Image Processing - Classification of Colonoscopy Images - Fast Fourier Transformation

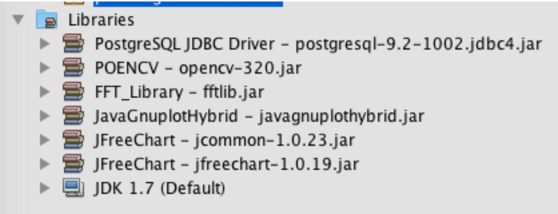
Cheropoulos Panteleimon, Shulika Ganna, Steininger Stefan

Overview

Setup

- Programming language: Java

- IDE: Netbeans 7, Eclipse
- Database: Postgresql



What we did:


```
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Size;
import org.opencv.imgproc.CLAHE;
import org.opencv.imgproc.Imgproc;
```

1. Preprocess

```
private void preprocess(Image image,
        double clipLimit, int gridsize) throws IOException {

    image.convertTo4ColorModels(); //Lab, HSV, HLS, YUV
    image.splitAllColorModelsTo3Channels();
    image.enchanceContrast(clipLimit, gridsize); //each color
    image.normalize();  //each color channel
}
```

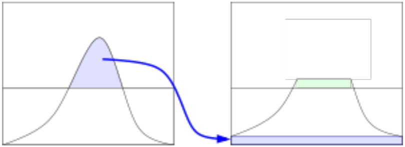
What we did:


```java
private void convertTo4ColorModels() throws IOException{
    Imgproc.cvtColor(imageRGB, imageHSV, Imgproc.COLOR_RGB2HSV);
    Imgproc.cvtColor(imageRGB, imageHLS, Imgproc.COLOR_RGB2HLS);
    Imgproc.cvtColor(imageRGB, imageLab, Imgproc.COLOR_RGB2Lab);
    Imgproc.cvtColor(imageRGB, imageYUV, Imgproc.COLOR_RGB2YUV);
}
```

```java
private void splitTo3ColorChannels(Mat colorM
    List<Mat> lRgb = new ArrayList<>(3);
    Core.split(colorModel, lRgb);

    lRgb.get(0).copyTo(channel1);
    lRgb.get(1).copyTo(channel2);
    lRgb.get(2).copyTo(channel3);
}
```

What we did:

```java
private void enchanceContrast(double clipLimit, int gridsize) {
    CLAHE c = Imgproc.createCLAHE();
    c.setClipLimit(clipLimit);
    c.setTilesGridSize(new Size(gridsize, gridsize));

    c.apply(imageHSV_H, imageHSV_H);
    c.apply(imageHSV_S, imageHSV_S);
```



-Costs: O(N) (N = pixel width of surrounding rectangle)

What we did:

```
private void normalize() {
    Core.normalize(imageHSV_H, imageHSV_H, 0, 255, Core.NORM_MINMAX, CvType
    Core.normalize(imageHSV_S, imageHSV_S, 0, 255, Core.NORM_MINMAX, CvType
    Core.normalize(imageHSV_V, imageHSV_V, 0, 255, Core.NORM_MINMAX, CvType
```

2. Process

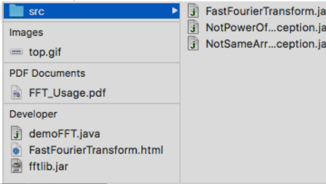
```
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Rect;
```

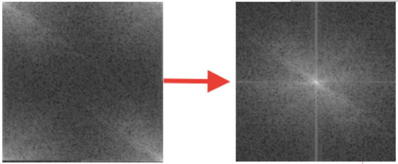
-library for FFT Laboratory of Neural Imaging
http://www.loni.usc.edu/Software/FFT

```
public void process(boolean vectorsInDB, double contrastClipLimit, int co

    for (Images.Image resImage : images) {

        preprocess(resImage, contrastClipLimit, contrastGridsize);

        resImage.fft(fftAmplitude);
        resImage.extractFeature(extractFeatureRadius);

    }
```

What we did:

```
static void performFastFourierTransform(Mat colorChannel, int amp) {
    colorChannel.convertTo(colorChannel, CvType.CV_64F);

    double[] realPart = new double[colorChannel.rows() * colorChannel.cols()];
    double[] imagenaryPart = new double[colorChannel.rows() * colorChannel.cols()];
    colorChannel.get(0, 0, realPart);

    for (int i = 0; i < realPart.length; i++) {
        realPart[i] = amp * realPart[i]; // real part
        imagenaryPart[i] = 0.0; // imagiary part
    }

    //the LONI library for 2D fft used
    FastFourierTransform.fastFT(realPart, imagenaryPart, true);

    colorChannel.put(0, 0, realPart);
    int cx = colorChannel.cols() / 2;
    int cy = colorChannel.rows() / 2;

    Mat q0 = new Mat(colorChannel, new Rect(0, 0, cx, cy));
    Mat q1 = new Mat(colorChannel, new Rect(cx, 0, cx, cy));
    Mat q2 = new Mat(colorChannel, new Rect(0, cy, cx, cy));
    Mat q3 = new Mat(colorChannel, new Rect(cx, cy, cx, cy));

    Mat tmp = new Mat();

    //replace 4 rectangles
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);

    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}
```





FFT vs DFT

- DFT - Approximation to fourier integral. Maps discrete vector to another discrete vector and it can be viewed as a matrix operator.

- FFT - Special computational algorithm for DFT

FFT vs DFT

- Given discrete time sequence x[i], i=1,2,....i-1 - and y[j], j=1,2,..., j-1,
- N=i-1 - M=j-1

- F(i , j) = $\sum_{k=0}^{N} \sum_{l=0}^{M} f(k,l) * e^{-2*\iota*\pi*((i*k)/n)*((j*l)/n)}$

- k=1,2,...,N

-it is represented by “magnitude” and “phase” rather that the “real” and “imaginary” parts
were:
- magnitude(F) $= Sqrt(real(F)^2 + imaginary(F)^2)$
and
- phase(F) $= Atan(imaginary(F)/real(F)))$

FFT vs DFT

- **Magnitude**: explains "how much" of a certain frequency component is present

- **Phase**: tells "where" the frequency component is in the image

- after the evaluation of DFT we become an image with different frequencies

- in the middle we see the brightest region, the Frequencies are the lowest in this position

FFT vs DFT (FFT)

- Efficient algorithm to compute the DFT and the IDFT

- Evaluation of DFT take $O(n^2)$ arithmetic operations

- FFT reduces that to $O(n\ log\ n)$

- FFT requires the number of data points to be a power of 2

FFT vs DFT (FFT - trick of the speed up)

- Divide & Conquer in use

- polynomial evaluation is splitted in “odd” and “even” parts , recursively example:

p(x)=p0*x$^0 + p1 * x^1$

$-----> p(x) = even + odd$

FFT vs DFT (FFT - trick of the speed up)

- why this?

- cause the vector must have a length of $M = 2^N$

- so we split this until we have vectors with length 2^0

- than it holds that $\forall k \in 0, ..., (N/2) - 1$:

- $\overrightarrow{f}(2k) = f(2k * 2\pi/N)$

- $\overrightarrow{f}(2k+1) = f([2k+1] * 2\pi/N)$

FFT vs DFT (FFT - trick of the speed up)

- then the fft is computed with N vectors of length 1

- recursively we have : $\forall k \in 0, ..., (N/2) - 1$

- $\hat{f}(k) = 1/2 * (\hat{f}(2k) + [e^{-2*i*\pi/N}]^k * \hat{f}(2k+1))$

- $\hat{f}(k + N/2) = 1/2 * (\hat{f}(2k) - [e^{-2*i*\pi/N}]^k * \hat{f}(2k+1))$

- Proof

1. Case 1: $\forall k \in 0, ..., (N/2)-1$

$$
\begin{aligned}
\hat{f}_k &= \langle \vec{f}, \vec{w}_k \rangle_N = \frac{1}{N} \sum_{m=0}^{N-1} f_m \cdot e^{-ikm \cdot 2\pi/N} \\
&= \frac{1}{N} \sum_{m=0}^{(N/2)-1} f_{2m} \cdot e^{-ik[2m] \cdot 2\pi/N} \\
&\quad + \frac{1}{N} \sum_{m=0}^{(N/2)-1} f_{2m+1} \cdot e^{-ik[2m+1] \cdot 2\pi/N} \\
&= \frac{1}{2} \cdot \frac{1}{N/2} \sum_{m=0}^{(N/2)-1} f_{2m} \cdot e^{-ikm \cdot 2\pi/(N/2)} \\
&\quad + \frac{1}{2} \cdot \frac{1}{N/2} \sum_{m=0}^{(N/2)-1} f_{2m+1} \cdot e^{-ikm \cdot 2\pi/(N/2)} \cdot e^{ik\pi/(N/2)} \\
&= \frac{1}{2} \cdot \left(\hat{f}_{2k} + [e^{-i \cdot 2\pi/N}]^k \cdot \hat{f}_{2k+1} \right)
\end{aligned}
$$

FFT vs DFT (FFT - trick of the speed up - Proof)

- Proof

2. Case 2:$\forall k \in 0, ..., (N/2) - 1$

$$
\begin{aligned}
\hat{f}_{k+\frac{N}{2}} &= \langle \vec{f}, \vec{w}_{k+\frac{N}{2}} \rangle_N = \frac{1}{N} \sum_{m=0}^{N-1} f_m \cdot e^{-i[k+\frac{N}{2}]m \cdot 2\pi/N} \\
&= \frac{1}{N} \sum_{m=0}^{(N/2)-1} f_{2m} \cdot e^{-i[k+\frac{N}{2}][2m] \cdot 2\pi/N} \\
&\quad + \frac{1}{N} \sum_{m=0}^{(N/2)-1} f_{2m+1} \cdot e^{-i[k+\frac{N}{2}][2m+1] \cdot 2\pi/N} \\
&= \frac{1}{2} \cdot \frac{1}{N/2} \sum_{m=0}^{(N/2)-1} f_{2m} \cdot e^{-i[k+\frac{N}{2}]m \cdot 2\pi/(N/2)} \\
&\quad + \frac{1}{2} \cdot \frac{1}{N/2} \sum_{m=0}^{(N/2)-1} f_{2m+1} \cdot e^{-i[k+\frac{N}{2}]m \cdot 2\pi/(N/2)} \cdot e^{-i[k+\frac{N}{2}] \cdot \pi/(N/2)}
\end{aligned}
$$

FFT vs DFT (FFT - trick of the speed up - Proof)

$\Rightarrow$

$$
\begin{aligned}
&= \frac{1}{2}\cdot\frac{1}{N/2}\sum_{m=0}^{(N/2)-1} f_{2m}\cdot e^{-ikm\cdot 2\pi/(N/2)}\cdot e^{-im\cdot 2\pi} \\
&\quad + \frac{1}{2}\cdot\frac{1}{N/2}\sum_{m=0}^{(N/2)-1} f_{2m+1}\cdot e^{-ikm\cdot 2\pi/(N/2)}\cdot e^{-im\cdot 2\pi}\cdot e^{-ik\pi/(N/2)}\cdot e^{-i\cdot\pi} \\
&= \frac{1}{2}\cdot\left(\hat{f}_{2k} - [e^{-i\cdot 2\pi/N}]^k\cdot\hat{f}_{2k+1}\right)
\end{aligned}
$$

FFT vs DFT (FFT - trick of the speed up)

- compute the DFT of the two smaller signals (N/2)

- So total number of multiplications performed in calculating DFT of S1 and S2 will be $2 * (N/2)^2$

- Then to get actual DFT of N point signal we need to add these components

- So we can see that we actually need approximately $2 * (N/2)^2 = (n^2)/2$ which is half that in case of N point signal

- join them to get the DFT of the larger signal

FFT vs DFT (FFT - trick of the speed up)

- This reduction of half is a result of dividing the signals once

- But if we continue dividing the signals again and again, we will end up in reducing the amount of computations in each step by an amount which is much lesser than N^2

- The number of all computations using this method comes of the order of $N * logN$

FFT vs DFT (FFT - trick of the speed up)

- to clarify the difference of FFT and DFT , assume that we have $N = 10^9$ points

- DFT needs 10^{18} nanoseconds that corresponds to 31 years!!!

- FFT needs
$N * logN = (10^9) * log(10^9) = (10^9) * 9 * log10 = (10^9) * 9 * 3.32$
that corresponds to 30 seconds!!!!

FFT vs DFT(FFT.java)

-public static void fastFT(double[][] realArray, double[][] imagArray, boolean direction) – computes the DFT in 2D arrays using the FFT algorithm

-The two double array representing the real and imaginary part of data

-they are required for computation

-The dimension must be an integer power of 2 or an error will be catched

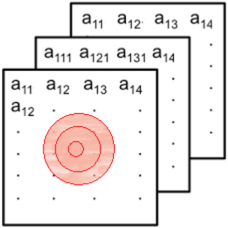
FFT vs DFT(FFT.java)

-or the Inverse function will be wrong

-If the Boolean value is “true” then the FFT will be performed

-otherwise the IFFT will be performed

Feature extractor

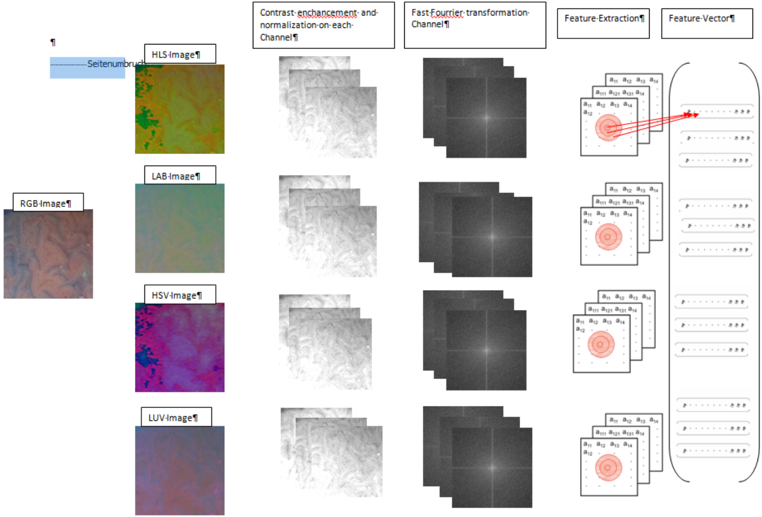


-Each color-channel is represented in a 2 d matrix.

-Starting in the center, each value is accumulated

- $\bar{x} := 1/n \sum_{i=0}^{n} x$ i

Summary creating the Feature Vector



Classification

-In this class we examine the Images and we try to classify them

-The Images are represented as 2D double arrays (12 vectors each of them is a colour channel from 4 colour models we used to convert to)

-Then we try from this vectors to find the K-nearest-neighbours

-The distance between the points of vectors is computed with the Euclidean Distance ($d(p, q) = \sqrt{\sum_{i=1}^{n}(qi - pi)^2}$)

Classification

-Also we take a Vector and we compare that with the k-nearest Vectors

-After that we have to find the majority voted values , the most common values

-This value corresponds to the Classification(represents the class of the image)

Classification

Not classified Images

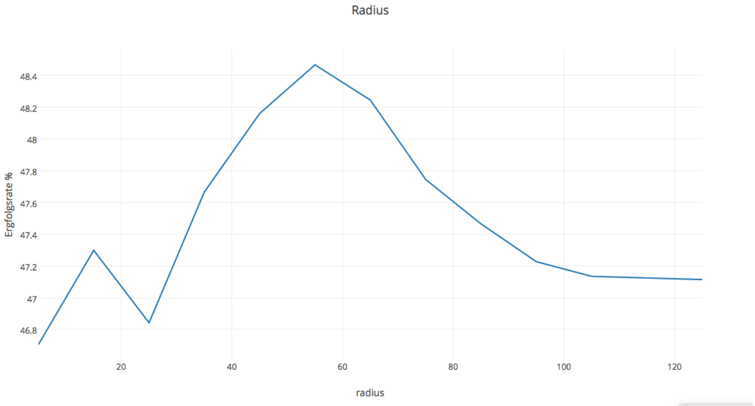
imageid character varying	patientid integer	imagevectors2darray double precision[]	cancerclass integer
akh0021-cut-1112628391503.png	4	488.586087103071,-0.490356735608626,1.0684641474169,0.0257577788...	
akh0021-cut-1112628394906.png	4	496.136406056119,1.02676365363704,-2.08242470529874,-0.010165977...	
akh0022-cut-1112628400010.png	4	487.844941493637,0.974526333844335,0.169075085437837,-1.47254070...	
akh0023-cut-1112628409356.png	4	442.212468647799,-19.6037371597897,-6.06960059333297,2.639640644...	
akh0023-cut-1112628412322.png	4	498.889486431014,-5.78463557603526,-1.54649110207206,-0.23485212...	
akh0024-cut-1112628422807.png	4	530.064223253549,11.6066280396119,1.19142652609627,-0.9593530390...	
akh0025-cut-1112628432080.png	4	452.962337502026,-0.135832988448613,-1.79552927923374,-1.0665782...	
akh0025-cut-1112628434637.png	4	712.886358070908,-11.7019075488949,-7.81132676962348,-0.48509805...	
akh0026-cut-1112628442292.png	4	502.266745214671,-9.55834267180161,-4.14706507620101,-2.77449215...	
akh0026-cut-1112628447141.png	4	708.651004533413,1.11921663450306,-1.41538833719811,0.3865531754...	
akh0027-cut-1117461382843.png	4	322.671653603719,-15.750697787635,0.611735711410681,-0.083931571...	
akh0028-cut-1117461387734.png	4	299.102490479583,-4.87981668051873,-6.0364461338033,4.6461948611...	
AKH0031_CUT2.png	5	558.045125848484,-10.5908085101333,2.65912618386934,0.5033585205...	
AKH0035_CUT1.png	5	354.088153836908,-12.9526762630556,-14.3820639730054,2.899111692...	

Classification

Classified Images

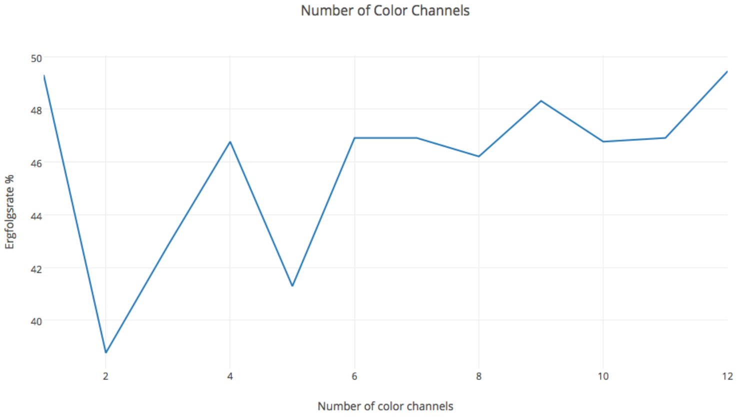
imageid character varying	patientid integer	imagevectors2darray double precision[]	cancerclass integer
akh0021-cut-1112628391503.png	4	488.586087103071,-0.490356735608626,1.0684641474169,0.0257577788...	2
akh0021-cut-1112628394906.png	4	496.136406056119,1.02676365363704,-2.08242470529874,-0.010165977...	2
akh0022-cut-1112628400010.png	4	487.844941493637,0.974526333844335,0.169075085437837,-1.47254070...	2
akh0023-cut-1112628409356.png	4	442.212468647799,-19.6037371597897,-6.06960059333297,2.639640644...	2
akh0023-cut-1112628412322.png	4	498.889486431014,-5.78463557603526,-1.54649110207206,-0.23485212...	2
akh0024-cut-1112628422807.png	4	530.064223253549,11.6066280396119,1.19142652609627,-0.9593530390...	2
akh0025-cut-1112628432080.png	4	452.962337502026,-0.135832988448613,-1.79552927923374,-1.0665782...	2
akh0025-cut-1112628434637.png	4	712.886358070908,-11.7019075488949,-7.81132676962348,-0.48509805...	2
akh0026-cut-1112628442292.png	4	502.266745214671,-9.55834267180161,-4.14706507620101,-2.77449215...	2
akh0026-cut-1112628447141.png	4	708.651004533413,1.11921663450306,-1.41538833719811,0.3865531754...	2
akh0027-cut-1117461382843.png	4	322.671653603719,-15.750697787635,0.611735711410681,-0.083931571...	2
akh0028-cut-1117461387734.png	4	299.102490479583,-4.87981668051873,-6.0364461338033,4.6461948611...	2
AKH0031_CUT2.png	5	558.045125848484,-10.5908085101333,2.65912618386934,0.5033585205...	2

Results

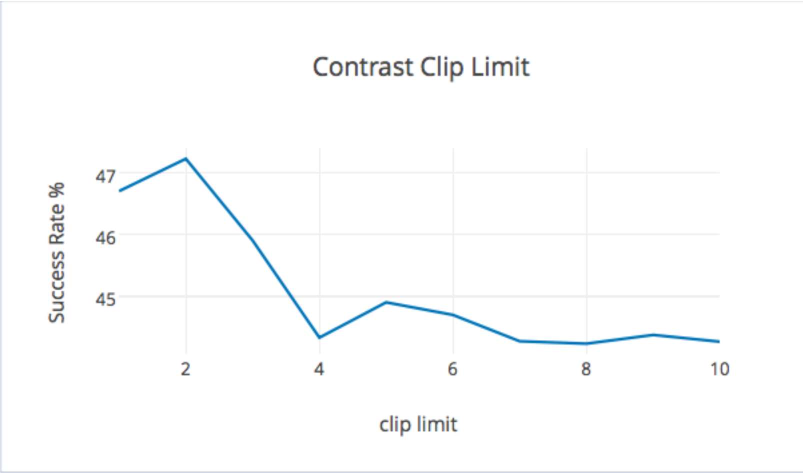


Radius
48.4
48.2
48
47.8
47.6
47.4
47.2
47
46.8
Ergfolgsrate %
20
40
60
80
100
120
radius

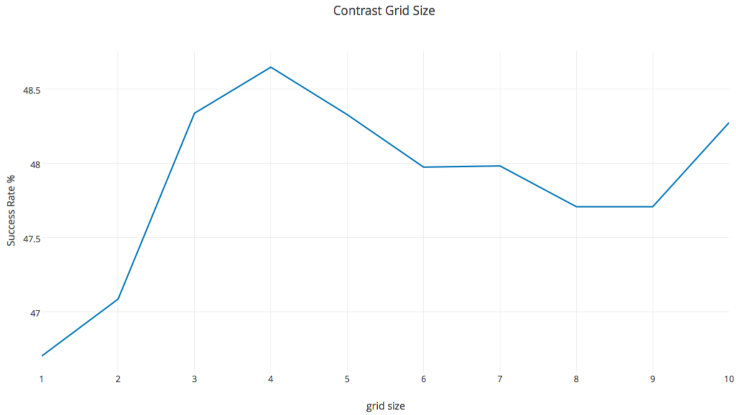
Results


Number of Color Channels
Ergfolgsrate %
50
48
46
44
42
40
2
4
6
8
10
12
Number of color channels

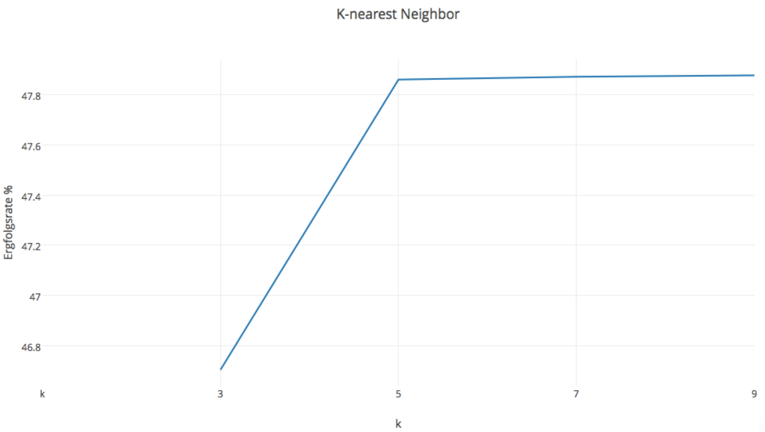
Results



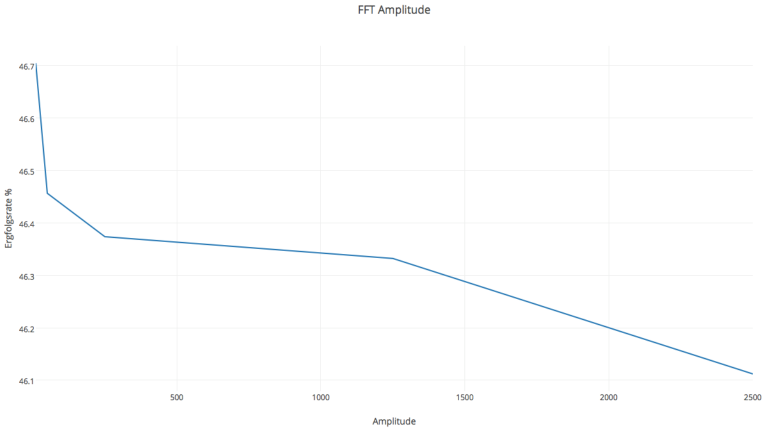
Results


Contrast Grid Size
48.5
48
47.5
47
Success Rate %
1
2
3
4
5
6
7
8
9
10
grid size

Results


K-nearest Neighbor
Ergfolgsrate %
47.8
47.6
47.4
47.2
47
46.8
k
3
5
7
9
k

Results



FFT Amplitude
46.7
46.6
46.5
46.4
46.3
46.2
46.1
Ergfolgsrate %
500
1000
1500
2000
2500
Amplitude

Results

Testing with the parameters that showed the best results:

contrastClipLimit = 2,
contrastGridsize = 4,
fftAmplitude = 10,
extractFeatureRadius = 55,
numOfChannels = 1,
k = 7,
Success rate = 53.576437587657786 (the maximum reached % of accuracy)

Overal results are between 45 - 49 %

Problems

1.OpenCV Software installation problems:
- installation bugs, difficulties, system recovery

- missing software for the installation (Homebrew, Cmake)

- version conflicts and missing functions(2, 3.0.0, 3.2.0)

- adding .jar files, libraries to Netbeans 7.1 and Eclipse Mars IDEs

Problems

- Mac Os Yosemite problems while installing, updating to Sierra helped a lot, problems with installing on Linux, it was impossible to set up the other computers for testing

2. Bad understanding how to build the vectors and do k-nearest neighbor classification

3. Time pressure

4. High computational demand: java.lang.outOfMemoryError after 2 hours of testing mixing different color channels (arguments(-Xmx512m) in the "VM Options" helped)

5. Finding intervals for parameters

Sources

1.Arenz, Patrick. "Diskrete und Schnelle Fourier Transformation."(2005). https://www.math.uni-trier.de/ /~schulz/Prosem-0405/Arenz.pdf?id=13814
2. FFT's are fast DFT Algorithms. http://www.engineeringproductivitytools.com/stuff/T0001/PT02.HTM
3. Explaining why FFT is faster than DFT http://cs.stackexchange.com/questions/11371/explaining-why-fft-is-faster-than-dft-for-the-general-pub

4. Wikipedia https://wikipedia.org
5. Kammeyer, Ing Karl-Dirk, and Ing Kristian Kroschel. "Die diskrete Fourier-Transformation (DFT)."Digitale Signalverarbeitung. Vieweg+ Teubner Verlag, 2002. 219-258.https://ti.tuwien.ac.at/cps/teaching/courses/dspv/files/DFT-FFT.pdf

Thank you for your attention!