Project Motivation

Every engineering discipline can benefit from different image processing techniques, but in civil and more specifically traffic engineering are the benefits more obvious. Image processing in civil engineering is extremely useful as managing flow of traffic both for vehicles and humans is very important. P.G. Michalopoulos' Autoscope program hopes to use image processing to track vehicles and more specifically incidents. This will allow civil engineers to gather data on hotspots of incidents along roads so that they can plan fixes to whatever civil structure is causing the issue. The program can use image processing techniques to identify an incident and then take record of that whether that be adding a 1 to a variable or doing more complicated calculations from there. Needless to say this data can display incident densities over a strip of road to help towards the betterment of traffic safety. Another more simple application of image processing is the detection of high density traffic. This allows civil engineers to gather data on when and how dense the traffic is and then make educated decisions to help fix the issue. As opposed to the algorithm that detects incidents, this program will calculate the number of distinct vehicles in an area and the return data based on that as to assist whichever civil engineering team is responsible for the improvement of the road. The last application discussed here will be the detection of crowd density for human beings. Safety is crucial to town planners and designers and finding how crowds move and fluctuate is important in making a town or building as safe as possible. Through image processing, engineers can gather data on how the density of crowds shifts in an area over time. From there civil engineers can use that data to create safer buildings in the future and modify existing structures to make them more safe. Image processing has been used in numerous applications over all engineering fields, and the techniques that have been applied in the past remain useful and practical to this day.

Project Overview and Methods

Input: The user will input a PNG file name and the input function will then see if the image can be found in the directory and if it can, the function will transform the image into a 3d matrix of [x][y][z] where x and y are the dimensions of the image in pixels and z contains the RGB values of each pixel. The function will then do some modifications to the matrix so that the RGB values are in the integer range from 0-255 and then return that matrix. If the file cannot be found, the function will print an error message before exiting.

Output: The function accepts a 2d matrix of [a][b] and attempts to convert the matrix into an image. The function assumes that the imputed matrix will be in the form where x and y are the dimensions of the image in pixels, with each value at [a][b] being associated with an acceptable RGB value. The function will create an array "x" of dimensions [a][b][3] so that the image handling functions can output an RGB image despite only being in grayscale. The function will

then set the value of each [a][b][1-3] in x equal to [a][b] of the inputted matrix. The function will divide the 3rd dimension values of x by 255 to convert them back into float point form so that the functions imsave and implot work properly. The function will then convert the matrix into an image and save it into a directory and will also plot the image in Spyder.

Grayscale: This function of the code controlled the conversion of the original image into a grayscale image. The function's input is a 3D array representing the original image. For each pixel in the original image, there are individual RGB values to control the color of the pixel. In order to take this information and create a grayscale image with a 2D array, which is needed to perform the rest of the code, the function creates a blank 2D array with the same XY sizes as the original image. Then, the code runs through every x value and y value in the original image. It takes the RGB values of each pixel, converts them to a grayscale brightness, and then inputs that grayscale value into the corresponding pixel in the blank 2D array to create a grayscale version of the image in a 2D array. In order to get this code to run I needed to do research

MyTrix: This code is the code that actually performs the various transformations that controls the edge detection of the image. This code will take the 2D array of the grayscale image, and first run it through the blurring part of the code. This will look at small chunks of the code and apply a matrix transformation that averages the values of a group of pixels, which keeps the code from picking up on small color changes and interpreting them as edges. Instead, it will only consider large changes: the actual edges of the image, to actually be edges. The enhance portion of the code runs next, which takes the value of each pixel below a certain threshold and divides it by two. Since the value of each pixel is between 0 - 255 and the threshold value used later in code is small, this division will only affect the darkest parts of the image and serves to remove background noise from the image. Next, this code will run through edge detection in the x direction. By applying a matrix that isolates x edge detection to the blurred image, we are given an image that faintly shows the edges of the image in the x direction. Then, the code performs the same type of function to the y direction of the image. After finding the x and y edges of the image, the brightness values of the two separate images are combined into one image. Finally, the function will take this faint image of the edges and apply a specific threshold to the image. If the brightness of any given pixel is larger than a certain given value, then the pixels white while every other pixel turns black, giving us a black and white image of the edges of the initial image.

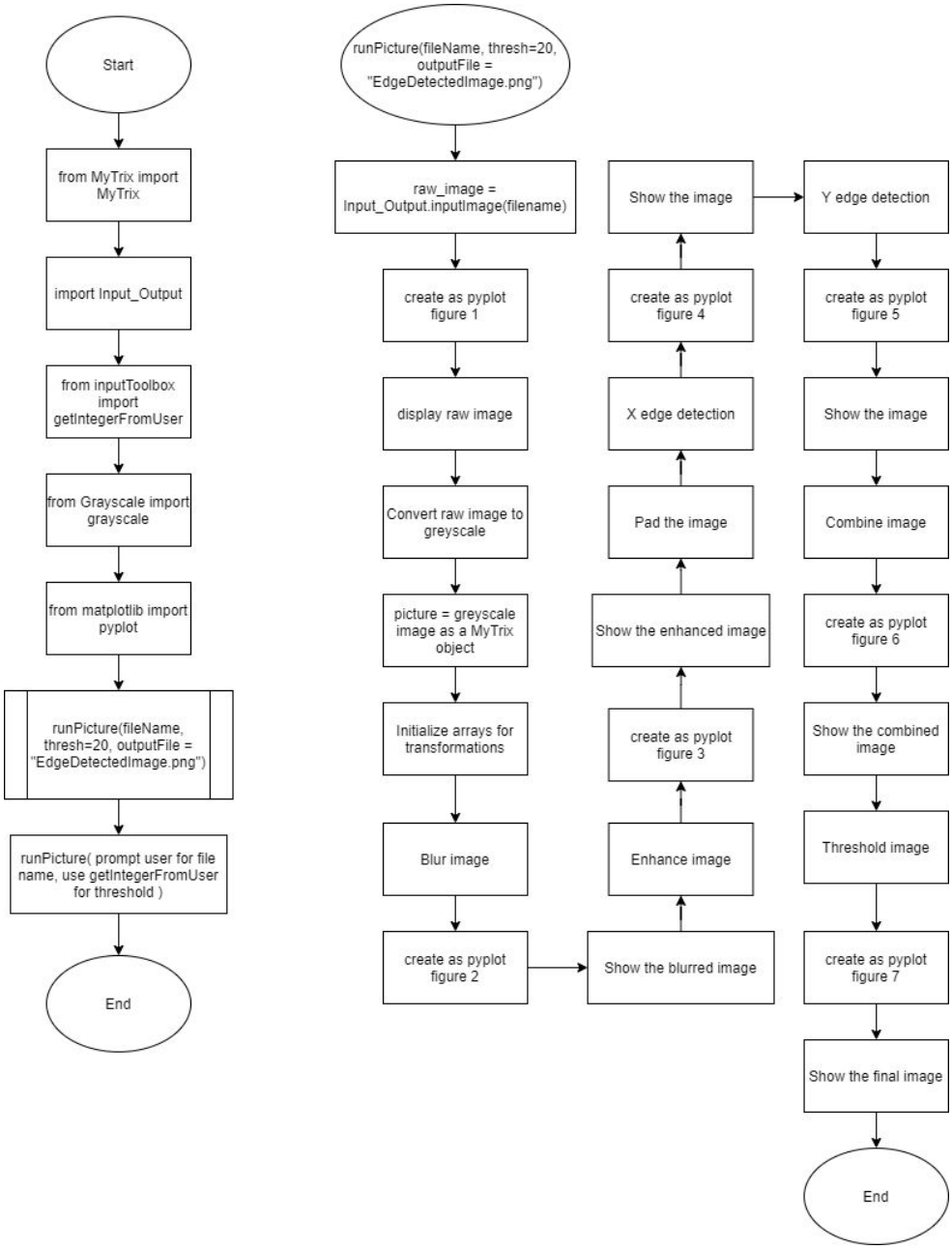
Main: The Main portion of the code is what times together all different modules previously written. In this code, the various transformations are defined, and then the code runs through each different module to tie everything together. It first runs through the Input function, which gets the name of the file and makes sure that the file can be accessed, then through the grayscale portion of the code, then MyTrix to actually transform the image, and finally runs through Output to return the final version of the image. Throughout this process, the program updates the user on what step it is on in the program, and prints out the image as it goes through various

steps to allow the user to see what is going on. This whole process usually takes a few seconds, but allows the user to follow along with the code while the edge detection is taking place.

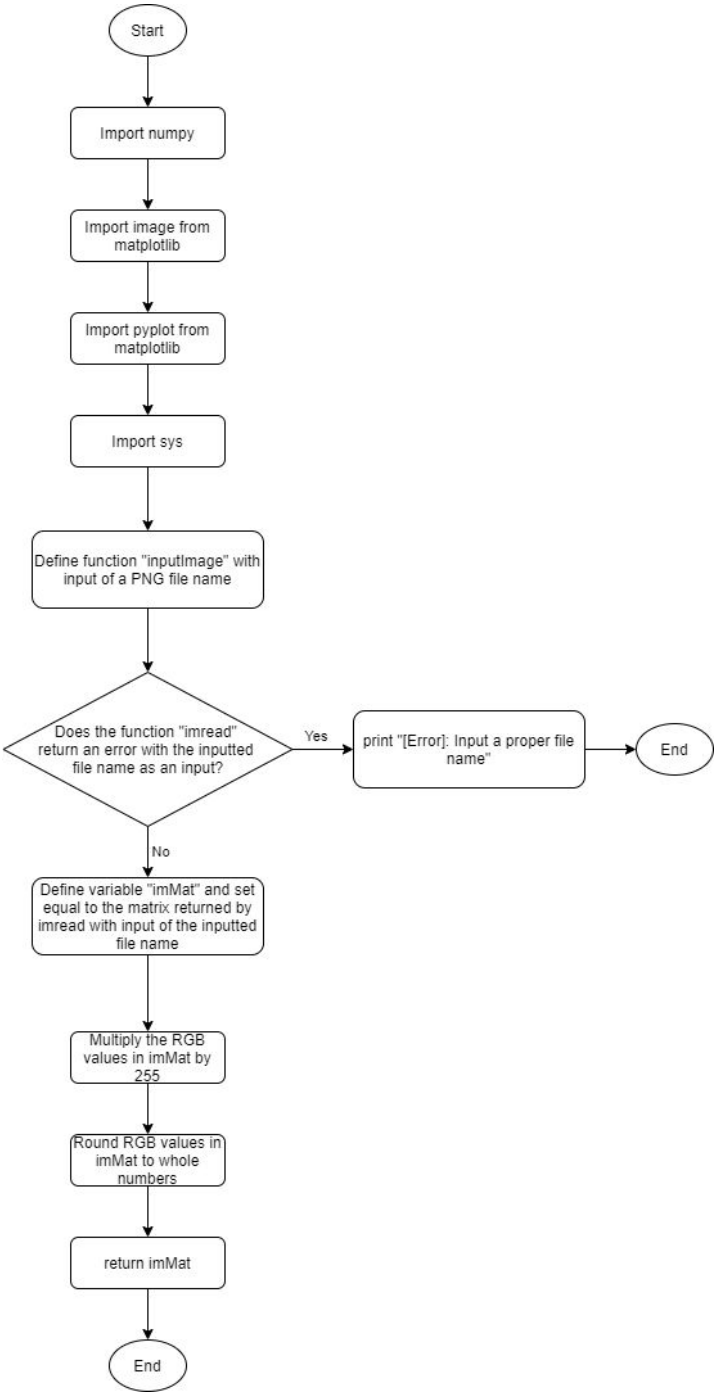
Discussion of Algorithm Design

Flowcharts:

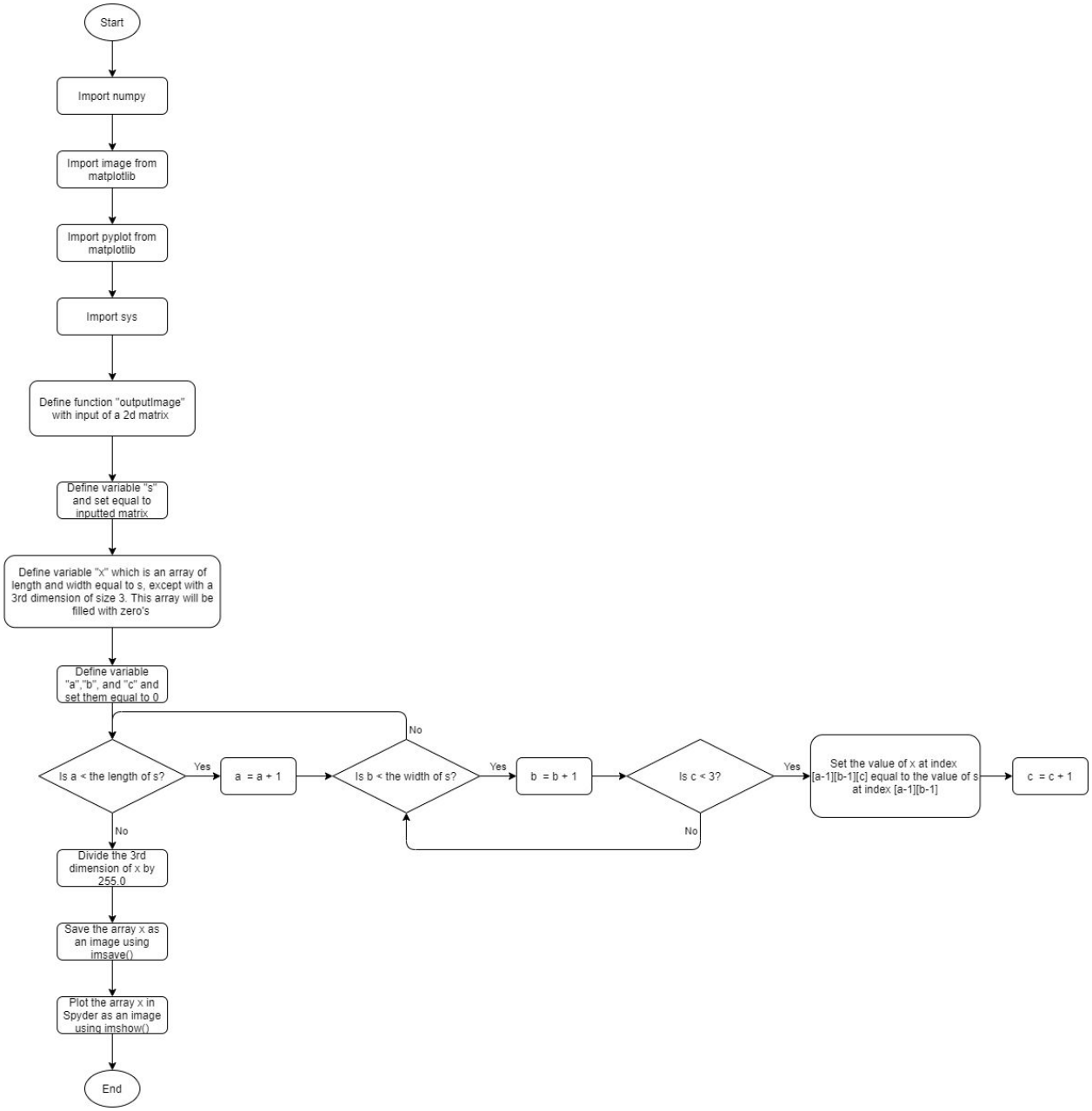
Main:



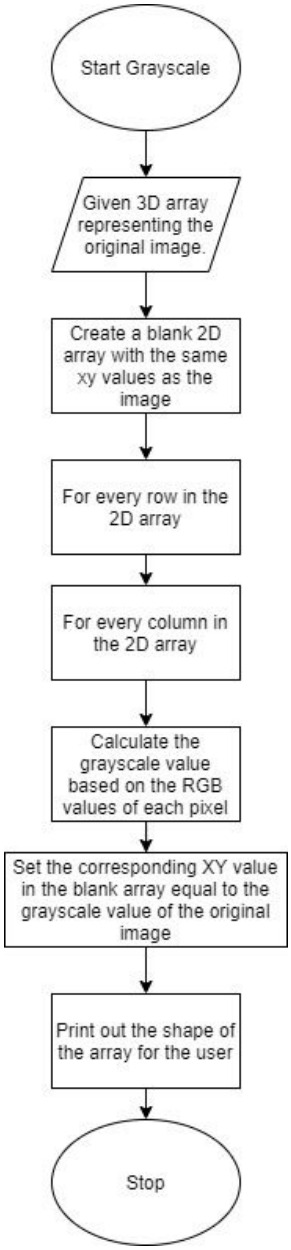
Input:


Start
Import numpy
Import image from matplotlib
Import pyplot from matplotlib
Import sys
Define function "inputImage" with input of a PNG file name
Does the function "imread" return an error with the inputted file name as an input?
Yes
print "[Error]: Input a proper file name"
End
No
Define variable "imMat" and set equal to the matrix returned by imread with input of the inputted file name
Multiply the RGB values in imMat by 255
Round RGB values in imMat to whole numbers
return imMat
End

Output:


Start
Import numpy
Import image from matplotlib
Import pyplot from matplotlib
Import sys
Define function "outputImage" with input of a 2d matrix
Define variable "s" and set equal to inputted matrix
Define variable "x" which is an array of length and width equal to s, except with a 3rd dimension of size 3. This array will be filled with zero's
Define variable "a","b", and "c" and set them equal to 0
Is a < the length of s?
Yes
a = a + 1
Is b < the width of s?
Yes
b = b + 1
Is c < 3?
Yes
Set the value of x at index [a-1][b-1][c] equal to the value of s at index [a-1][b-1]
c = c + 1
No
No
No
Divide the 3rd dimension of x by 255.0
Save the array x as an image using imsave()
Plot the array x in Spyder as an image using imshow()
End

Grayscale:

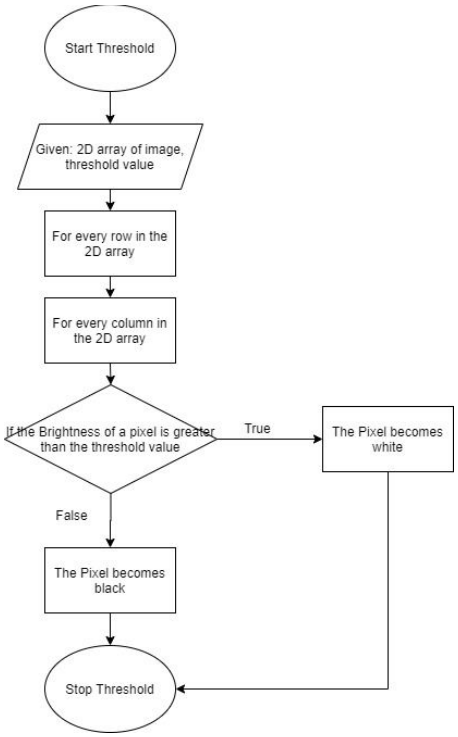

Start Grayscale
Given 3D array representing the original image.
Create a blank 2D array with the same xy values as the image
For every row in the 2D array
For every column in the 2D array
Calculate the grayscale value based on the RGB values of each pixel
Set the corresponding XY value in the blank array equal to the grayscale value of the original image
Print out the shape of the array for the user
Stop

MyTrix:

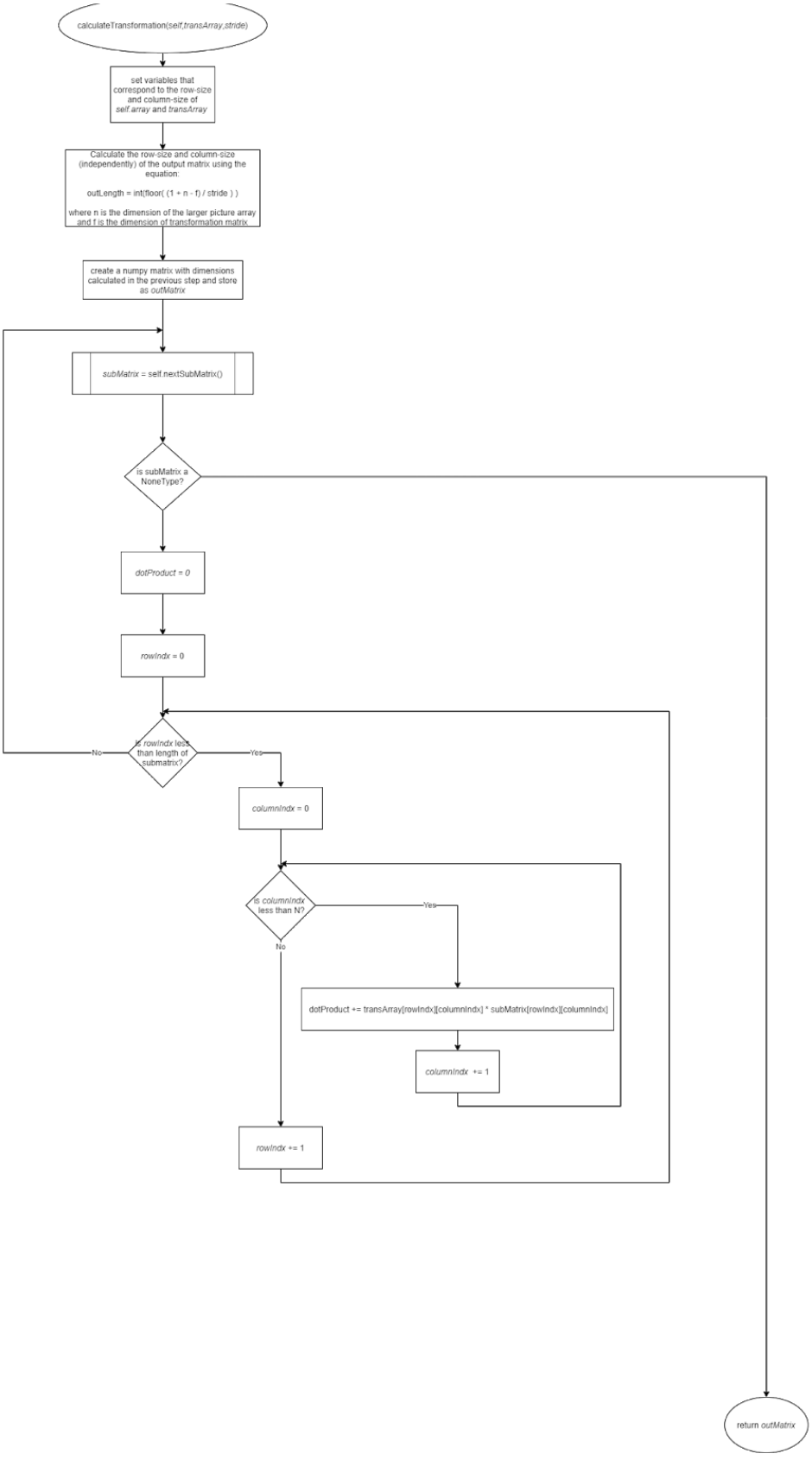
UML:

MyTrix
array: numpy.ndArray(float) debugMode: boolean largestValueAllowed: int subRowSize: int subColumnSize: int stride: int subRowIndx: int subColumnIndx: int
__init__(self): None at(int,int): float __str__(self): String __getitem__(self,int key): array[] __add__(self,MyTrix addMat): MyTrix setDebugMode(self, boolean debugMode): None getArrayCopy(self): numpy.ndArray(float) toInt(self): numpy.ndArray(int) threshold(self, int thresh, int highVal, int lowVal): none enhance(self): None startSubMatrixAquisition(self, int rowSize, int columnSize, int stride) nextSubMatrix(self): None replaceSubMatrix(self,array[] newArray): None pad(self,int padding): None calculateTransformation(self,array[][] transArray, int stride): numpy.ndArray(float) transform(self, array[][] transArray, int padding, int stride): None

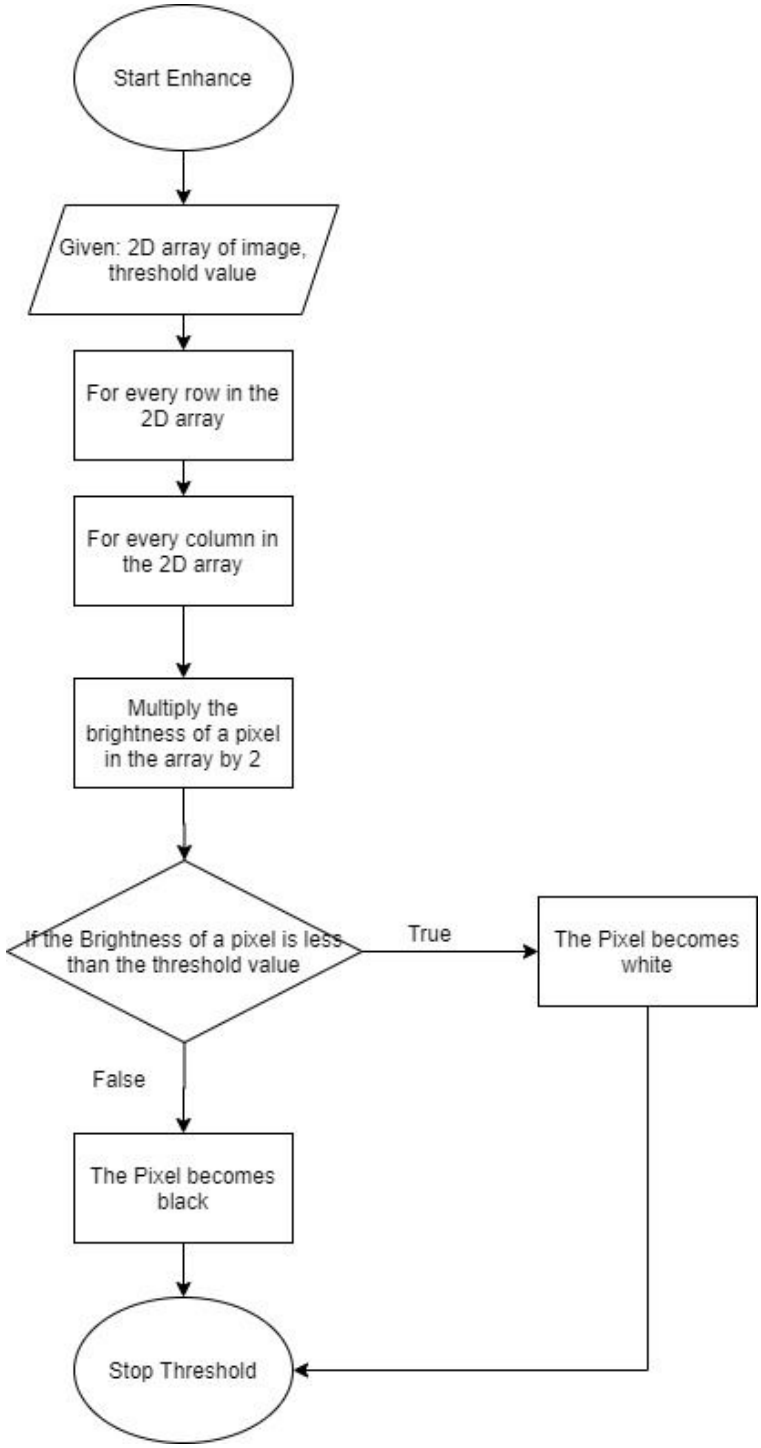
Threshold:



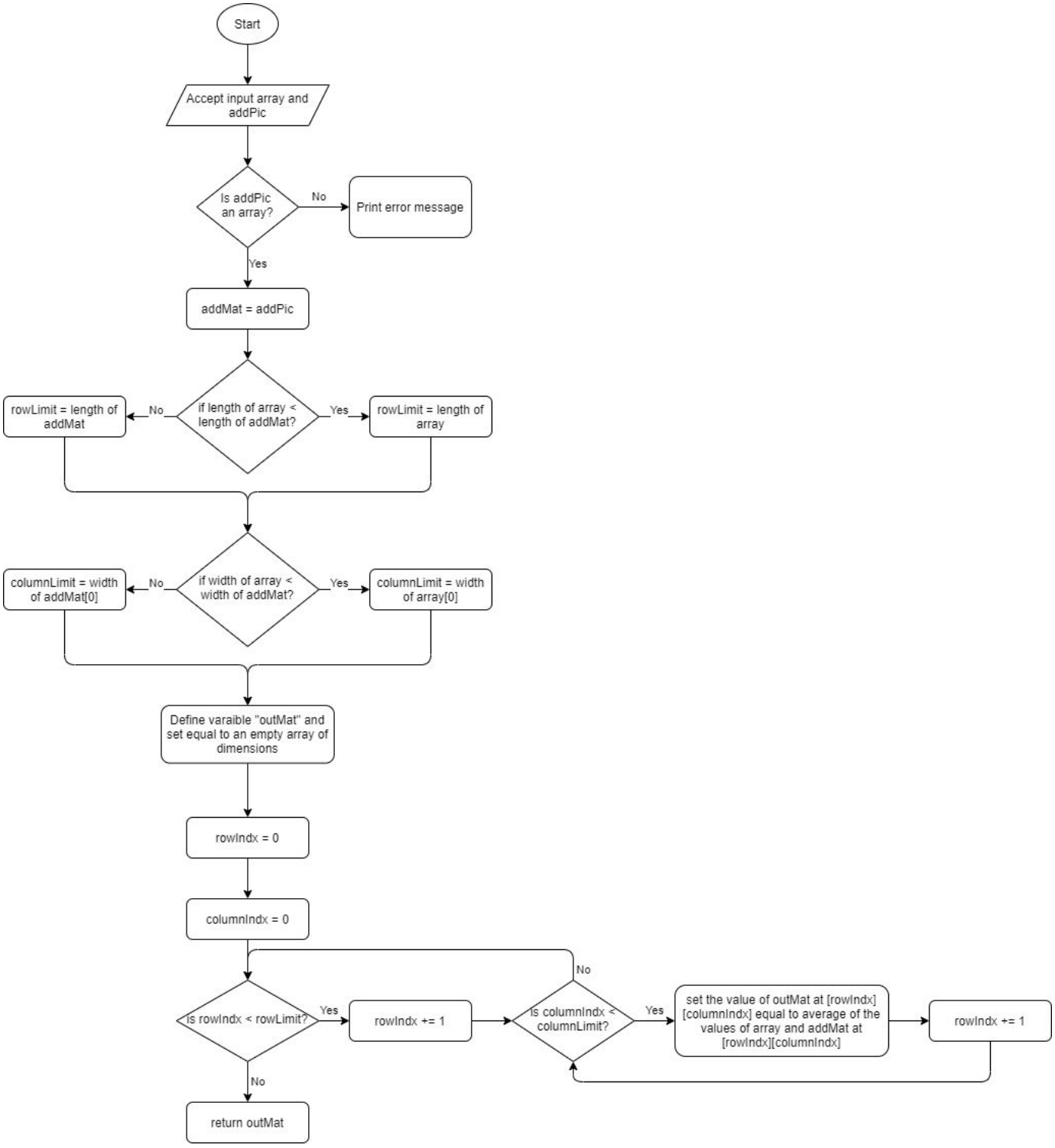
calculateTransformation:



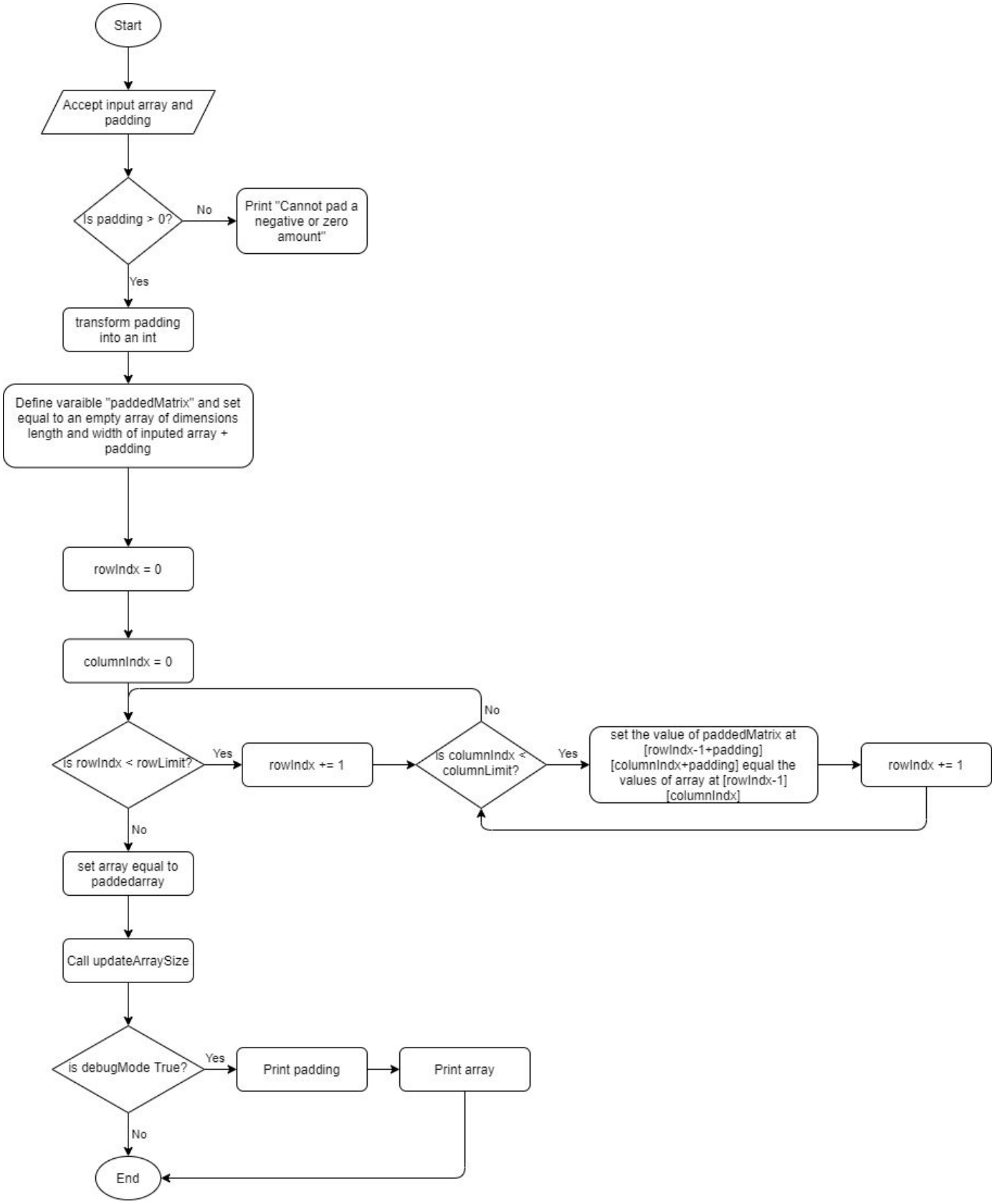
Enhance:


Start Enhance
Given: 2D array of image, threshold value
For every row in the 2D array
For every column in the 2D array
Multiply the brightness of a pixel in the array by 2
If the Brightness of a pixel is less than the threshold value
True
The Pixel becomes white
False
The Pixel becomes black
Stop Threshold

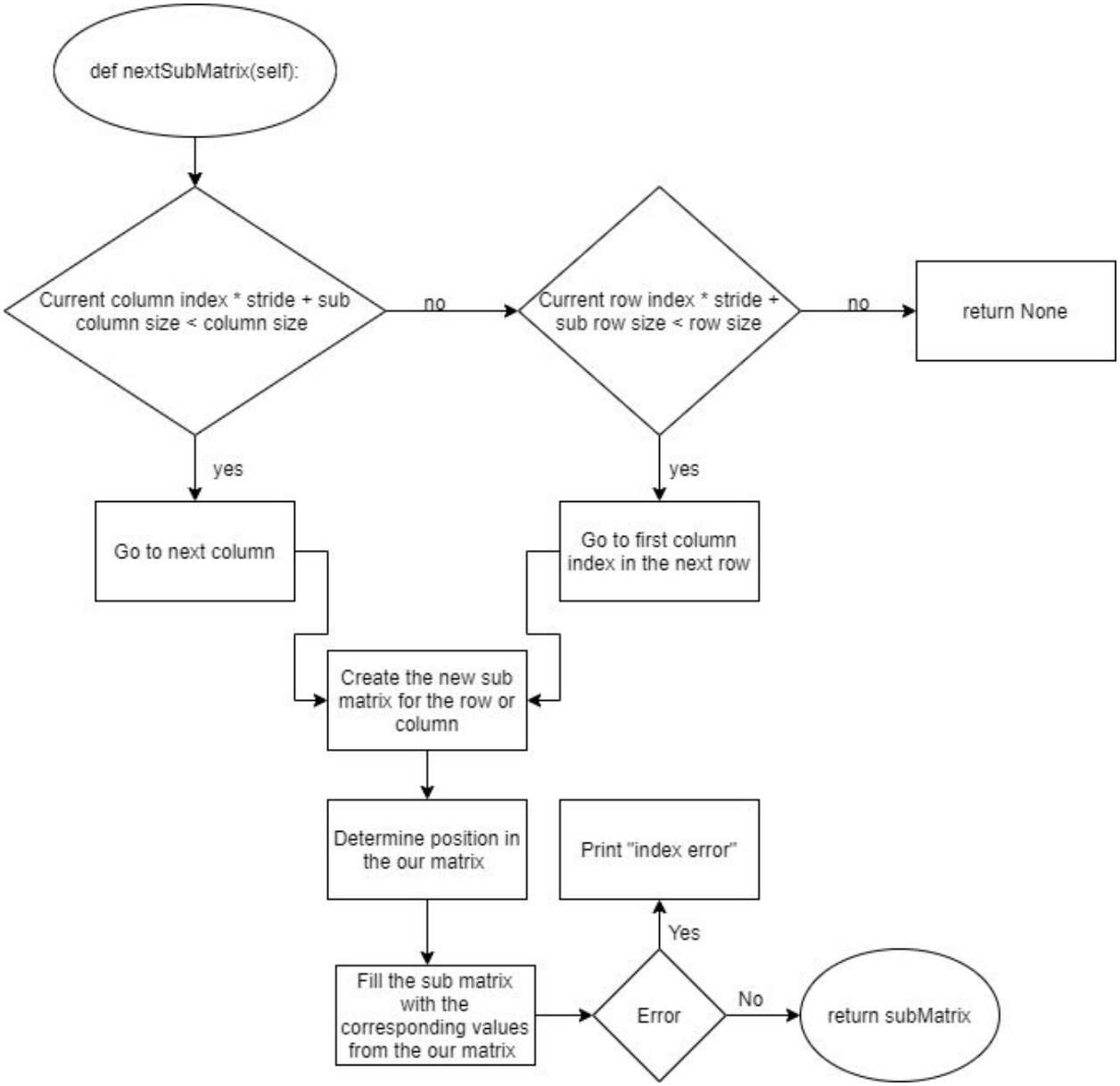
__add__:


Start
Accept input array and addPic
Is addPic an array?
No
Print error message
Yes
addMat = addPic
if length of array < length of addMat?
No
rowLimit = length of addMat
Yes
rowLimit = length of array
if width of array < width of addMat?
No
columnLimit = width of addMat[0]
Yes
columnLimit = width of array[0]
Define varaible "outMat" and set equal to an empty array of dimensions
rowIndx = 0
columnIndx = 0
Is rowIndx < rowLimit?
Yes
rowIndx += 1
Is columnIndx < columnLimit?
Yes
set the value of outMat at [rowIndx][columnIndx] equal to average of the values of array and addMat at [rowIndx][columnIndx]
rowIndx += 1
No
No
return outMat

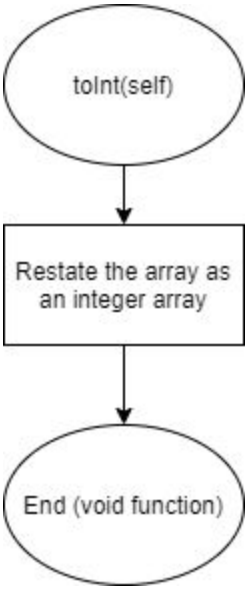
Pad:


Start
Accept input array and padding
Is padding > 0?
No
Print "Cannot pad a negative or zero amount"
Yes
transform padding into an int
Define varaible "paddedMatrix" and set equal to an empty array of dimensions length and width of inputed array + padding
rowIndx = 0
columnIndx = 0
Is rowIndx < rowLimit?
Yes
rowIndx += 1
Is columnIndx < columnLimit?
No
Yes
set the value of paddedMatrix at [rowIndx-1+padding] [columnIndx+padding] equal the values of array at [rowIndx-1] [columnIndx]
rowIndx += 1
No
set array equal to paddedarray
Call updateArraySize
is debugMode True?
Yes
Print padding
Print array
No
End

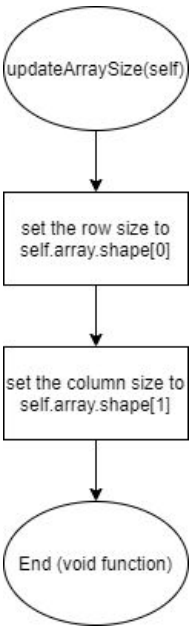
nextSubMatrix:



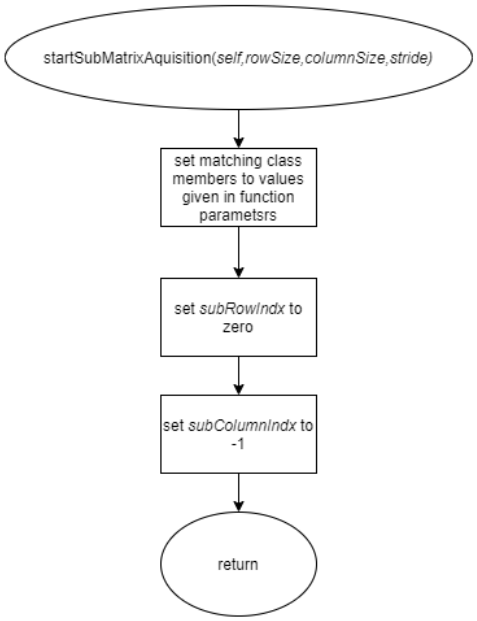
toInt:



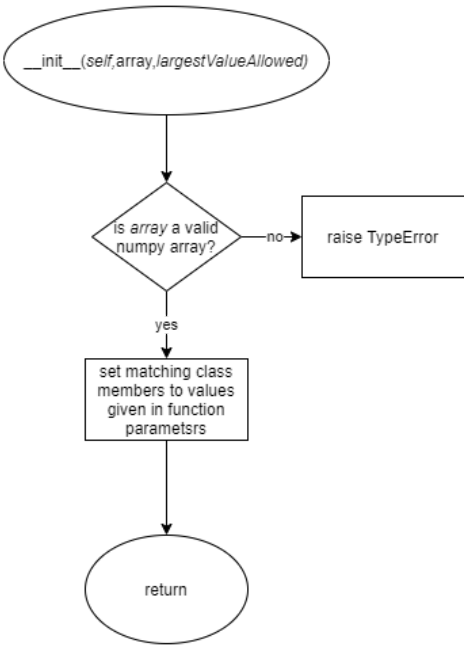
updateArraySize:



startSubmatrixAcquisition:



__init__:



The input, output and grayscale functions are each unique and the code function for other transformations accepts an array input. This makes the code extremely easy to mix in new functionality so long as any newly desired transformations are inputted with the correct modification array. If a different method for edge detection is desired, the user can input the array associated with that different method into the transformation function and the program will modify the image array accordingly. As an example of the program's modularization, if a user only wanted to see the grayscale version of the image then they could write the code to only use the input, grayscale, and output functions and the program would output the desired image. The program helps assist the user by checking to see if the user's input has ".png" at the end and appends it if the input does not. Otherwise the program returns an error that describes that the filename cannot be found or make sure the file is a png and then exits the program. A similar process happens when the user inputs the output file name. Modifying the Edge detection parameters is also made easy by the simplicity in the input of the edge detection matrices and threshold. Simply adjust the values for xEdgeDetectectArray, yEdgeDetectectArray and blurTransArray to get a different process and the user will input the desired threshold. This makes the process extremely simple and easy to modify. The program outputs a png file of the edge detected image and a plot to Spyder for observation. The code can also be modified easily to return the array of the image if further processes want to be done on the edge detected image.

The MyTrix function was designed to have maximum reusability in the form of a wrapper class around a numpy array. We chose to do this so that we could have one module that completely managed the numpy matrix in a single encapsulated file, which we could individually send transformation arrays to be applied to the matrix. Arguably, the two most common ways of doing this is by either storing the numpy matrix in a global variable or using a class and storing the matrix as a class member. We chose to use the class implementation because it was a cleaner way of having many related functions that operate on the same data. However we recognize that the global variable approach would have a similar implementation and end result.

The MyTrix function additionally is designed to be as little implementation-specific as possible, and thus all the parameters can be adjusted to work with any type or matrix, not just picture or integer values, and it can use transformation arrays of any size, including rectangular matrices rather than square ones.
For non transformation applications, the sub-matrix iteration methods can be used in isolation from or with the transformation methods.

In addition to this, little to no functionality is lost due to the matrix being in a wrapper class due to the overloading of the __add__, __getitem__, and __str__ functions for MyTrix so most basic array functionality can be accessed directly from the wrapper class. For all other functions and

methods of the numpy matrix, the actual numpy array can be extracted from the MyTrix object by simply using MyTrix.array .

References:

P. G. Michalopoulos, "Vehicle detection video through image processing: the Autoscope system," in IEEE Transactions on Vehicular Technology, vol. 40, no. 1, pp. 21-29, Feb. 1991, doi: 10.1109/25.69968.

Hoose, N. *Computer image processing in traffic engineering*. United States: N. p., 1991. Web.https://www.osti.gov/biblio/5000574

Marana, A.N.; Velastin, S.A.; Costa, L.F.; Lotufo, R.A.: 'Estimation of crowd density using image processing', IET Conference Proceedings, 1997, p. 11-11, DOI: 10.1049/ic:19970387 IET Digital Library, https://digital-library.theiet.org/content/conferences/10.1049/ic_19970387

Appendix:

[Video Link]

User Manual:

See User Manual.pptx

Project Management Plan:

Sam Graham: Wrote the input and output functions for the main code and did the respective flowcharts and comments. Did the flowcharts for the ___add___ and pad functions in MyTrix. Wrote the Modularization and project Management plan collaboration paragraphs as well as the Project Motivation.

James Long: Grayscale function code with comments, flowcharts for grayscale function, threshold function, and enhance function.

Alec Pannunzio: Transformation code with comments, converting the grayscale image to the outline image, flowcharts for:

Jonathan Dufresne: Main code with comments. Flow charts for Main and MyTrix functions nextSubMatrix, updateArraySize, and toInt.

Methods: When the project was assigned we essentially divided up the code work and agreed whomever took the more difficult sections would have to do less of the report. From there we split to work on our assigned sections, asking for help from the other team members when needed. After working on the code individually for a period of time we convened to test how each function interacted and make changes to have everything run smoothly. This gave each member the opportunity to learn about and "own" a particular part of the project, and then return to the group setting to teach the other group members what process they used. For the sharing of files we initially attempted to use GitHub, but after some complications stemming from the majority of the group's inexperience with it we defaulted to google drive instead. Our collaboration process was initially hampered by the absence of one of our group members on the 1st day of the project, along with the rest of the groups over willingness to begin work. We also had some communication issues between what exactly each other's code was taking for inputs and outputs, making us spend extra time explaining our code to each other. These issues can be remedied by us having stronger and more frequent communication in the future.

Discussion of Design Process:

For our design process, we started by outlining all the different problems we would need to solve and ended up separating the code into four main sections: Input/Output, Grayscale, MyTrix, and Main. Input/Output controlled the imported and exported images, along with handling error messages that a user might encounter. Grayscale was responsible for taking the 3D array of the original image and converting it to a 2D array of the grayscale version. MyTrix was responsible for controlling all the various transformations used to convert the grayscale image into the edge defined image that we then output. Finally, Main is used to tie the rest of the code together so it runs in a much more simple way. We assigned each person with a main role or focus for the project, but we also periodically went and worked on other aspects of the code, to ensure that there were multiple perspectives to every problem and solution, and also to ensure that every member of the team understood the code as a whole. During our group meetings, we would find problems with our code, go off on our own to try and solve those problems independently, and then meet back up to compare and combine our solutions so that they are optimized.

Code:

Main:

```
from MyTrix import MyTrix
import Input_Output
from inputToolbox import getIntegerFromUser
from Grayscale import grayscale
from matplotlib import pyplot
```

```
def runPicture(fileName,thresh=20,outputFile = "EdgeDetectedImage.png"):

    raw_image = Input_Output.inputImage(fileName);
    pyplot.figure(1)
    pyplot.imshow(raw_image)
    grayScale_image = grayscale(raw_image);



    '''
    NOTE: Not all methods in MyTrix are used. The ones that are used are outlined in the
flowcharts
    '''
    picture = MyTrix(grayScale_image);


    '''
    these are the matrices for the transformations that we do
    '''
    blurTransArray = [[1,1],[1,1]];

    xEdgeDetectArray = [[-1,0,1],[-2,0,2],[-1,0,1]];

    yEdgeDetectArray = [[-1,-2,-1],[0,0,0],[1,2,1]];



    print("Blurring");
    picture.transform(blurTransArray,0);
    picture.toInt();
    pyplot.figure(2)
    Input_Output.outputImage(picture.array);


    print("Enchancing")
    picture.enhance(10);
    pyplot.figure(3)
    Input_Output.outputImage(picture.array);


    #pad the array before we do edge detection
    picture.pad();
```

```
    print("X Edge Detection");
    xEdgeDetectTransMat = picture.calculateTransformation(xEdgeDetectArray);
    xEdgeDetectTransformed = MyTrix(xEdgeDetectTransMat);
    xEdgeDetectTransformed.toInt();
    pyplot.figure(4)
    Input_Output.outputImage(xEdgeDetectTransformed.array);

    print("Y Edge Detection");
    yEdgeDetectTransMat = picture.calculateTransformation(yEdgeDetectArray);
    yEdgeDetectTransformed = MyTrix(yEdgeDetectTransMat);
    yEdgeDetectTransformed.toInt();
    pyplot.figure(5)
    Input_Output.outputImage(yEdgeDetectTransformed.array);

    picture.array = (xEdgeDetectTransformed + yEdgeDetectTransformed);

    print("Combining");
    pyplot.figure(6)
    Input_Output.outputImage(picture.array);


    print("Thresholding");
    picture.threshold(thresh);
    pyplot.figure(7)
    Input_Output.outputImage(picture.array,outputFile);

    print("Done");

runPicture(input("What is the fileName?"),getIntegerFromUser("What threshold do you want to
use? (must be between 0 and 255 to do anything) ","threshold must be a valid integer between
0 and 255 to do anything "),input("What fileName do you want to output the file to?"));
#runPicture("coins",20,"edgyCoins.png");
#runPicture("Purdue_Arch",8,"edgyArch.png");
#runPicture("cheerios.png",5,"edgycheerios.png");
#runPicture("CMOS.png",15,"edgyCMOS.png");
```

Grayscale:

```
# Import Numpy to gain access to numpy arrays
import numpy as np

#user defined function to grayscale the image
def grayscale(im):
    #define y, which is used as a generic value for the brightness of pixels
```

```
    i = 0
    #create a blank 2D array that is the same xy shape as the image
    gray = np.zeros((im.shape[0],im.shape[1]))

    #go through every row in the array and go through every column in the array
    #define the grayscale value of each pixel and set the corresponding value in teh blank array
equal to the grayscale brightness
    for x in range(0,len(im)):
        for y in range(0,len(im[0])):
            i = 0.2126 * im[x,y,0] + 0.7152 * im[x,y,1] + 0.0722 * im[x,y,2]
            gray[x,y] = i

    #Show the user the shape of the image
    print(f'File Shape is {np.shape(im)}')


    return gray
```

Input/Output:

```
import numpy as np #Importing numpy for matrix use
from matplotlib import image #Importing the functinos of matplotlib that can transform an image
into a matrix and back again along with other various functions
from matplotlib import pyplot #Importing the functions from matplotlib that allow us to plot the
image in Spyder
import sys #Imports sys so that we can exit the code upon an error occuring


def inputImage(s):
    try: #Tries the following line of code and if an exception or error is returned then it moves on
to the code under except
        imMat = image.imread(s) #The imread function transforms an inputed image into a 3d
matrix where each [x][y] is home to a list of 3 values from 0-1 representing RGB values
    except FileNotFoundError: #This is where the errors are handled, tests to see if user forgot
".png", and if the file cannot be found, then it returns an error message
        try:
            s = (s+".png")
            imMat = image.imread(s)
        except FileNotFoundError:
            print("[Error]:file not found, look in directory or re-check spelling")
            sys.exit()
    imMat = imMat * 255 #Multiplies the 3rd dimension of the matrix (the one containing the 0-1
float RGB values) by 255 for later integer conversion
    imMat = np.around(imMat) #Rounds the RBG values to the nearest whole number, still
returns values as floats
```

```
    imMat = imMat.astype(int) #Transforms the rounded float RGB values to integers
    return imMat #Returns the array containing the image information

def outputImage(s, imgName = "EdgeDetectedImage.png"):
    x = np.zeros((s.shape[0],s.shape[1],3)) #Creates an array of equal size to that of "s", except
each [x][y] coordinate is a list of 3 values, all filled with zeros
    for a in range(0,s.shape[0]): #Loops through the [x] dimension of the array
        for b in range(0, s.shape[1]): #Loops through the [y] dimension of the array
            for c in range(0,3): #Loops through the [z] dimension of array x
                x[a][b][c] = s[a][b] #Sets the values of every number in the list at[x][y] in array x equal
to the value of the array at [x][y] in s
    x = x/255.0 #Divides every number in the list by 255.0, converting them to float point form
    pyplot.imshow(x); #plots the image
    try: #Attempts to save the image with the inputted file name
        image.imsave(imgName,x); #Uses the given matrix of dimensions [x][y][z] to write an
image file that is x pixels wide by y pixels tall with color at each pixel determines by the RGB
values in z
    except: #If the file name doesn't work then the function will attempt to append ".png" to the
end and try to save once more before returning an error
        try:
            imgName = imgName + ".png"
            image.imsave(imgName,x);
        except:
            print("Error: Please enter a valid file name (No spaces or special characters)")
```

MyTrix:

```
import numpy as np
from math import floor

class MyTrix():
    def __init__(self,array, largestValueAllowed = 255): #initialize a new MyTrix object with a
numpy array

        if not (type(array) is np.ndarray): #make sure the user entered in an array we can work with
                raise TypeError("MyTricks must be initialized with a numpy array");
        else:
            # store the passed array as one of our class members
            # self.array is the main data that this class will work with
            self.array = array;

            # debug mode is whether or not we should output verbose updates
            self.debugMode = False;

            #the largest value we want elements of this array to have on transformations
```

```
        self.largestValueAllowed = largestValueAllowed;

    # just a different way of accessing an element in the array
    def at(self,i,j):
      return self.array[i][j];

    def __str__(self):
        return str(self.array);

    def __getitem__(self,key):
        return self.array[key];

    def __add__(self,addPic): #adds to MyTrixes together

        # make sure addPic is another MyTrix
        try:
            addMat = addPic.array;
        except TypeError:
            print("Cannot add type", type(addPic), "and Picture");


        #figure out which matrix is smaller and limit the indices accordingly
        if (len(self.array) < len(addMat)):
            rowLimit = len(self.array);
        else:
            rowLimit = len(addMat);


        if (len(self.array[0]) < len(addMat[0])):
            columnLimit = len(self.array[0]);
        else:
            columnLimit = len(addMat[0]);

        #create an output array with the smaller of the two mat's dimensions
        outMat = np.empty((rowLimit,columnLimit));

        for rowIndx in range(0,rowLimit):
            for columnIndx in range(0,columnLimit):
                outMat[rowIndx][columnIndx] = (self.array[rowIndx][columnIndx] +
addMat[rowIndx][columnIndx])/2;
        return outMat;

    #sets whether to give verbose progress logs or not
    def setDebugMode(self, debugMode):
```

```
        self.debugMode = debugMode;

    #returns a copy of the array
    def getArrayCopy(self):
        return np.arrCopy(self.array);


    #converts all of the elements of the matrix to integers
    def toInt(self):
        self.array = self.array.astype(int);
        return self.array;

    #will make all values smaller than the threshold lowVal and larger than threshold highVal
    def threshold(self,thresh = 155, highVal = 255,lowVal = 0):

        # iterate through all of the elements
        for rowIndx in range(0,len(self.array)):
            for columnIndx in range(0,len(self.array[0])):


                if (self.array[rowIndx][columnIndx] >= thresh):
                    # if this element is larger than the threshold then set it to highval
                    self.array[rowIndx][columnIndx] = highVal;
                else:
                    #otherwise set it to lowVal
                    self.array[rowIndx][columnIndx] = lowVal;


    #will make all values smaller than the threshold lowVal and larger than threshold highVal
    def enhance(self, threshold):

        # iterate through all of the elements
        for rowIndx in range(0,len(self.array)):
            for columnIndx in range(0,len(self.array[0])):

                if (self.array[rowIndx][columnIndx] < threshold):
                    self.array[rowIndx][columnIndx] = int((self.array[rowIndx][columnIndx])/2);

    #initializes the process of parcing through the MyTrix by sub-matricies of the specified size
    def startSubMatrixAquisition(self,rowSize, columnSize, stride=1):

        # set the settings members for the subMatrix aquisition to what was passed into the
function
        self.subRowSize = rowSize;
```

```
        self.subColumnSize = columnSize;

        self.stride = stride; # stride is how many elements we iterate each subMatrix



        # bring the iteration variables back to their starting position
        self.subRowIndx = 0;
        self.subColumnIndx = -1; #we start at -1 because we increment the indeces at the start of
the nextSubMatrix method



    # will return the next submatrix of the specified size. (startSubMatrixAquisition() must be
called before this method for it to work properly)
    def nextSubMatrix(self):

        #advance to the next subMatrix indices
        if ((self.subColumnIndx*self.stride) + self.subColumnSize < len(self.array[0])): #have we
gone to the end of this row?
                self.subColumnIndx += 1; #advance to the next column in this row

        elif ( (self.subRowIndx*self.stride) + self.subRowSize < len(self.array)): #have we done all
of the rows?
            self.subColumnIndx = 0; #go to the first column
            self.subRowIndx += 1; #advance to the next row
        else:
            return None;



        #create the submatrix to fill with values
        subMatrix = np.zeros((self.subRowSize,self.subColumnSize));



        try:
            #determine where to start grabbing values from the our matrix
            startRowIndx = self.subRowIndx * self.stride;
            startColumnIndx = self.subColumnIndx * self.stride;

            #fill the submatrix with our matrix's values
            for relativeRowIndx in range(0,self.subRowSize):
                for relativeColumnIndx in range(0,self.subColumnSize):
                    subMatrix[relativeRowIndx][relativeColumnIndx] = self.at(startRowIndx +
relativeRowIndx, startColumnIndx + relativeColumnIndx);
```

```
        except IndexError:
            print("index error (nextSubMatrix)");
            pass;

        return subMatrix;


    def replaceSubMatrix(self,newArray):

        try:
            #calculate where the matrix to replace will start
            startRowIndx = self.subRowIndx * self.subRowSize;
            startColumnIndx = self.subColumnIndx * self.subColumnSize;


            #figure out which matrix is smaller and limit the indices accordingly
            if (self.subRowSize < len(newArray)):
                rowLimit = self.subRowSize;
            else:
                rowLimit = len(newArray);

            if (self.subColumnSize < len(newArray[0])):
                columnLimit = self.subColumnSize;
            else:
                columnLimit = len(newArray[0]);

            for relativeRowIndx in range(0,rowLimit):
                for relativeColumnIndx in range(0,columnLimit):
                    self.array[startRowIndx + relativeRowIndx][startColumnIndx + relativeColumnIndx]
= newArray[relativeRowIndx][relativeColumnIndx];

                relativeColumnIndx = 0;

        except IndexError:
            print("index error (replaceSubMatrix)");

    # pads this array by adding [padding] layers of zeros around it
    def pad(self,padding = 1):
        if (padding > 0):
            padding = int(padding);
            paddedMatrix = np.zeros((len(self.array)+2*padding,len(self.array[0])+2*padding));

            for rowIndx in range(0,len(self.array)):
```

```
            for columnIndx in range(0,len(self.array[0])):
                paddedMatrix[rowIndx+padding][columnIndx+padding] =
self.array[rowIndx][columnIndx];

        self.array = paddedMatrix;

        if (self.debugMode):
            print(f"Padded Matrix {padding} layers: ");
            print(self.array);
    else:
        print("cannot pad a negative or zero amount.");



# calculates the transformation of this matrix by the passed transArray
def calculateTransformation(self,transArray,stride=1):


    #get the lengths of our array and the transformation array
    nR = len(self.array);
    nC = len(self.array[0]);
    fR = len(transArray);
    fC = len(transArray[0]);

    #calculate output matrix size
    outLenR = int(floor( (1 + nR - fR) / stride ) );
    outLenC = int(floor( (1+ nC - fC) / stride ) );


    #create a numpy matrix
    outMatrix = np.zeros((outLenR,outLenC));

    if (self.debugMode):
        print(f"outSize = int(floor( (1 + {nR} - {fR}) / {stride} ) )")
        print("OutMatrix before");
        print(outMatrix);

    #start subMatrix stepping
    self.startSubMatrixAquisition(len(transArray),len(transArray[0]),stride);


    subMatrix = self.nextSubMatrix();
    while (type(subMatrix) != type(None)):
```

```
            dotProduct = 0;
            for rowIndx in range(0,len(subMatrix)):
                for columnIndx in range(0,len(subMatrix[0])):
                    dotProduct += transArray[rowIndx][columnIndx] * subMatrix[rowIndx][columnIndx];

            outMatrix[self.subRowIndx][self.subColumnIndx] = abs(dotProduct/(fR*fC)); #put the dot
product of the transformation in the output matrix



            if (self.debugMode):
                print("SubMatrix:");
                print(subMatrix);
                print(f"Dot product: {dotProduct}");

            subMatrix = self.nextSubMatrix();

        return outMatrix;



    #transforms the picture
    def transform(self,transArray,padding = 0, stride = 1):
        if (padding > 0):
            self.pad(padding);

        self.array = self.calculateTransformation(transArray,stride);



#this is a testing method not used in any project
def testMyTrix():
    #create a 2D numpy array
    a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13,14,15,16]]);

    #initialize a MyTricks with the numpy array
    picture = MyTrix(a);



    #turn on debug mode so we can see what is happening
    picture.setDebugMode(True);

    #MyTrix manipulation should work the same as normal array manipulations at base levels
    print("rowSize:", picture.rowSize);
```

```
print("columnSize:", picture.columnSize);
print("\n Matrix:")
print(picture);
print(picture.at(1,1));
print(picture[0][0]);
print(picture[0]);
picture[0][0] = -1;

print("\n Matrix with single replacement:")
print(picture)

print("\n")

#test subMatrix stepping
picture.startSubMatrixAquisition(2,2);
print("Sub matrices:");

subMatrix = picture.nextSubMatrix();
while (type(subMatrix) != type(None)):
    print(subMatrix);
    subMatrix = picture.nextSubMatrix();

#test subMatrix replacement
print("\nMatrix with replacement:");
picture.replaceSubMatrix([[1,-2,3],[4,5,6],[7,8,9]]);
print(picture);


#test transformation with padding 1
print("\n Test transformation:");
picture.transform([[-1,2],[3,4]],1);
print(picture);


#test transformation with padding 2 and larger transformation
print("\n Test transformation:");
picture.transform([[1,-2,3],[4,5,6],[7,8,9]],2);
print(picture);
```