Vinit Ramesh Gore

M20CS064

dB2 Coefficients

Decomposition

lowpass = [-0.12940952255092145, 0.22414386804185735, 0.836516303737469, 0.48296291314469025]

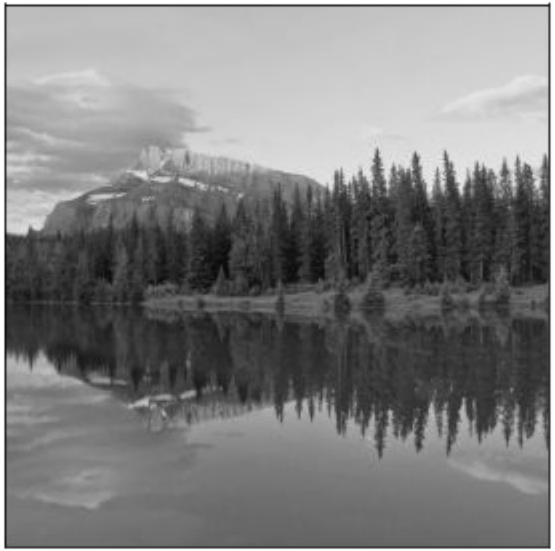
highpass = [-0.48296291314469025, 0.836516303737469, -0.22414386804185735, -0.12940952255092145]

Reconstruction

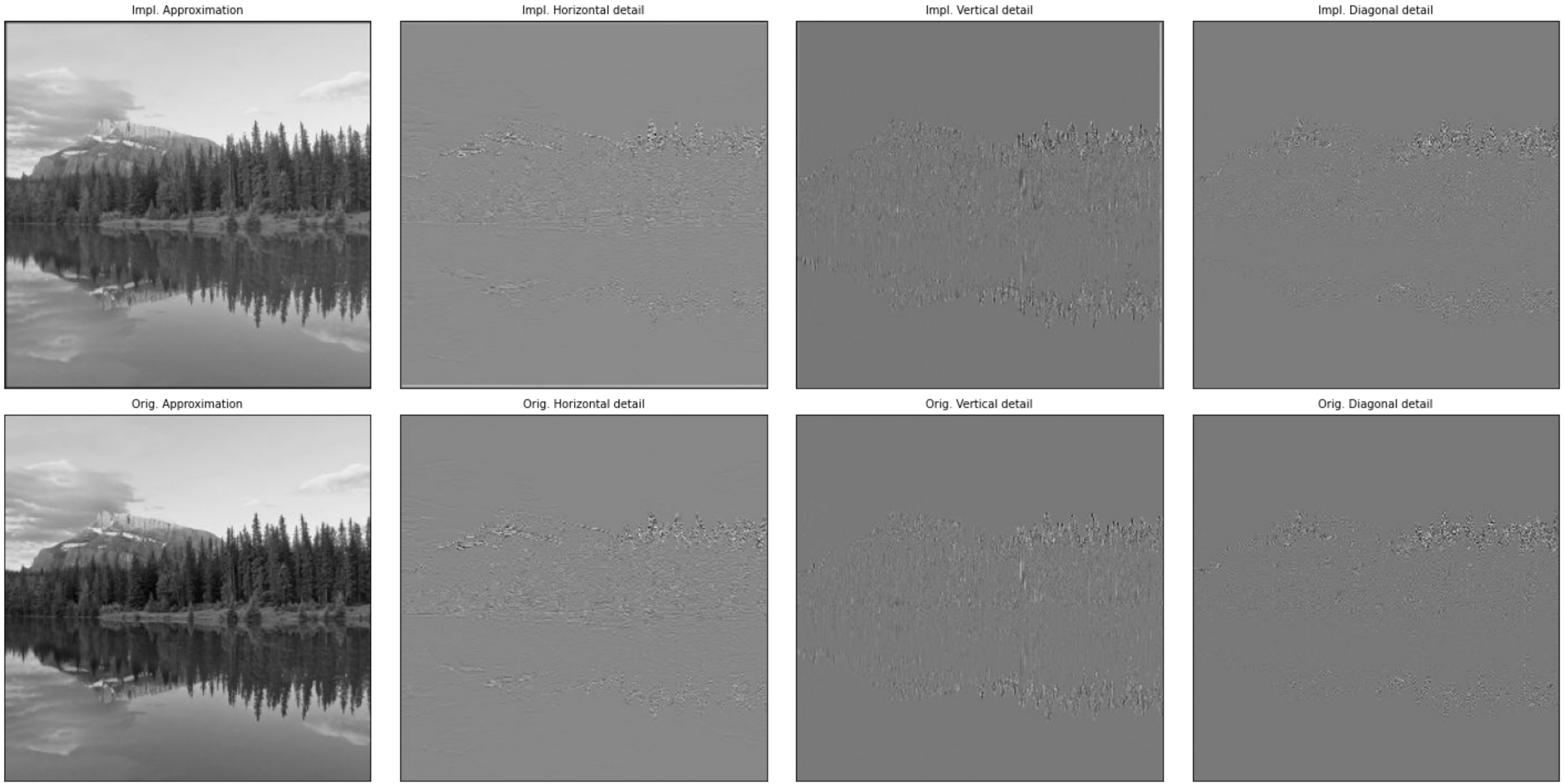
lowpass = [0.48296291314469025, 0.836516303737469, 0.22414386804185735, -0.12940952255092145]

highpass = [-0.12940952255092145, -0.22414386804185735, 0.836516303737469, -0.48296291314469025]

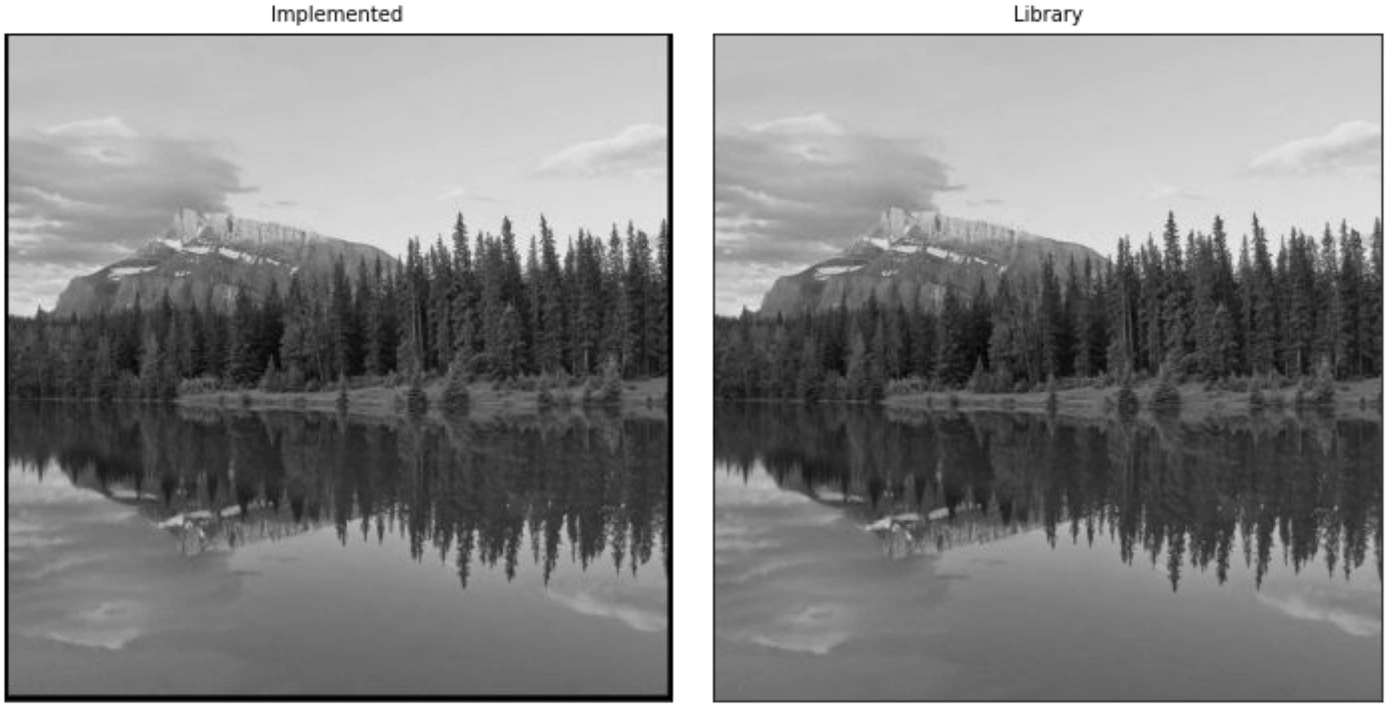
Input Image



2D DWT Subbands



Reconstructed Image



Implemented

Library

Code for Decomposition

```
decomp_coeff_l = [-0.12940952255092145, 0.22414386804185735, 0.836516303737469,
0.48296291314469025]
decomp_coeff_h = [-0.48296291314469025, 0.836516303737469, -0.22414386804185735,
-0.12940952255092145]

#stage 1
np_single_img_l = image_convolve_and_downsample(np_single_img, decomp_coeff_l,'H')
np_single_img_h = image_convolve_and_downsample(np_single_img, decomp_coeff_h,'H')

#stage 2
ll = image_convolve_and_downsample(np_single_img_l, decomp_coeff_l,'V')
lh = image_convolve_and_downsample(np_single_img_l, decomp_coeff_h,'V')
hl = image_convolve_and_downsample(np_single_img_h, decomp_coeff_l,'V')
hh = image_convolve_and_downsample(np_single_img_h, decomp_coeff_h,'V')
```

Code for Decomposition

```python
def image_convolve_and_downsample (image: np.ndarray, coeff: np.ndarray, axis: str):
  if len(image.shape) !=  2:
    raise ShapeError( 'Numpy image should be 2D' )

  #converting to numpy array
  coeff = np.asarray(coeff)

  imgH = image.shape[ 0]
  imgW = image.shape[ 1]

  #convolve each row with coeff
  if axis == 'H':
    #convolution
    convolved_img = np.array([])
    for row in image:
      convolved_row = np.convolve(row, coeff)
      convolved_img = np.append(convolved_img, convolved_row)
    convolved_img = convolved_img.reshape(imgH,  -1)
    #downsample
    result_img = np.array([])
    for row in convolved_img:
      downsampled_row = np.array([])
      for i, px in enumerate(row):
        if i%2 == 0:
          downsampled_row = np.append(downsampled_row, px)
      result_img = np.append(result_img, downsampled_row)
    result_img = result_img.reshape(imgH,  -1)
    return result_img
```

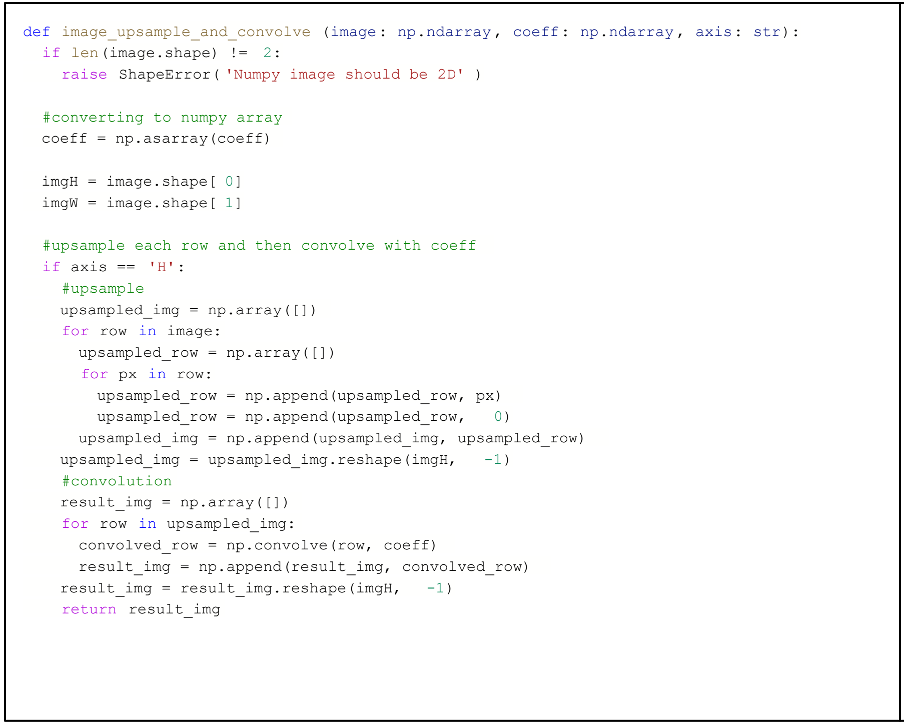
```python
  #convolve each col with coeff
  if axis == 'V':
    imageT = np.transpose(image)
    #convolution
    convolved_imgT = np.array([])
    for row in imageT:
      convolved_row = np.convolve(row, coeff)
      convolved_imgT = np.append(convolved_imgT, convolved_row)
    convolved_imgT = convolved_imgT.reshape(imgW,  -1)
    #downsample
    result_imgT = np.array([])
    for row in convolved_imgT:
      downsampled_row = np.array([])
      for i, px in enumerate(row):
        if i%2 == 0:
          downsampled_row = np.append(downsampled_row, px)
      result_imgT = np.append(result_imgT, downsampled_row)
      result_imgT = result_imgT.reshape(imgW,  -1)
    result_img = np.transpose(result_imgT)
    return result_img
```

Code for Reconstruction

```
reconstr_coeff_l = [0.48296291314469025, 0.836516303737469, 0.22414386804185735, -0.12940952255092145]
reconstr_coeff_h = [-0.12940952255092145, -0.22414386804185735, 0.836516303737469, -0.48296291314469025]
rec_ll = image_upsample_and_convolve(ll, reconstr_coeff_l, 'V')
rec_lh = image_upsample_and_convolve(lh, reconstr_coeff_h, 'V')
syn_ll_lh = (rec_ll + rec_lh)
rec_hl = image_upsample_and_convolve(hl, reconstr_coeff_l, 'V')
rec_hh = image_upsample_and_convolve(hh, reconstr_coeff_h, 'V')
syn_hl_hh = (rec_hl + rec_hh)

rec_ll_lh = image_upsample_and_convolve(syn_ll_lh, reconstr_coeff_l, 'H')
rec_hl_hh = image_upsample_and_convolve(syn_hl_hh, reconstr_coeff_h, 'H')
syn = (rec_ll_lh + rec_hl_hh)
```

Code for Reconstruction

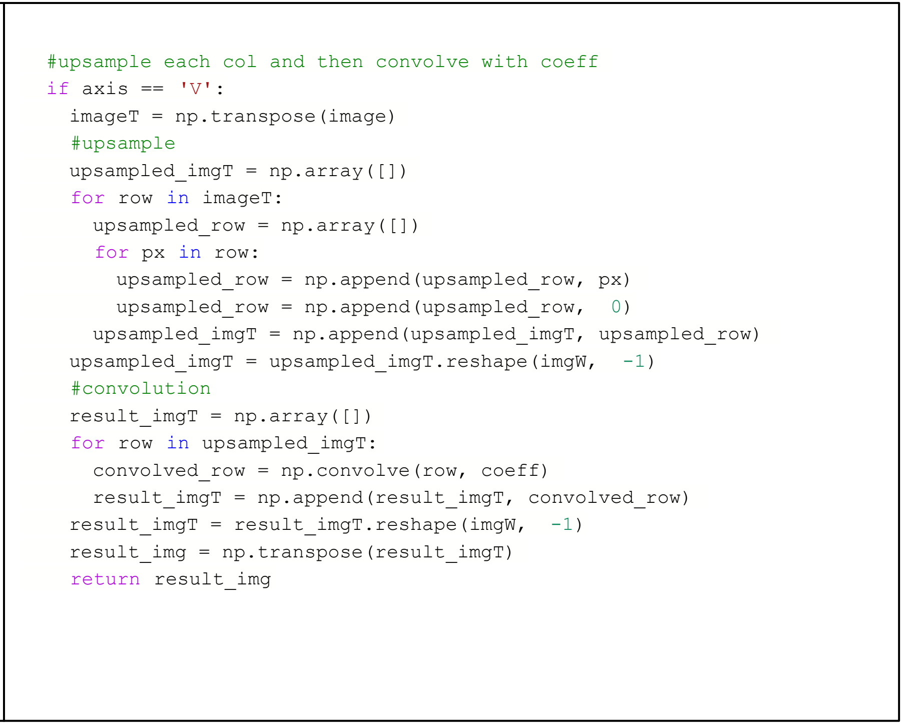


```
def image_upsample_and_convolve (image: np.ndarray, coeff: np.ndarray, axis: str):
  if len(image.shape) != 2:
    raise ShapeError( 'Numpy image should be 2D' )

  #converting to numpy array
  coeff = np.asarray(coeff)

  imgH = image.shape[ 0]
  imgW = image.shape[ 1]

  #upsample each row and then convolve with coeff
  if axis == 'H':
    #upsample
    upsampled_img = np.array([])
    for row in image:
      upsampled_row = np.array([])
      for px in row:
        upsampled_row = np.append(upsampled_row, px)
        upsampled_row = np.append(upsampled_row,   0)
      upsampled_img = np.append(upsampled_img, upsampled_row)
    upsampled_img = upsampled_img.reshape(imgH,   -1)
    #convolution
    result_img = np.array([])
    for row in upsampled_img:
      convolved_row = np.convolve(row, coeff)
      result_img = np.append(result_img, convolved_row)
    result_img = result_img.reshape(imgH,   -1)
    return result_img
```



```
  #upsample each col and then convolve with coeff
  if axis == 'V':
    imageT = np.transpose(image)
    #upsample
    upsampled_imgT = np.array([])
    for row in imageT:
      upsampled_row = np.array([])
      for px in row:
        upsampled_row = np.append(upsampled_row, px)
        upsampled_row = np.append(upsampled_row,  0)
      upsampled_imgT = np.append(upsampled_imgT, upsampled_row)
    upsampled_imgT = upsampled_imgT.reshape(imgW,  -1)
    #convolution
    result_imgT = np.array([])
    for row in upsampled_imgT:
      convolved_row = np.convolve(row, coeff)
      result_imgT = np.append(result_imgT, convolved_row)
    result_imgT = result_imgT.reshape(imgW,  -1)
    result_img = np.transpose(result_imgT)
    return result_img
```