range(3):
             image = datatestSet[sameindex[i]]
             image = np.array(image,dtype = 'float')
             pixels = image.reshape((32,32))
             plt.subplot(1,3,i+1)
             plt.imshow(pixels, cmap='gray')
             plt.title('true label:{} \n pred Label: {}'.format(datalabeltest[sameindex[i]],predictLabel[sameindex[i]]))
         plt.show()
```

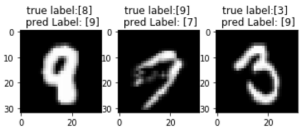


Figure 229: *The images that both classified wrong*

therefore only around **2.7%** test errors are the same.The others are different. Now I would look into the error image in CNN and error images in Saak and compare the performance difference.I pick 40 images misclassified by CNN and Saak

```
In [40]: for i in range(40):
             image = mnist.test.images[wrongIndexarray[i]]
             image = np.array(image,dtype = 'float')
             pixels = image.reshape((28,28))
             plt.subplot(4,10,i+1)
             plt.imshow(pixels, cmap='gray')
             plt.title('true label:{} \n pred Label: {}'.format(np.argmax(mnist.test.labels[wrongIndexarray[i]]),predictLabels[wrongIndexa
         plt.show()
         plt.rcParams["figure.figsize"] = [15,20]
```

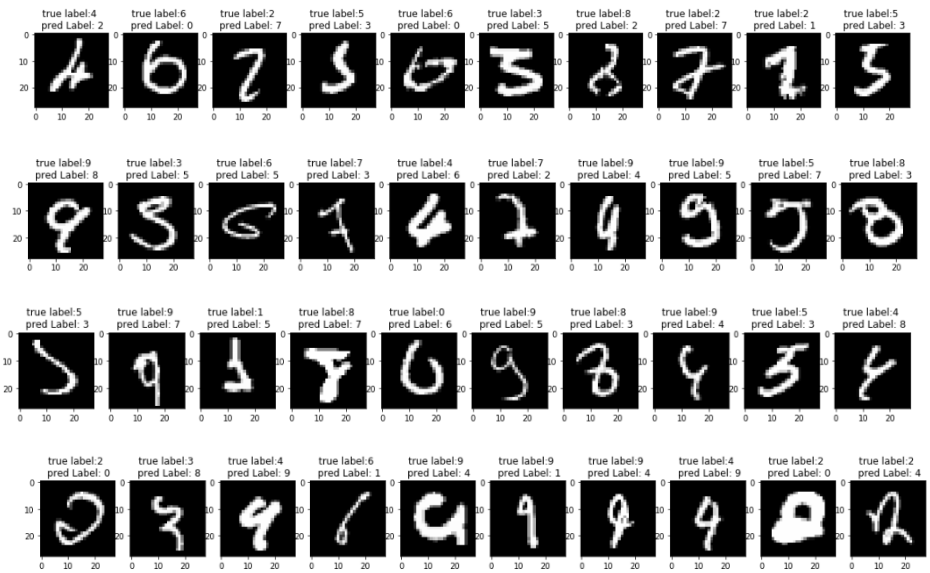


Figure 230: *Wrong Classification under LeNet-5*

```
In [37]: for i in range(40):
             image = datatestSet[saakerrorIndex[i]]
             image = np.array(image,dtype = 'float')
             pixels = image.reshape((32,32))
             plt.subplot(4,10,i+1)
             plt.imshow(pixels, cmap='gray')
             plt.title('true label:{} \n pred Label: {}'.format(datalabeltest[saakerrorIndex[i]],predictLabel[saakerrorIndex[i]]))
         plt.show()
         plt.rcParams["figure.figsize"] = [15,20]
```

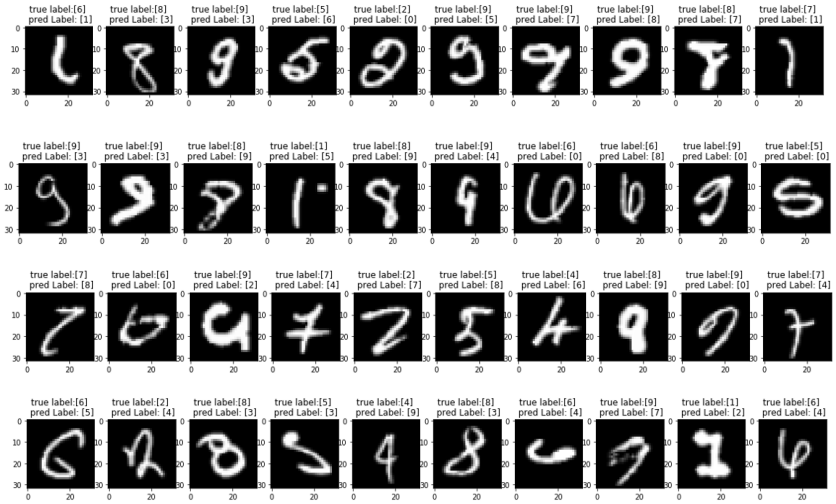


Figure 231: *wrong classification with Saak*

Confusion Matrix

```
In [45]: predictLabel = np.asarray(predictLabel)
         datalabeltest = np.asarray(datalabeltest)
         from sklearn.metrics import confusion_matrix
         C = confusion_matrix(datalabeltest,predictLabel)
         print(C)
```

```
[[ 974    0    1    0    0    2    1    1    1    0]
 [   0 1128    4    0    0    1    0    0    1    1]
 [   4    0 1010    4    1    0    1    6    5    1]
 [   0    0    3  995    1    2    0    5    3    1]
 [   0    1    3    0  961    0    4    1    1   11]
 [   2    0    0    7    1  876    2    1    2    1]
 [   3    3    0    0    3    5  941    0    3    0]
 [   0    3    9    0    3    0    0 1005    1    7]
 [   4    0    1    5    1    4    0    2  954    3]
 [   2    2    5   10    8    2    0    9    2  969]]
```

(a) *Saak Confusion Matrix*

```
In [48]: trueLabel = np.argmax(mnist.test.labels,1)

In [49]: from sklearn.metrics import confusion_matrix
         C = confusion_matrix(trueLabel,predictLabels)
         print(C)
```

```
[[ 972    0    1    0    0    0    3    1    2    1]
 [   0 1132    0    0    0    1    2    0    0    0]
 [   2    1 1018    1    1    0    0    5    4    0]
 [   0    0    1 1001    0    3    0    1    4    0]
 [   0    0    1    0  976    0    1    0    2    2]
 [   1    0    0    7    0  881    2    1    0    0]
 [   4    2    0    0    2    2  948    0    0    0]
 [   0    1    3    1    0    0    0 1019    1    3]
 [   1    0    2    4    1    2    2    2  958    2]
 [   0    3    0    4   10    5    0    4    4  979]]
```

(b) *LeNet-5 Confusion Matrix*

Figure 232: *Confusion Matrix*

We can see that both CNN and Saak have problem to distinguish 4 with 7(with line in between), and the 8 and 9 when the under circle of the 8 is small, and 1 and 7 when the head of the seven is small. Moreover is the 3 and 5, when the head of the 5 is not proper placed. Also the 4 and 9 has some incorrectness, maybe because of the closure of the circle.Actually a lot of images would lead to misunderstanding, and the accuracy performed by Saak and CNN is promising. So , I would take some images above to explain the potential improvement.

Potential improvement

First, for the common error in CNN and Saak, we can disgard such misclassification, as u can see, the true label for the first image in figure 15 is 8, well I believe most of us recognize it as 9 like the CNN and Saak does. And for the second image, no one can tell it is 9. For the third image, it

is a bit unsatisfied, so may be the CNN and Saak should find the feature of a closure of a circle. So I have such imagnation, if a system see a digit look like 9 but actually labeled 4, it design a filter with 'breaking'operation , to break the head of the digit to be 4. Also , if it sees a digit look like 4 but actually labeled 9, it performs 'Connecting' operation to connect the head into a circle. Moreover, if it sees a digit looks like 9 but actually labeled 8, it add 'erosion' operation to the filter to let the bottom part of the digit forming a circle.
The difficulty is that different images have different orientations and different misleading reasons, so a lot of operations should be created, which is not a easy job to design such a system. Maybe it is why we need to make a CNN system more and more complicated and design a lot of different filters. For the Saak, actually I think we can build a system with the same idea but with different settings on the kernel size and amount of important components, then we can get a lot of features with different settings, then all the features form a 'feature forest', with the given label, we should find a path to walk out of the forest while collecting good information in the forest. This is just a thought which can be very complicated. First is how to design the feature forest, and how to orgnize the data, second is what is the optimization to select the best features for training.
To sum up, for both CNN and Saak, we are still trying to find the transform kernel to extract the best features we want in each image, and we don't want the kernels to be complicated because for human, we only need several obvious kernels to recognize a digit. like whether it has two holes, whether it has one corner. So a simple but beautiful method is still to be find.

References

[1] Tony Linderberg *Feature Detection with Automatic Scale Selection* International Journal of Computer Vision 30(2), 79-116, 1998

[2] David G.Lowe *Distinctive Image Features from Scale-Invariant Keypoints* International Journal of Computer Vision 60(2), 91-110, 2004

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool *Speeded-Up Robust Features (SURF)* Computer Vision and Image Understanding 110 (2008) 346-359