Bildverarbeitung – Übungsblatt 2

Sebastian Kürten, Max Michels

Aufgabe 1

Erweitern der Matrix wurde implementiert in der Funktion “padZero”:

```
function A = padZero(A)
        A = [zeros(size(A,1),1) A];
        A = [zeros(1,size(A,2));A];
        A(end+1,:) = 0;
        A(:,end+1) = 0;
end;
```

Anwenden einer Konvolution auf eine Matrix. “Per Hand” implementiert in der Funktion “falte”:

```
function B = falte(A, k)
        A = int32(A);
        B = zeros(size(A));
        Ap = padZero(A);
        for i = 1 : size(A, 1)
                for j = 1 : size(A, 2)
                        B(i,j) = sum(sum(k .* Ap(i:i+2,j:j+2)));
                end
        end
end;
```

Dementsprechend ist das Ergebnis die Ausgabe von “falte(B, k)”:

```
    49    71    78    76    70    45
    68   103   118   113    99    60
    57    93   113   108    88    48
    28    54    76    74    54    24
     9    22    36    37    25     9
     1     6    13    13     8     1
```

Aufgabe 2

Das absolute Histogramm ist eine Liste der Länge 256. Wir iterieren durch die Pixel des Bilds und erhöhen jeweils den Eintrag des angetroffenen Grauwerts:

```
function H = histogramAbs(A)
        H = zeros(1, 256);
        for i = 1 : size(A, 1) * size(A,2)
                H(A(i) + 1) += 1;
        end
end;
```

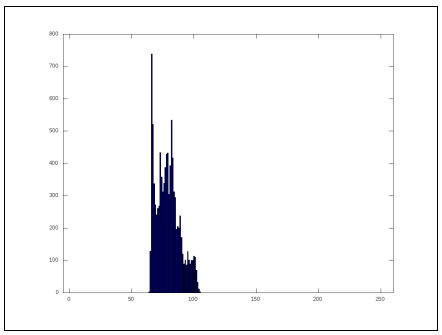
Das relative Histogramm lässt sich aus dem absoluten Histogramm ermitteln, indem durch die Anzahl der Pixel im Bild geteilt wird:

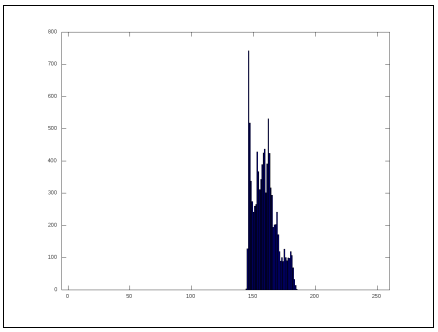
```
function H = histogramRel(A)
        H = histogramAbs(A) / (size(A, 1) * size(A,2));
end;
```

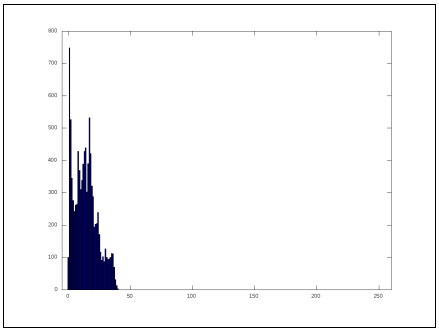
Um das kumulierte relative Histogramm zu erhalten, werden die Werte des Histogramms, beginnend beim ersten, ersetzt durch die Summe aller Einträge bis zum aktuellen Eintrag:

```
function H = histogramRelCum(A)
        H = histogramRel(A);
        for i = 2 : size(H, 2)
                H(i) = H(i) + H(i - 1);
        end
end;
```

Die absoluten Histogramme für die drei Eingabebilder:

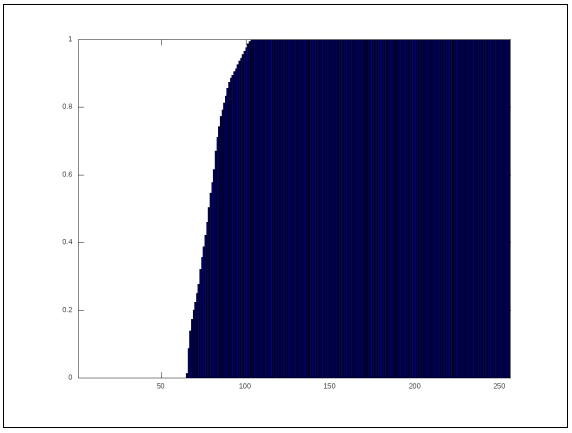


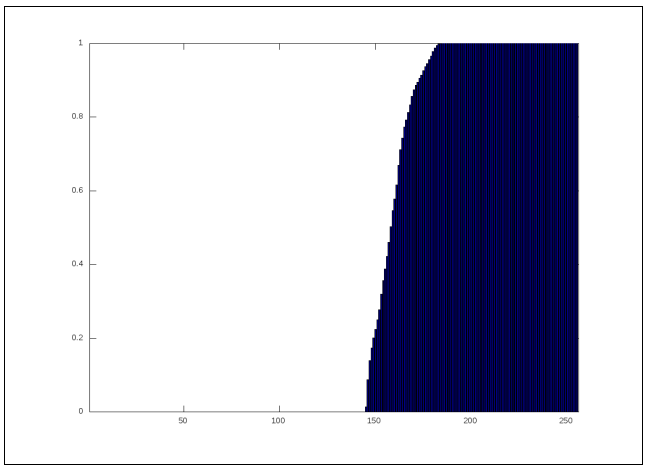


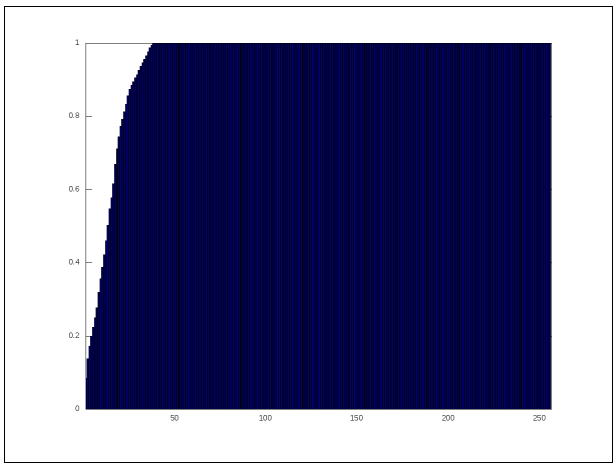


Die relativen Diagramme unterscheiden sich in ihrer Gestalt nicht von den relativen Histogrammen.

Hier die kumulierten Histogramme:







Der Histogrammausgleich wurde in der Funktion “histAusgleich” implementiert. Dabei wird für jeden Pixel der Grauwert des Ausgangsbilds im kumulierten Histogramm nachgeschlagen, mit 255 multipliziert und neu zugewiesen:

```
function A = histAusgleich(A)
        H = histogramRelCum(A);
        for i = 1 : size(A, 1) * size(A,2)
                A(i) = H(A(i) + 1) * 255;
        end
end;
```

Das Ergbenis ist jeweils das gleiche:

Beispielcode:

```
Einlesen eines Bilds: LENA1 = imread("lena-bw-lc1.bmp");
zeichnen eines kumulierten Histograms: bar([0:255], histogramRelCum(LENA1))
erzeugen eines ausgeglichenen Bildes: imwrite(histAusgleich(LENA1), "ausgeglichen1.png");
```

Aufgabe 3

Es wurde das Padding durch Spiegelung für die Erweiterung um einen Pixel implementiert:

```
function A = padMirror(A)
        A = horzcat(A(:,1),A,A(:,end));
        A = vertcat(A(1,:),A,A(end,:));
end;
```

Die Methode zur Konvolution aus Aufgabe 1 wurde angepasst, sodass diese Padding-Funktion benutzt wird:

```
function B = falte(A, k)
        A = int32(A);
        B = zeros(size(A));
        Ap = padMirror(A);
        for i = 1 : size(A, 1)
                for j = 1 : size(A, 2)
                        B(i,j) = sum(sum(k .* Ap(i:i+2,j:j+2)));
                end
        end
end;
```

Die Ergebnisse entsprechen den Resultaten auf der Website und werden hier daher nicht nocheinmal aufgeführt. Folgende Matrizen wurden zur Anwendung der Filter verwendet:

```
Identity = 1/9 * [0 0 0; 0 9 0; 0 0 0]; %zum Testen, die Identität
Mittelwert = 1/9 * [1 1 1; 1 1 1; 1 1 1];
Gauss = 1/16 * [1 2 1; 2 4 2; 1 2 1];
SobelX = [1 0 -1; 2 0 -2; 1 0 -1];
SobelY = [1 2 1; 0 0 0; -1 -2 -1];
Laplace4 = [0 -1 0; -1 4 -1; 0 -1 0];
Laplace8 = [-1 -1 -1; -1 8 -1; -1 -1 -1];
```

So wird beispielsweise der Gaußfilter angewandt:

```
SKY = imread(“skyline-bw.png”);
SKY_GAUSS = falte(SKY, Gauss);
imwrite(uint8(SKY_GAUSS), "sky-gauss.png")
```

Das Bild SobelXY kann erzeugt werden, indem die Ergebnisse von Sobel-X und Sobel-Y auf die folgende Art und Weise kombiniert werden:

```
SKY_SOBELX = falte(SKY, SobelX);
imwrite(uint8(SKY_SOBELX / 8 + 127), "sky-sobelx.png");
SKY_SOBELY = falte(SKY, SobelY);
imwrite(uint8(SKY_SOBELY / 8 + 127), "sky-sobely.png");
SKY_SOBELXY = sqrt(SKY_SOBELX .* SKY_SOBELX + SKY_SOBELY .* SKY_SOBELY);
imwrite(uint8(SKY_SOBELXY/1414 * 255), "sky-sobelxy.png");
```

Der Minimum- / Maximum- / Medianfilter wurde folgendermaßen implementiert:

```
function B = maxfilter(A)
        A = int32(A);
        B = zeros(size(A));
        Ap = padMirror(A);
        for i = 1 : size(A, 1)
                for j = 1 : size(A, 2)
                        B(i,j) = max(max(Ap(i:i+2,j:j+2)));
                end
        end
end;

function B = minfilter(A)
        A = int32(A);
        B = zeros(size(A));
        Ap = padMirror(A);
        for i = 1 : size(A, 1)
                for j = 1 : size(A, 2)
                        B(i,j) = min(min(Ap(i:i+2,j:j+2)));
                end
        end
end;

function B = medianfilter(A)
        A = int32(A);
        B = zeros(size(A));
        Ap = padMirror(A);
        for i = 1 : size(A, 1)
                for j = 1 : size(A, 2)
                        P = Ap(i:i+2,j:j+2);
                        B(i,j) = median([P(1,:),P(2,:),P(3,:)]);
                end
        end
end;
```

Die Anwendung des Medianfilters liefert ein Bild, welches sehr nah am Orginalbild, jenem ohne Rauschen, liegt. Der Minimumfilter entfernt das Salz (also die weißen Stellen), aber verstärkt die schwarzen Stellen, weil diese “größer werden”, da diese lokal immer das Minimum darstellen. Der Maximumfilter arbeitet in etwa umgekehrt: er entfernt den Pfeffer (also die schwarzen Stellen), aber verstärkt das weiße Rauschen.