M20AIE318-BV-F1-A1

August 10, 2022

Implemented by: *Utkarsh Thusoo (M20AIE318)*

---

1 Problem statement

Write a code for convolution and test it on a simple artificial image.

Suggested steps:

1. Create a chessboard image of dimension 1024 x 1024, alternate squares of black (8) and white (8) along the rows and columns.
2. Define Horizontal and Vertical Sobel filters as 2D arrays.
3. Write code for convolution – Do not use library routine
4. Test your convolution code with the Sobel filters on the chessboard image. The outputs should be horizontal and vertical lines.

2 Implementation Notes

- Google Colab link
- Code is documented for readability purposes
- Horizontal Sobel Filter `[[-1, 0 , 1], [-2 , 0 , 2], [-1, 0 , 1]]`
- Vertical Sobel Filter `[[-1, -2 , -1], [0 , 0 , 0], [1, 2 , 1]]`

---

[69]:
```
# Import NumPy for array/matrix operations, and matplotlib.pyplot for plotting/
↪drwing image
import numpy as np
import matplotlib.pyplot as plt

print( "Environment set")
```

```
Environment set
```

[70]:
```
# Create a chess-board image

height=1024 # Image dimensions
```

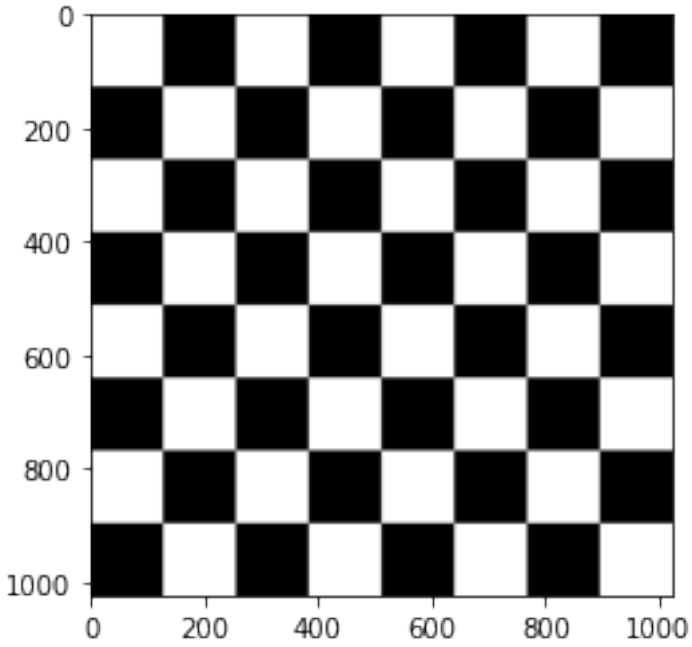
```
width =1024

chessboard = np.zeros((height, width), dtype=np.uint8) # Initialize with zeroes␣
↪(black)


for i in range(0,height):
  for j in range(0, width):
      if(((i%256 < 128 ) and (j%256 < 128)) or ((i%256 >= 128 ) and (j%256 >=␣
↪128))) :
        chessboard[i,j] = 255    # Make it white

# Show the image
fig = plt.figure() # use default display size
plt.imshow(chessboard, cmap="gray")
```

[70]: <matplotlib.image.AxesImage at 0x7fa84cd15850>



[71]:
```
# Define horizontal and vertical Sobel filters

# Creating horizontal Sobel filter
filter_h = np.array([[-1, 0 , 1], [-2 , 0 , 2], [-1, 0 , 1]])

# Creating vertical Sobel filter
filter_v = np.array([[-1, -2 , -1], [0 , 0 , 0], [1, 2 , 1]])
```

```
print("filter_h = ", filter_h)
print("filter_v = ", filter_v)
```

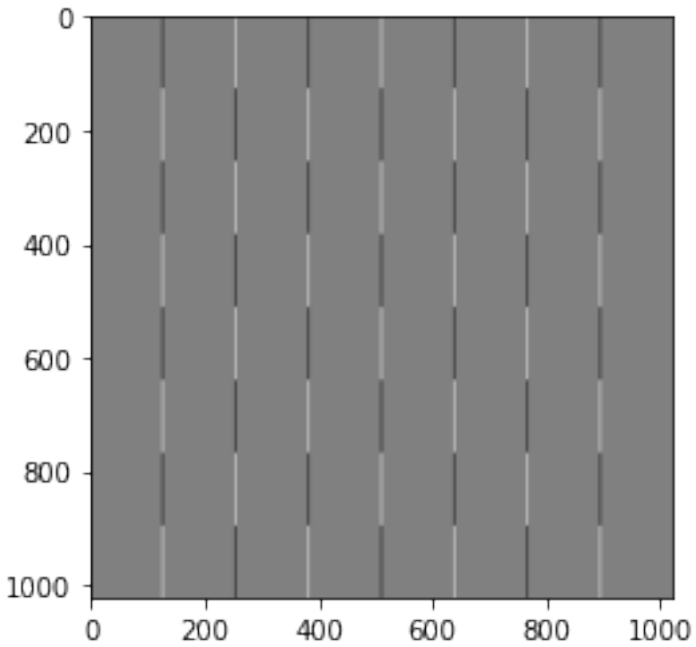
```
filter_h =  [[-1  0  1]
 [-2  0  2]
 [-1  0  1]]
filter_v =  [[-1 -2 -1]
 [ 0  0  0]
 [ 1  2  1]]
```

[72]:
```
# Convolve with filter_h

conv_image_h = my2Dconvolution(filter_h, chessboard)

# Show convoluted image
fig = plt.figure() # use default display size
plt.imshow(conv_image_h, cmap="gray")
```

[72]:
```
<matplotlib.image.AxesImage at 0x7fa84cd02390>
```

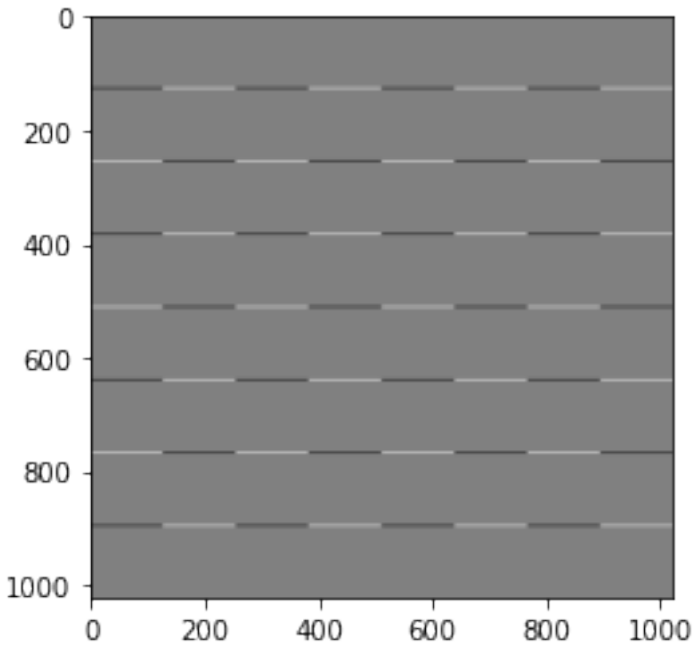


[73]:
```
# Convolve with filter_v

conv_image_v = my2Dconvolution(filter_v, chessboard)
```

```
# Show convoluted image
fig = plt.figure() # use default display size
plt.imshow(conv_image_v, cmap="gray")
```

[73]: <matplotlib.image.AxesImage at 0x7fa84cc65a90>



[74]:
```
# define 2D convolution function
def my2Dconvolution(filter, image):
  #Input image matrix rows
  W_in_row = int(image.shape[0])

  #Input image matrix columns
  W_in_col = int(image.shape[1])

  #Rows in Filter matrix
  F_x = int(filter.shape[0])

  #Columns in Filter matrix
  F_y = int(filter.shape[1])

  #Strides we will use
  S = 1

  #Number of rows in the result matrix
  W_out_row = int(((W_in_row - F_x) / S) + 1)
```

```
  #Number of columns in the result matrix
  W_out_col = int(((W_in_col - F_y) / S) + 1)

  #Default initialization of matrix with 0's
  result = np.zeros((W_out_row,W_out_col), dtype=int)

  #Iterating over the rows for based on the resultant size of rows
  for i in range(0, W_out_row):
    #Iterating over the columns for based on the resultant size of columns
    for j in range(0, W_out_col):

      # Value that needs to be stored at the location [W_x][W_y] based on
      # the calculations
      temp = 0

      # Iterarative variable for filter array.
      # 'fx_counter' Corresponds to row count in filter array
      fx_counter = 0
      for W_h in range(i, i + F_x):
        # Iterarative variable for filter array.
        # 'fy_counter' Corresponds to column count in filter array
        fy_counter = 0
        for W_v in range(j, j + F_y):
          # Computing the value for [W_x][W_y] based on the multiplication
          # result of imgae[x][y] * filter[x][y]
          temp = temp + image[W_h][W_v] * filter[fx_counter][fy_counter]

          fy_counter = fy_counter + 1
        fx_counter = fx_counter + 1

      # Storing the value of temp in result array
      result[i][j] = temp

  return result
```

[81]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('M20AIE318-BV-F1-A1.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/M20AIE318-BV-F1-A1.ipynb to pdf
[NbConvertApp] Support files will be in M20AIE318-BV-F1-A1_files/
[NbConvertApp] Making directory ./M20AIE318-BV-F1-A1_files
[NbConvertApp] Making directory ./M20AIE318-BV-F1-A1_files
[NbConvertApp] Making directory ./M20AIE318-BV-F1-A1_files
[NbConvertApp] Writing 34124 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 63446 bytes to /content/drive/My
Drive/M20AIE318-BV-F1-A1.pdf
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[81]: 'File ready to be Downloaded and Saved to Drive'