|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color Image Processing** | **HW2** | | **何柏昇** | **M11107309** |
| **第一題 景深擴張** | | | | |
| 1. 讀取並顯示對焦在前景(fg)與背景(bg)的兩幅影像，並轉換至 float 格式。 2. 自訂八方向 Laplacian 濾鏡。f = [-1 -1 -1; -1 8 -1; -1 -1 -1] 3. 高通濾波：將兩幅影像由彩色轉換至灰階格式，並分別做 Laplacian 高通濾波後，取絕對值。 4. 分別得到fg\_hipass與bg\_hipass。顯示這兩張圖，如果看不清楚，將灰階值乘3。 5. 製作前景遮罩mask = fg\_hipass - bg\_hipass 6. 將遮罩做「均值濾波」，濾鏡尺寸要很大，才不至於使區塊破碎。 7. 以0為門檻，將前景遮罩二值化。 8. 顯示二值化後的遮罩(可能要調亮)。 9. 根據二值遮罩分別取前景(fg)與背景(bg)的清晰像素，組成景深擴增影像。 10. 顯示並儲存景深擴增影像。 | | | | |
| Laplacian濾鏡 | | | | |
|  | | | | |
| **Result** | | | | |
| Foreground | | Background | | |
|  | |  | | |

|  |  |
| --- | --- |
|  | |
| Foreground-HPF | Background-HPF |
|  |  |
|  | |
| Foreground-BinaryMask | Background--BinaryMask |
|  |  |
| Clear | |
|  | |

|  |
| --- |
| **Code** |
| import os  import numpy as np  import cv2 as cv  from matplotlib import pyplot as plt  # Load images  bg = cv.imread('images/5bg.jpg')  fg = cv.imread('images/5fg.jpg')  # BGR to GRAY, and uint8 to double[0,1]  fg = cv.cvtColor(fg, cv.COLOR\_BGR2GRAY).astype(np.float32)/255  bg = cv.cvtColor(bg, cv.COLOR\_BGR2GRAY).astype(np.float32)/255 |
| 首先使用OpenCV來讀取影像，於函式中輸入影像位置，以及color type：0為gray、1為RGB。  　　接著把影像由RBG轉換成Gray，並將資料型態由int轉為double型態，並且縮放到[0 1]，之後計算時會有較為精準的結果。 |
| # High-pass filter (Laplacian)  laplacian = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])  fg = abs(cv.filter2D(fg,-1,laplacian))  bg = abs(cv.filter2D(bg,-1,laplacian)) |
| 建立一個Laplacian Filter，透過OpenCV提供的filter2D進行卷積，並取絕對值，及可獲得圖像的高頻特徵，也就是圖像輪廓。 |

|  |
| --- |
| # mask = fg\_HPF - bg\_HPF  mask = fg – bg  # Mean filter, kernel size must be bigger.  img\_blur = cv.blur(mask,(20,20))  # Binary the mask  mask\_fg = (img\_blur>0)  mask\_bg = (img\_blur<0) |
| 透過將前景與背景進行相減獲得一個mask。並加圖片透過mean-filter進行平均濾波，必須要注意mean filter的size必須要很大。  　　分別建立前景與背景的二值化遮罩，透過數值的判斷，進行前景與背景得區分。 |
| # Relaod images  bg = cv.imread('images/5bg.jpg')  fg = cv.imread('images/5fg.jpg')  # Clear image with binary-mask  fg\_clear = cv.merge([fg[:,:,0]\*mask\_fg, fg[:,:,1]\*mask\_fg, fg[:,:,2]\*mask\_fg])  bg\_clear = cv.merge([bg[:,:,0]\*mask\_bg, bg[:,:,1]\*mask\_bg, bg[:,:,2]\*mask\_bg])  # clear=fg+bg  clear = fg\_clear+bg\_clear  # Show clear image  fig = plt.figure()  plt.axis('off')  plt.title("Clear")  plt.imshow(cv.cvtColor(clear, cv.COLOR\_BGR2RGB))  if not os.path.exists('images'):      os.mkdir('images')  cv.imwrite('images/ex2.1.2.jpg', clear) |
| 重新載入原圖，分別將前景與背景圖套用各自的遮罩，取得圖像中清晰的區域。  　　並將前景與背景的結果相加，就可以獲得一張擁有全景對焦的影像。  　　最後再將結果顯示並儲存。 |

|  |  |
| --- | --- |
| **第二題 視覺異常模擬** | |
| 1. 紅綠色盲：自行找一張色彩豐富的圖片，將RGB影像轉換至浮點格式，再轉換至LAB空間，將a\*設為0，再轉回RGB空間。 2. 黃藍色盲：將RGB影像轉換至浮點格式，再轉換至LAB空間，將 b\*設為0，再轉回RGB空間。 3. 青光眼：讀取RGB影像的尺寸，利用fspecial函式建立與影像同尺寸的2D高斯濾鏡(Gaussain filter)，sigma值必須很高，才有效果。將濾鏡數值矩陣的每個數值除以其最大值。再將濾鏡點對點乘上影像的RGB值。模擬青光眼患者視野狹窄的現象。 | |
| **Result** | |
| Red-Green color blindness | |
|  | |
| Blue-Yellow color blindness | |
|  | |
| Glaucoma | |
|  |  |
| **Code** | |
| import cv2 as cv  import numpy as np  import os  from matplotlib import pyplot as plt  # Red-Green color blindness  img = cv.imread('images/test.png').astype(np.float32)/255.0  lab = cv.cvtColor(img, cv.COLOR\_BGR2LAB)  lab[:,:,1] \*=0  img = (cv.cvtColor(lab, cv.COLOR\_LAB2BGR))\*255  img = img.astype(np.uint8)  fig = plt.figure()  plt.axis('off')  plt.imshow(cv.cvtColor(img, cv.COLOR\_BGR2RGB))  if not os.path.exists('images'):      os.mkdir('images')  cv.imwrite('images/ex2.2.1.jpg', img) | |
| 首先使用OpenCV來讀取影像，於函式中輸入影像位置，以及color type：0為gray、1為RGB。必須注意，使用OpenCV進行影像讀取，影像的顏色型態為BGR，不同於RBG。  　　讀取後將資料型態由int轉為double型態，並進行正規化壓縮至[0,1]，因為cvtColor使用int型態時，誤差會很大，所以必須要使用double型態，並且有規定輸入的範圍必須為[0,1]。  　　透過cvtColor將BGR轉換成LAB空間，並將a的維度全部設為0，再轉回BRG，就可以模擬紅綠色盲。 | |

|  |
| --- |
| # Blue-Yellow color blindness  img = cv.imread('images/test.png').astype(np.float32)/255.0  lab = cv.cvtColor(img, cv.COLOR\_BGR2LAB)  lab[:,:,2] \*=0  img = (cv.cvtColor(lab, cv.COLOR\_LAB2BGR))\*255  img = img.astype(np.uint8)  fig = plt.figure()  plt.axis('off')  plt.imshow(cv.cvtColor(img, cv.COLOR\_BGR2RGB))  if not os.path.exists('images'):      os.mkdir('images')  cv.imwrite('images/ex2.2.2.jpg', img) |
| 首先使用OpenCV來讀取影像，於函式中輸入影像位置，以及color type：0為gray、1為RGB。必須注意，使用OpenCV進行影像讀取，影像的顏色型態為BGR，不同於RBG。  　　讀取後將資料型態由int轉為double型態，並進行正規化壓縮至[0,1]，因為cvtColor使用int型態時，誤差會很大，所以必須要使用double型態，並且有規定輸入的範圍必須為[0,1]。  　　透過cvtColor將BGR轉換成LAB空間，並將b的維度全部設為0，再轉回BRG，就可以模擬藍黃色盲。 |

|  |
| --- |
| # Glaucoma  img = cv.imread('images/test.png').astype(np.float32)/255.0  # Gaussian kernel  gh ,hr = divmod(np.size(img,0),2)  gw, wr = divmod(np.size(img,1),2)  x, y = np.mgrid[-gh:gh+hr, -gw:gw+wr]  sigma = 14\*\*2  gaussian\_kernel= np.exp(-(x\*\*2+y\*\*2)/(sigma\*\*2))  #Normalization  gaussian\_kernel = gaussian\_kernel / gaussian\_kernel.max()  blur = np.zeros(img.shape)  fig = plt.figure()  plt.axis('off')  plt.imshow(gaussian\_kernel, cmap=plt.get\_cmap('jet'), interpolation='nearest')  plt.colorbar()  blur[:,:,0] = (img[:,:,0] \* gaussian\_kernel)  blur[:,:,1] = (img[:,:,1] \* gaussian\_kernel)  blur[:,:,2] = (img[:,:,2] \* gaussian\_kernel)  blur = (blur\*255).astype(np.uint8)  fig = plt.figure()  plt.axis('off')  plt.imshow(cv.cvtColor(blur, cv.COLOR\_BGR2RGB))  if not os.path.exists('images'):      os.mkdir('images')  cv.imwrite('images/ex2.2.3.jpg', blur) |
| 首先使用OpenCV來讀取影像，於函式中輸入影像位置，以及color type：0為gray、1為RGB。必須注意，使用OpenCV進行影像讀取，影像的顏色型態為BGR，不同於RBG。  　　讀取後將資料型態由int轉為double型態，並進行正規化壓縮至[0,1]，降低計算誤差。  　　建立一個與圖片大小相同的Gaussian kernel，並且sigma必須很大，使遮罩的中間範圍都還是維持在1，而外圍才逐漸變小。  　　最後再將遮罩與圖片進行矩陣內元素相乘，就可以模擬青光眼。 |
| **第二題 以多維空間分析樹葉的差異** |
| 1. load 90 images 2. foreground mask 3. analysis  * gradient of signature * mean brightness by gray * red ratio R/(R+G+B) * mean high frequency (Laplacian + abs + mean)  1. 1000\*1000 white background 2. plot images to relative position by analysis |
| **Result** |
|  |

|  |
| --- |
| **Code** |
| **leaveFeature.m** |
| %image feature extraction: signature  function [signature\_gradient, redness, lightness, laplace\_texture] = leaveFeature(path)  %輸入葉片影像  img1 = double(imread(path));  **%%中間省略%%**  **%%中間省略%%**  **%%中間省略%%**  for x=1:h  for y=1:w  if mask(x,y) ==1  theta = mod(atan2(cx-x,y-cy)\*180/pi,360); %換算theta角  i = ceil(theta/interval+.01); %0.01是為了防止i=0  r = sqrt((cx-x)^2+(cy-y)^2); %遠離質心c的距離  if r > r\_histogram(i)  r\_histogram(i) = r; %更新r\_histogram中的最大r值  end  end  end  end |

|  |
| --- |
| %% calculate gradient of signature  signature\_gradient = mean(abs(gradient(r\_histogram)));  %% Redness  img4 = double(cat(3,mask,mask,mask));  img4 = img1.\*img4;  sum\_r = sum((img4(:,:,1)),"all");  sum\_rgb = sum(img4,"all");  redness = sum\_r/sum\_rgb;  %% Lightness  img5 = double(rgb2gray(uint8(img1)));  img5 = img5.\*mask;  lightness = sum(img5,"all")/sum(mask,"all");  %% Laplace Texture  laplace\_filter = fspecial("laplacian");  mean\_filter = fspecial("average",size(laplace\_filter));  img6 = abs(imfilter(img5, laplace\_filter));  img6 = imfilter(img6,mean\_filter);  laplace\_texture = sum(img6,"all")/sum(img6>0,"all");  end |
| 將參考程式進行修改，將加入其他特徵分析：   1. Gradient of Signature   　　透過呼叫gradient計算各點的梯度，並將計算結果取絕對值，最後計算整張圖片的平均值。   1. Redness   　　首先將圖片轉為double型態，再進行sum的運算，才不會使最大被限制在255(uint8)，獨立將red channel取出來後，將其與RGB影像相除，就可以計算出紅色的占比。   1. Lightness   　　要計算平均亮度必須先把RGB影像轉成Gray，這樣就能夠直接取得影像的亮度。   1. Laplace Texture   　　先將影像轉為Gray，與Laplacian Filter與Mean Filter進行濾波，並取絕對值，就可以獲得圖像的高頻特徵(紋理)。 |
| 修改完成後，將整個程式包裝成function，因為總共有90張圖片，必須要能夠重複執行。(紅框部分為原參考程式的部分，不對其進行修改。) |

|  |
| --- |
| **featureDistribution.m** |
| function white = featureDistribution(rows, colums)  white = ones(1000,1000,3)\*255;  for i=1:90  path=strcat("leaves/",int2str(i),".jpg");    img=double(imread(path));  [h w ch] = size(img);  mask = true(h,w); %建立二值遮罩(mask)  mask(find(img(:,:,3)>=240)) = 0; %假設影像藍色channel為240者, 必定為白色。  mask(find(mask>0)) = 1; %其餘設為0  mask = double(cat(3,mask,mask,mask));  row=uint32(max(rows)-rows(i)+1);  colum=colums(i);  white(row:row+h-1,colum:colum+w-1,:) = white(row:row+h-1,colum:colum+w-1,:).\*(not(mask)) + img.\*mask;  end  white=uint8(white);  end |
| 這邊建立一個featureDistribution的function，用來繪製樹葉特徵分佈圖，首先進行圖片的讀取，取得葉子部分的遮罩，並建立一個空白的矩陣。  　　輸入計算後的特徵值，作為圖像的座標，作法為將其設為圖像的左上角，這樣只需要以此點出發，取出跟圖像相同大小的矩陣，可以進行繪製。 |

|  |
| --- |
| **Main.m** |
| %HW2.3  clear;clc;  %feature = [signature\_gradient, redness, lightness, laplace\_texture]  signature\_gradient= zeros(90,1);  redness= zeros(90,1);  lightness= zeros(90,1);  laplace\_texture= zeros(90,1);  for i=1:90  path=strcat("leaves/",int2str(i),".jpg");  [signature\_gradient(i), redness(i), lightness(i), laplace\_texture(i)]=leaveFeature(path);  end  feature = [signature\_gradient, redness, lightness, laplace\_texture];  % normalization feature = uint32[1:800]  for i=1:4  feature(:,i) = uint32((feature(:,i)-min(feature(:,i)))/(max(feature(:,i))-min(feature(:,i)))\*800+1);  end  figure()  white = featureDistribution(feature(:,3),feature(:,2));  subplot(121)  imshow(white)  xlabel("Redness")  ylabel("Lightness")  white = featureDistribution(feature(:,4),feature(:,1));  subplot(122)  imshow(white)  xlabel("Absolute gradient of the signature cure")  ylabel("Laplacian Texture") |
| 此為主程式的部分，透過for loop的方式，輸入圖像的路徑，並呼叫leaveFeature()，計算90張葉子圖片的特徵，並將其結果存放在feature的list中。  　　為了將特徵值作為分布圖的x-y軸，必須先將其正規化至[1~800]，再呼叫featureDistribution()，繪製其特徵分布圖。 |