Lesson 1,2,3:

#include "opencv2\opencv.hpp"

using namespace cv;

int main(int argv, char\*\* argc) {

Mat test = imread("color.jpg", CV\_LOAD\_IMAGE\_UNCHANGED);

namedWindow("color", CV\_WINDOW\_FREERATIO);

//resizeWindow("color", file1.cols/2, file1.rows/2);

//moveWindow("color", 1600, 800);

imshow("color", test);

test = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

imshow("gray", test);

imwrite("gray.jpg", test);

waitKey();

}

Lesson 4:

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

int main(int argv, char\*\* argc) {

Mat original = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

Mat modified = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

for (int r = 0; r < modified.rows; r++) {

for (int c = 0; c < modified.cols; c++) {

modified.at<uint8\_t>(r, c) = original.at<uint8\_t>(r, c) \* 0.5f;

}

}

Mat origcol = imread("color.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat modicol = imread("color.jpg", CV\_LOAD\_IMAGE\_COLOR);

for (int r = 0; r < modicol.rows; r++) {

for (int c = 0; c < modicol.cols; c++) {

modicol.at<Vec3b>(r, c)[0] = origcol.at<Vec3b>(r, c)[0] \* 0;

}

}

namedWindow("original", CV\_WINDOW\_FREERATIO);

namedWindow("modified", CV\_WINDOW\_FREERATIO);

moveWindow("original", 200, 500);

moveWindow("modified", 600, 500);

namedWindow("origcol", CV\_WINDOW\_FREERATIO);

namedWindow("modicol", CV\_WINDOW\_FREERATIO);

moveWindow("origcol", 1000, 500);

moveWindow("modicol", 1400, 500);

imshow("original", original);

imshow("modified", modified);

imshow("origcol", origcol);

imshow("modicol", modicol);

waitKey(10000);

}

Lesson 5:

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

int main(int argv, char\*\* argc) {

Mat origcol = imread("color.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat splitChannels[3];

split(origcol, splitChannels);

splitChannels[2] = Mat::zeros(splitChannels[2].size(), CV\_8UC1); //CV\_8U, CV\_8UC, CV\_8UC1 = kollha l istess.

Mat output;

merge(splitChannels, 3, output);

namedWindow("origcol", CV\_WINDOW\_AUTOSIZE);

moveWindow("origcol", 200, 200);

namedWindow("b", CV\_WINDOW\_AUTOSIZE);

moveWindow("b", 600, 200);

namedWindow("g", CV\_WINDOW\_AUTOSIZE);

moveWindow("g", 200, 800);

namedWindow("r", CV\_WINDOW\_AUTOSIZE);

moveWindow("r", 600, 800);

namedWindow("merge", CV\_WINDOW\_AUTOSIZE);

moveWindow("merge", 800, 200);

imshow("origcol", origcol);

imshow("b", splitChannels[0]);

imshow("g", splitChannels[1]);

imshow("r", splitChannels[2]);

imshow("merge", output);

waitKey();

}

Lesson 6:

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

void takeDFT(Mat& source, Mat& destination) {

}

int main(int argv, char\*\* argc) {

//dft only works on grayscale. for color: split to grayscale.

Mat original = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

//object for converted values

Mat originalFloat;

//convert from 255 to 1

original.convertTo(originalFloat, CV\_32FC1, 1.0 / 255.0);

//matrix object to hold both real and imaginary components for dft

Mat originalComplex[2] = { originalFloat, Mat::zeros(originalFloat.size(), CV\_32F) };

Mat dftReady;

//(input array (source), 2 channels (2 arrays becoming two channel), output array (destination))

merge(originalComplex, 2, dftReady);

Mat dftofOriginal;

//input array, output array, flags

dft(dftReady, dftofOriginal, DFT\_COMPLEX\_OUTPUT);

//we use pass by reference so not to make a copy of the obj, modify it and send copy back to replace

waitKey();

}

Lesson 7:

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

void takeDFT(Mat& source, Mat& destination) {

//object for int to float

Mat originalFloat;

//convert from 255 to 1

source.convertTo(originalFloat, CV\_32FC1, 1.0 / 255.0);

//matrix object to hold both real and imaginary components for dft

Mat originalComplex[2] = { originalFloat, Mat::zeros(originalFloat.size(), CV\_32F) };

Mat dftReady;

//(input array (source), 2 channels (2 arrays becoming two channel), output array (destination))

merge(originalComplex, 2, dftReady);

//input array, output array, flags

dft(dftReady, destination, DFT\_COMPLEX\_OUTPUT);

//we use pass by reference so not to make a copy of the obj, modify it and send copy back to replace

}

void recenterDFT(Mat& source) {

int centerX = source.cols / 2;

int centerY = source.rows / 2;

Mat q1(source, Rect(0, 0, centerX, centerY));

Mat q2(source, Rect(centerX, 0, centerX, centerY));

Mat q3(source, Rect(0, centerY, centerX, centerY));

Mat q4(source, Rect(centerX, centerY, centerX, centerY));

Mat swapMap;

q1.copyTo(swapMap);

q4.copyTo(q1);

swapMap.copyTo(q4);

q2.copyTo(swapMap);

q3.copyTo(q2);

swapMap.copyTo(q3);

}

void invertDFT(Mat& source) {

Mat inverse;

dft(source, inverse, DFT\_INVERSE | DFT\_REAL\_OUTPUT | DFT\_SCALE);

imshow("invert", inverse);

}

void showDFT(Mat& source) {

Mat splitArray[2] = { Mat::zeros(source.size(), CV\_32F), Mat::zeros(source.size(), CV\_32F) };

//input matrix is source, output is split in two channels

split(source, splitArray);

Mat dftMagintude;

magnitude(splitArray[0], splitArray[1], dftMagintude);

//add one to every value to make the log function properly

dftMagintude += Scalar::all(1);

//(input, output)

log(dftMagintude, dftMagintude);

//(input, output, min, max, how we normalizing[from n=min to max])

normalize(dftMagintude, dftMagintude, 0, 1, CV\_MINMAX);

recenterDFT(dftMagintude);

imshow("DFT", dftMagintude);

invertDFT(source);

}

int main(int argv, char\*\* argc) {

//dft only works on grayscale. for color: split to grayscale.

Mat original = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

if (original.empty())

return -1;

//the performance of a DFT is dependent of image size so we pad borders. expand input image to optimal size

Mat padded;

int r = getOptimalDFTSize(original.rows);

int c = getOptimalDFTSize(original.cols);

//on border add zero pixels

copyMakeBorder(original, padded, 0, r - original.rows, 0, c - original.cols, BORDER\_CONSTANT, Scalar::all(0));

Mat dftofOriginal;

takeDFT(padded, dftofOriginal);

showDFT(dftofOriginal);

waitKey();

}

Lesson 8:

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

void createGaussian(Size& size, Mat& output, int uX, int uY, float sigmaX, float sigmaY, float amplitude = 1.0f) {

Mat temp = Mat(size, CV\_32F);

for (int r = 0; r < size.height; r++) {

float y = ((r - uY) \* ((float)r - uY)) / (2.0f \* sigmaY \* sigmaY);

for (int c = 0; c < size.width; c++) {

float x = ((c - uX) \* ((float)c - uX)) / (2.0f \* sigmaX \* sigmaX);

float value = amplitude \* exp(-(x + y));

temp.at<float>(r, c) = value;

}

}

normalize(temp, temp, 0.0f, 1.0f, NORM\_MINMAX);

output = temp;

}

int main(int argv, char\*\* argc) {

Mat output;

createGaussian(Size(256, 256), output, 256 / 2, 256 / 2, 10, 10);

imshow("gaussian", output);

waitKey();

}

Lesson 9:

//split the matrix in 4 with a line

#include "opencv2\opencv.hpp"

#include <stdint.h>

using namespace cv;

using namespace std;

int main(int argv, char\*\* argc) {

//create a matrix

Mat output = Mat(256, 256, CV\_8UC1, Scalar(0));

//retrieving the row at index 0

Mat row\_128 = output.row(128);

//setting values of all elements in the row with index 128

row\_128.setTo(Scalar(255));

//retrieveing the col with index 3

Mat col\_128 = output.col(128);

//setting values of all elements in the row with index 128

col\_128.setTo(Scalar(255));

imshow("output", output);

waitKey();

}

Lesson 10:

#include "opencv2\opencv.hpp"

#include "opencv2\highgui.hpp"

#include <stdint.h>

#include <iostream>

using namespace cv;

using namespace std;

int main(int argv, char\*\* argc) {

//create an empty image

Mat image;

cout<< "This image is " << image.rows << " by " << image.cols <<endl;

//This will produce an image made of unsigned bytes that OpenCV designates with the constant CV\_8U

image = imread("color.jpg", IMREAD\_COLOR); //read image, decode it, allocate mem

if (image.empty()) {

cout << "no data found" << endl;

}

namedWindow("original image"); //define the window(optional)

imshow("original image", image);

cout << "Now this image is " << image.rows << " by " << image.cols << " and has " << image.channels() << " number of channels " << endl;

Mat result; //create another empty image

flip(image, result, 1); //positive for horizontal, 0 for vertical, negative for both

//Point class is used to specify pixel coordinate.

circle( result, //desitnation image

Point(image.cols / 2, image.rows /2 ), //center coordinates

65, //radius

0, //color

2); //thickness

putText(result, //destination image

"This is the center", //text

Point(result.cols /2 , 280), //center coordinate

FONT\_HERSHEY\_COMPLEX, //font

0.5, //font scale

255, //color (course its just blue)

2); //font thickness

imshow("fliped", result);

waitKey(0); //0 specifies for a key pressed

}

Lessons 10:

#include <iostream>

#include <opencv2\core.hpp>

#include <opencv2\highgui.hpp>

using namespace cv;

using namespace std;

//test function that creates an image

Mat grayFunction() {

//create image

Mat ima(500, 500, CV\_8U, 50);

//only a shallow copy of the image will be transferred

return ima;

//ima is destroyed outside this scope but copy of image is intact in the returned var.

}

int main() {

//create a new image made of 240 rows and 320 columns, type unsigned char = 1 byte pix (grayscale)

Mat image1(240, 320, CV\_8U, 100);

imshow("image", image1); //show the image

waitKey(0);

//re-allocate a new image

image1.create(200, 200, CV\_8U);

image1 = 200;

imshow("Image", image1); //show the image

waitKey(0);

//create a red color image. channel order is BGR. Scalar holds color

Mat image2(240, 320, CV\_8UC3, Scalar(0, 0, 255));

//Mat image2(Size(320, 240), CV\_8UC3); image2 = Scalar(0,0,255);

imshow("Image", image2);

waitKey(0);

Mat image3 = imread("color.jpg");

//all these images point to the same data block

Mat image4(image3);

image1 = image3;

//these images are new copies of the source image

image3.copyTo(image2);

Mat image5 = image3.clone();

//transform the image for testing

flip(image3, image3, 1);

//check which images have been effected by the processing

imshow("Image 3", image3);

imshow("Image 1", image1);

imshow("Image 2", image2);

imshow("Image 4", image4);

imshow("Image 5", image5);

waitKey(0);

//get a gray level image from a function

Mat gray = grayFunction();

imshow("gray", gray);

waitKey(0);

//read the image in gray scale

image1 = imread("color.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

//copy an image into another image that does not necessarily have the same data type

image1.convertTo(image2, CV\_32F, 1 / 255.0, 0.0);

imshow("image2", image2);

waitKey(0);

return 0;

}

Lesson 11:

#include <iostream>

#include <opencv2\core.hpp>

#include <opencv2\highgui.hpp>

using namespace cv;

using namespace std;

int main() {

Mat image = imread("color.jpg", CV\_LOAD\_IMAGE\_COLOR);

imshow("image", image);

cout << image.type() << " " << image.channels() << endl;

Mat logo(100, 100, CV\_8UC3, Scalar(0, 0, 255));

// imageROi and image point to the same matrix. any op will affect the same image.

Mat imageROI(image, Rect( image.cols - logo.cols, //ROI upper left corner start x

image.rows - logo.rows, //ROI upper left corner start y

logo.cols, //ROI width x

logo.rows)); //ROI height y

// row and col ranges. range is a continuous sequence from a start index to an end index excluding both)

//imageROI = image(Range(image.rows - logo.rows, image.rows), Range(image.cols - logo.cols, image.cols));

logo.copyTo(imageROI);

imshow("image", image);

waitKey(0);

return 0;

// to define an ROI made up of some lines of an image:

// Mat imageROI = image.rowRange(start, end);

// Mat imageROI = image.colRange(start, end);

}

Lesson 12:

#include <iostream>

#include <random>

#include <opencv2\core.hpp>

#include <opencv2\highgui.hpp>

using namespace cv;

using namespace std;

/\*

For element access, MAt has the at (int row, int col) method. However the type returned by the metod must be known at compile time,

and since Mat can hold elements of any type, the programmer needs to specify the return type that is expected. The 'at' method has

been implemented as a template method. So when you call it, you must specify the image type as: image.at<uchar>(rows, cols) = pixel value;

In color images, each pixel is associated with BGR, i.e. 3 channels. Mat class for a color image returns a vector of three 8 bit values.

Vec3b is a vector of three unsigned characters. The channel[index] designates one of the three color channels.

\*/

void salt(Mat& image, int numOfGrains) {

default\_random\_engine generator;

uniform\_int\_distribution<int> randomRow(0, image.rows - 1);

uniform\_int\_distribution<int> randomCol(0, image.cols - 1);

int c, r;

for(int i = 0; i < numOfGrains; i++){

//random image coordinate

c = randomCol(generator);

r = randomRow(generator);

//for gray level images

if (image.type() == CV\_8UC1) {

//singel channel 8 bit image

image.at<uchar>(r, c) = 255;

}//for color images

else if (image.type() == CV\_8UC3) {

image.at<Vec3b>(r, c)[0] = 255;

image.at<Vec3b>(r, c)[1] = 255;

image.at<Vec3b>(r, c)[2] = 255;

//image.at<Vec3b>(r, c) = Vec3b(255, 255, 255);

}

}

}

int main() {

Mat image = imread("color.jpg", IMREAD\_COLOR);

salt(image, 3000);

imshow("salty", image);

waitKey(0);

}

Lesson 13: