**ปิรามิดรูปภาพ**

เป้าหมาย

ในบทนี้,

* เราจะเรียนรู้เกี่ยวกับภาพพีระมิด
* เราจะใช้รูปภาพ pyramids เพื่อสร้างผลไม้ใหม่ "Orapple"
* เราจะเห็นฟังก์ชันเหล่านี้: **cv2.pyrUp ()** , **cv2.pyrDown ()**

ทฤษฎี

ปกติเราใช้ในการทำงานกับภาพที่มีขนาดคงที่ แต่ในบางโอกาสเราจำเป็นต้องใช้ภาพที่มีความละเอียดแตกต่างกันของภาพเดียวกัน ตัวอย่างเช่นในขณะที่ค้นหาบางสิ่งบางอย่างในภาพเช่นใบหน้าเราไม่แน่ใจว่าขนาดของวัตถุจะอยู่ในภาพอย่างไร ในกรณีนี้เราจะต้องสร้างชุดของภาพที่มีความละเอียดแตกต่างกันและค้นหาวัตถุในภาพทั้งหมด ชุดภาพเหล่านี้มีความละเอียดแตกต่างกันเรียกว่าภาพพีระมิด (เพราะเมื่อเก็บภาพไว้ในกองที่มีภาพที่ใหญ่ที่สุดที่ด้านล่างและภาพที่เล็กที่สุดที่ด้านบนเช่นพีระมิด)

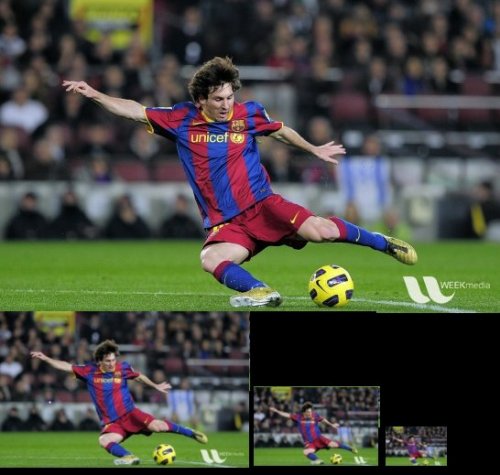
มีปิรามิดภาพจำนวน 2 ชนิด 1) Pyramid Gaussian และ 2) Laplacian Pyramids

ระดับที่สูงขึ้น (ความละเอียดต่ำ) ใน Pyramid Gaussian ถูกสร้างขึ้นโดยการลบแถวและคอลัมน์ที่ต่อเนื่องกันในภาพระดับล่าง (ความละเอียดสูงกว่า) จากนั้นแต่ละพิกเซลในระดับที่สูงขึ้นจะถูกสร้างขึ้นโดยการมีส่วนร่วมจาก 5 พิกเซลในระดับต้นแบบที่มีน้ำหนักเกร็ง เมื่อทำเช่นนี้M \ times Nภาพจะกลายเป็นM / 2 \ times N / 2ภาพ ดังนั้นพื้นที่จะลดลงเหลือหนึ่งในสี่ของพื้นที่เดิม เรียกว่า Octave รูปแบบเดียวกันนี้ยังคงดำเนินต่อไปเมื่อไปที่พีระมิดตอนต้น (เช่นความละเอียดจะลดลง) ในทำนองเดียวกันในขณะที่การขยายพื้นที่จะกลายเป็น 4 ครั้งในแต่ละระดับ เราสามารถพบปิรามิด Gaussian ใช้**cv2.pyrDown ()**และ**cv2.pyrUp ()**ฟังก์ชั่น

img = cv2.imread('messi5.jpg')

lower\_reso = cv2.pyrDown(higher\_reso)

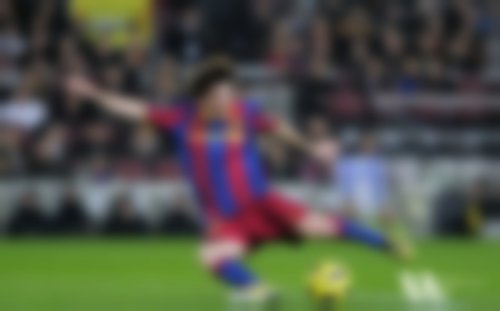
Below is the 4 levels in an image pyramid.



Now you can go down the image pyramid with **cv2.pyrUp()** function.

higher\_reso2 = cv2.pyrUp(lower\_reso)

โปรดจำไว้ว่า*higher\_reso2*ไม่เท่ากับ*higher\_reso*เพราะเมื่อคุณลดความละเอียดลงแล้วคุณจะหลวมข้อมูล ด้านล่างภาพเป็น 3 ระดับลงปิรามิดที่สร้างขึ้นจากภาพที่เล็กที่สุดในกรณีก่อนหน้านี้ เปรียบเทียบกับภาพต้นฉบับ:

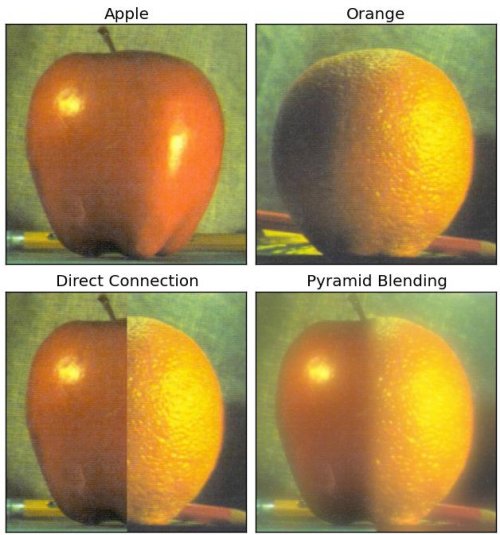


พีระมิด Laplacian ถูกสร้างขึ้นจากปิรามิด Gaussian ไม่มีฟังก์ชันพิเศษสำหรับที่ ภาพพีระมิด Laplacian เหมือนภาพขอบเท่านั้น องค์ประกอบส่วนใหญ่เป็นศูนย์ ใช้ในการบีบอัดภาพ ระดับในพีระมิด Laplacian สร้างขึ้นจากความแตกต่างระหว่างระดับนั้นใน Pyramid Gaussian และการขยายระดับของ Upper Pyramid ใน Gaussian Pyramid ระดับของระดับ Laplacian สามระดับจะมีลักษณะดังนี้ (ความคมชัดถูกปรับเพื่อเพิ่มเนื้อหา):



การผสมภาพโดยใช้ปิรามิด

โปรแกรมประยุกต์หนึ่งของปิรามิดคือการผสมผสานรูปภาพ ตัวอย่างเช่นในการเย็บภาพคุณจะต้องวางซ้อนภาพสองภาพไว้ด้วยกัน แต่อาจดูไม่ดีเนื่องจากไม่ต่อเนื่องระหว่างภาพ ในกรณีนี้การผสมภาพด้วยปิรามิดช่วยให้คุณสามารถผสมผสานอย่างลงตัวโดยไม่ต้องทิ้งข้อมูลมากในภาพ หนึ่งในตัวอย่างคลาสสิกของการผสมผสานของผลไม้สองสีส้มและแอปเปิ้ล ดูผลตอนนี้ตัวเองที่จะเข้าใจสิ่งที่ฉันพูด:



โปรดตรวจสอบการอ้างอิงครั้งแรกในรีซอร์สเพิ่มเติม แต่ก็มีรายละเอียดเกี่ยวกับการผสมผสานรูปภาพปิรามิด Laplacian Pyramids แบบเต็มรูปแบบเพียงทำตามขั้นตอนดังนี้:

1. โหลดภาพแอปเปิลและส้มสองภาพ
2. ค้นหา Piramids Gaussian สำหรับแอปเปิลและส้ม (ในตัวอย่างเฉพาะนี้จำนวนระดับคือ 6)
3. จากปิรามิด Gaussian หาปิรามิด Laplacian ของพวกเขา
4. ตอนนี้ใช้ครึ่งซ้ายของแอปเปิ้ลและครึ่งด้านขวาของสีส้มในแต่ละระดับของ Laplacian Pyramids
5. สุดท้ายจากภาพร่วมกันปิรามิดสร้างภาพต้นฉบับ

ด้านล่างนี้เป็นรหัสเต็ม (เพื่อความเรียบง่ายแต่ละขั้นตอนจะทำแยกกันซึ่งอาจใช้หน่วยความจำมากขึ้นคุณสามารถเพิ่มประสิทธิภาพได้หากต้องการ)

**import** **cv2**

**import** **numpy** **as** **np**,**sys**

A = cv2.imread('apple.jpg')

B = cv2.imread('orange.jpg')

*# generate Gaussian pyramid for A*

G = A.copy()

gpA = [G]

**for** i **in** xrange(6):

G = cv2.pyrDown(G)

gpA.append(G)

*# generate Gaussian pyramid for B*

G = B.copy()

gpB = [G]

**for** i **in** xrange(6):

G = cv2.pyrDown(G)

gpB.append(G)

*# generate Laplacian Pyramid for A*

lpA = [gpA[5]]

**for** i **in** xrange(5,0,-1):

GE = cv2.pyrUp(gpA[i])

L = cv2.subtract(gpA[i-1],GE)

lpA.append(L)

*# generate Laplacian Pyramid for B*

lpB = [gpB[5]]

**for** i **in** xrange(5,0,-1):

GE = cv2.pyrUp(gpB[i])

L = cv2.subtract(gpB[i-1],GE)

lpB.append(L)

*# Now add left and right halves of images in each level*

LS = []

**for** la,lb **in** zip(lpA,lpB):

rows,cols,dpt = la.shape

ls = np.hstack((la[:,0:cols/2], lb[:,cols/2:]))

LS.append(ls)

*# now reconstruct*

ls\_ = LS[0]

**for** i **in** xrange(1,6):

ls\_ = cv2.pyrUp(ls\_)

ls\_ = cv2.add(ls\_, LS[i])

*# image with direct connecting each half*

real = np.hstack((A[:,:cols/2],B[:,cols/2:]))

cv2.imwrite('Pyramid\_blending2.jpg',ls\_)

cv2.imwrite('Direct\_blending.jpg',real)