**Problem Set 5 Report**

1. Greyscale function



2. Correlation function. Note: I have two correlation functions, one that takes in a filter value and a dimension, and one that takes in a 1D array. Both are basically the same.

/\* Correlates a filter value on an image. This function takes in a greyscale BMPImage, and a separable filter with n x n dimensions and a uniform filter value. The filter value parameter should be the value before separation. For instance, if the box filter has all values of 1/49, then 1/49 should be used as input, as this function will separate the box filter into two 1D filters with the value of 1/7. The function then applies the two 1D filters to the image.\*/

void correlationFunction(BMPImage \*image, float filter, int n) {

float readVal, writeVal, filterVal;

filterVal = sqrt(filter);

//for each pixel in the image

for (int y = 0; y < image->getYSize(); y++) {

for (int x = 0; x < image->getXSize(); x++) {

writeVal = 0;

readVal = 0;

//filter across x

for (int i = -1 \* (n / 2); i <= (n / 2); i++) {

int offsetXCoord = x + i;

//padding left

if (offsetXCoord <= 0) {

//use left most value in row

image->readPixel(0, y, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

//padding right

else if (offsetXCoord >= image->getXSize() - 1) {

//use right most value in row

image->readPixel(image->getXSize() - 1, y, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

else {

//use value at offset in row

image->readPixel(offsetXCoord, y, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

}

clampValues(&writeVal);

image->writePixel(x, y, writeVal, writeVal, writeVal);

}

}

//for each pixel in the image

for (int y = 0; y < image->getYSize(); y++) {

for (int x = 0; x < image->getXSize(); x++) {

writeVal = 0;

readVal = 0;

//filter across y

for (int i = -1 \* (n / 2); i <= (n / 2); i++) {

int offsetYCoord = y + i;

//padding top

if (offsetYCoord <= 0) {

//use top most value in column

image->readPixel(x, 0, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

//padding bottom

else if (offsetYCoord >= image->getYSize() - 1) {

//use top most value in column

image->readPixel(x, image->getYSize() - 1, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

else {

//use value at offset in column

image->readPixel(x, offsetYCoord, readVal, readVal, readVal);

writeVal += readVal \* filterVal;

}

}clampValues(&writeVal);

image->writePixel(x, y, writeVal, writeVal, writeVal);

}

}

}



1. Gaussian function

/\* Creates a gaussian filter. This function takes in a value for sigma and a pointer to an integer. The function will create a new 1D filter and populate it with values for the gaussian filter. The function will return a pointer to the 1D filter, and will modify the integer pointer to reflect the dimension of the filter. \*/

float\* filterGaussianFunction(float sigma, int \*dimen) {

const float PI = 3.1415927;

int filterDimen = 2 \* (ceilf(3 \* sigma)) + 1;

float \*filter = new float[filterDimen];

int j = 0; //index in the filter

//populate the filter

for (int i = -1 \* (filterDimen / 2); i <= (filterDimen / 2); i++) {

//calculate filter value and place in filter

filter[j] = sqrt(1 / (2 \* sigma \* sqrt(2 \* PI))) \*exp(-1 \* (i \* i) / (2 \* sigma \* sigma));

j++;

}

//normalize the filter values

float sum = 0.0f;

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

sum += filter[i];

}

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

filter[i] = (filter[i] / sum);

}

\*dimen = filterDimen; //"return" the dimension of the filter

return filter; //return the filter

}



1. Sharpening function

/\* Creates a sharpening filter. This function takes in a value for sigma, a pointer to a 1D Gaussian filter and the dimension of the filter. The function will subtract the Gaussian filter from an all-pass filter with a center value of 2, with the rest being zero. The function returns the filter T-G. \*/

float\* filterSharpeningFunction(float sigma, float \*filter, int dimen) {

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

if (i == (dimen / 2)) {

filter[i] = 2 - filter[i]; //center value of filter T is 2, subtract the center gaussian filter value

}

else {

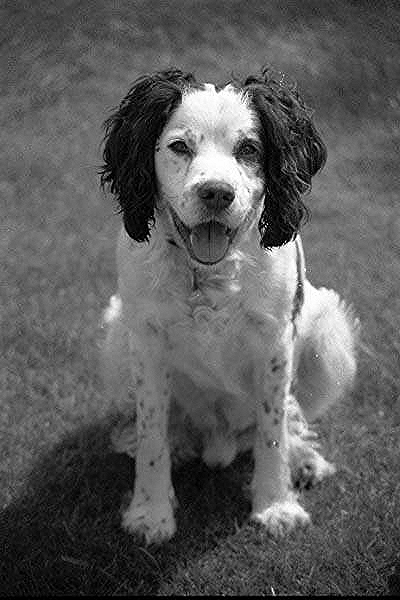
filter[i] \*= -1.0f; //all other values of T are 0, so subtract gaussian filter value at this position

}

}

return filter;

}



1. Resize function

/\* Resizes an image using bilinear interpolation to fill in pixel values. The function will look at surrounding pixels and interpolate the color for a given pixel. \*/

void resizeFunction(BMPImage \*image, float scale, char \*saveName) {

float temp1 = image->getXSize() \* scale;

float temp2 = image->getYSize() \* scale;

/\* compute the image size \*/

int remainder = (int)fmodl(temp1, 4L);

if (remainder != 0) {

temp1 += (4 - remainder);

}

BMPImage scaledImage = BMPImage(temp1, temp2);

int xLeft, xRight, yTop, yBot;

float topLeftVal, topRightVal, botLeftVal, botRightVal;

float weightX, weightY;

float leftBorderVal, rightBorderVal;

float finalVal;

//for all pixels in the scaled image

for (int scaledX = 0; scaledX < scaledImage.getXSize(); scaledX++) {

for (int scaledY = 0; scaledY < scaledImage.getYSize(); scaledY++) {

//get coordinates of surrounding pixels in original image

xLeft = scaledX / scale;

//make sure we don't exceed bounds of image

if (xLeft == image->getXSize() - 1) {

xRight = xLeft;

}

else {

xRight = xLeft + 1;

}

yTop = scaledY / scale;

//make sure we don't exceed bounds of image

if (yTop == image->getYSize() - 1) {

yBot = yTop;

}

else {

yBot = yTop + 1;

}

//get color value at surrounding pixels in original image

image->readPixel(xLeft, yTop, topLeftVal, topLeftVal, topLeftVal); //top left value

image->readPixel(xRight, yTop, topRightVal, topRightVal, topRightVal);

//top right value

image->readPixel(xLeft, yBot, botLeftVal, botLeftVal, botLeftVal); //bot left value

image->readPixel(xRight, yBot, botRightVal, botRightVal, botRightVal);

//bot right value

//get weight in the y direction

weightY = (scaledY / scale) - yTop;

//get color value on left border at the y value

leftBorderVal = (weightY \* (botLeftVal - topLeftVal)) + topLeftVal;

//get color value on right border at the y value

rightBorderVal = (weightY \* (botRightVal - topRightVal)) + topRightVal;

//get weight in the x direction

weightX = (scaledX / scale) - xLeft;

//get color between the two weighted border values

finalVal = (weightX \* (rightBorderVal - leftBorderVal)) + leftBorderVal;

scaledImage.writePixel(scaledX, scaledY, finalVal, finalVal, finalVal);

}

}

scaledImage.save(saveName);

}

Dog scaled by 3:



Dog scaled by 0.75



Rug scaled by 0.05 Rug sigma 10 Gaussian and then scaled by 0.05

