Homework2_ankitparekh

September 17, 2022

1 ECE 4554/ ECE 5554 / Computer Vision

This file contains the coding problems (Machine Problems 1, 2, and 3) for Homework 2. Your job is to implement/modify the sections within this notebook that are marked with ”TO DO”.

1.1 TO DO: Enter your Virginia Tech Username (PID) here: ankitparekh

1.2 Honor Code reminder

Once again, please review the Honor Code statement in the syllabus. This is not a ”team project”.

1.3 Submission guidelines for the coding problems (Google Colab)

1. Please verify that you have entered your Virginia Tech Username in all of the appropriate places.
2. After your solutions are complete, click Runtime->”Restart and run all”; then verify that all of your solutions are visible in this notebook.
3. Click File->Save near the top of the page to save the latest version of your notebook at Google Drive.
4. Verify that the last 2 cells have executed, creating a PDF version of this notebook at Google Drive. (Note: if you face difficulty with this step, please refer to https://pypi.org/project/notebook-as-pdf/)
5. Look at the PDF file and check that all of your solutions are displayed correctly there.
6. Download your notebook file and the PDF version to your laptop.
7. On your laptop, create a ZIP version of this notebook file. (Please don’t include any separate data files.) Use file name Homework2_Code_USERNAME.zip, with your own Username.
8. For your PDF version, use file name Homework2_Notebook_USERNAME.pdf, with your own Username.
9. **Submit these 2 files and your PDF file for Problems 1-4 SEPARATELY to Canvas.** Do not zip them all together.

2 Set up the environment

[1]:
```
# Mount your Google Drive to this notebook
# The purpose is to allow your code to access to your files
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
# Change the directory to your own working directory
# Your code will be able to read and write files in your working directory
# TO DO: enter the name of your directory
import os
os.chdir('/content/drive/My Drive/5554/HW2')
```

[3]:
```
# Import library modules
import sys
import cv2 # OpenCV library
from PIL import Image # Python Imaging Library
import numpy as np
import matplotlib.pyplot as plt

# (Note: we would use cv2.imshow if running on your laptop;
#  cv2.imshow is not allowed in Colab, so use cv2_imshow instead)
from google.colab.patches import cv2_imshow
```

---

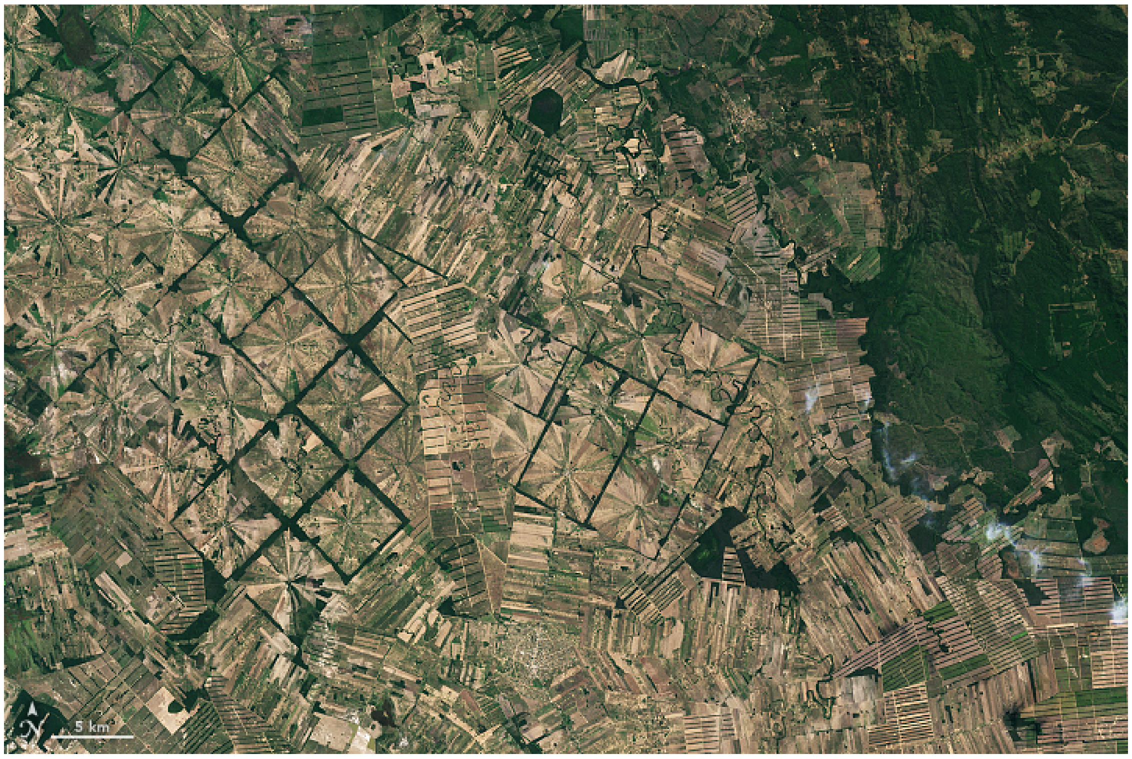
3 Machine Problem 1: Image Filtering (10 points)

Write Python/OpenCV code that will perform linear filtering of an image. Demonstrate correct operation of your code by applying filters that are commonly used for smoothing.

For this problem, do not use any OpenCV functions other than basic operations for loading/saving/displaying image files. During the filtering operation, your code must access pixel values directly, probably with nested 'for' loops. (We know that OpenCV has built-in functions that could be used here, such as cv2.filter2D and cv2.GaussianBlur, but you are not allowed to use them. The purpose of this problem is for you to gain a good understanding of operations at the pixel level.)

[4]:
```
# GETTING STARTED
# Verify that you can input an image from your working directory
# and convert it to grayscale format.
# The resulting img_grayscale will be the input to the filtering operations␣
 ↪below.
```

```
# Image source:
# https://visibleearth.nasa.gov/images/150257/
 ↪patterns-of-forest-change-in-bolivia/150257f

filename = "boliviaradial_oli_2022234_front.jpg"
img_color = cv2.imread(filename, cv2.IMREAD_COLOR)
cv2_imshow(img_color)

print ('\n')
img_grayscale = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
cv2_imshow(img_grayscale)
```



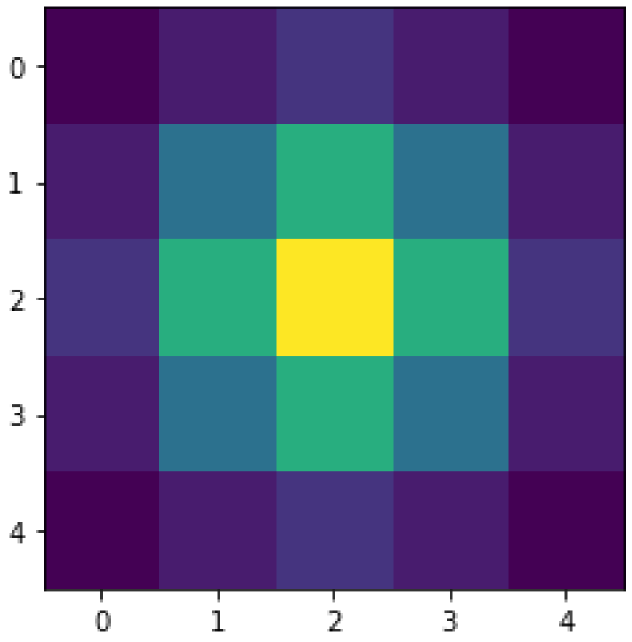
a) Write a Python function linear_filter() that accepts an image and a kernel as input parameters. Your function must create an output image by applying the kernel (also called "filter", "operator", "template") to the input image. For convenience, use cross-correlation in your computations. (Apply the kernel directly, without any reversal of index.) This new image is returned by your function.

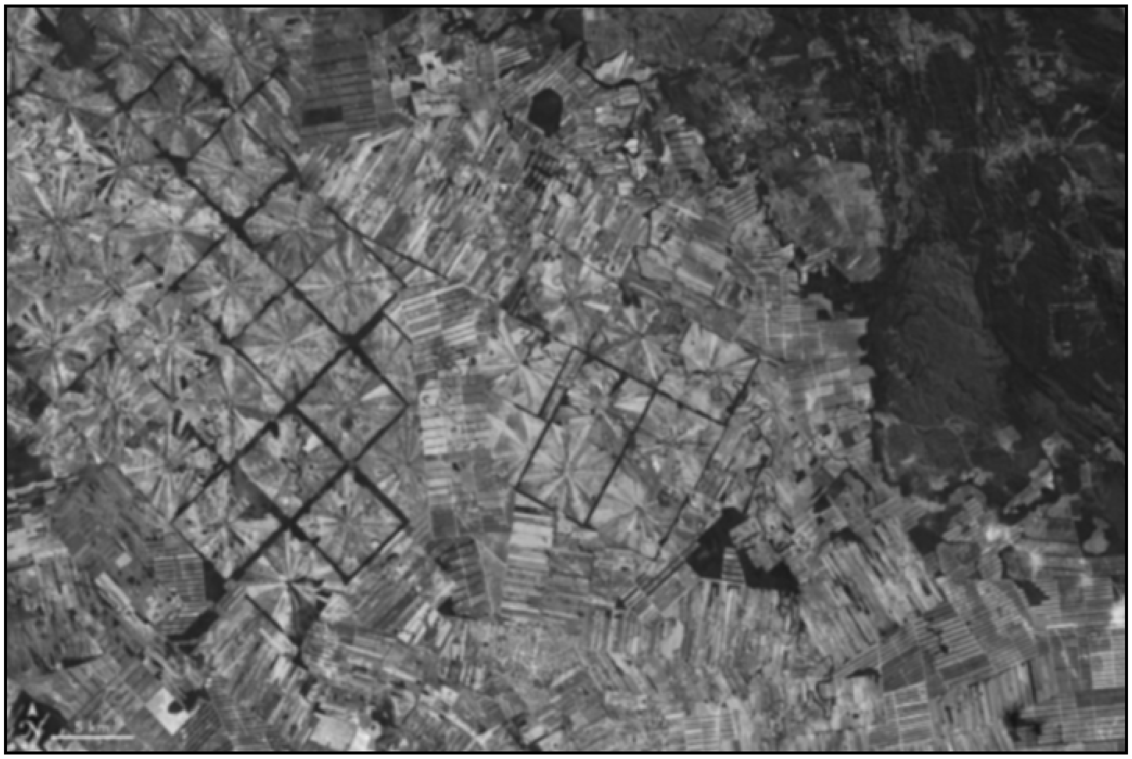
[5]:
```
###################################
# TO DO: Implement the function

def linear_filter(img_in, kernel):
  '''Filter an input image by applying cross-correlation with a kernel.

  Input:
    img_in: a grayscale image of any size larger than the kernel,
     in both row and column directions.
    kernel: a 2D array of floating-point values;
     you may assume that this array is square,
     with an odd number of rows and an odd number of columns;
     use the *center* of this kernel as its point of reference for filtering.
     Perform cross-correlation (do not reverse the kernel).

  Return value:
     an image with the same row/column size as img_in,
     but each pixel is a floating-point value;
    apply the kernel only at locations where it fits entirely within the
    input image;
    the remaining pixels (near the outside border of the output image)
    must be set to zero;
    for any negative values, take the absolute value;
    clip the final output so that every pixel value lies in the range 0 to 255.

  '''

  kernel_height, kernel_width = np.shape(kernel)

  img_height, img_width = np.shape(img_in)

  img_out = np.zeros([img_height, img_width])

  for row in range(kernel_height//2, img_height-(kernel_height//2)):
    for column in range(kernel_width//2, img_width-(kernel_width//2)):

      img_out[row][column] = abs(np.sum(img_in[row-(kernel_height//2):
↪row+(kernel_height//2)+1, column-(kernel_width//2):column+(kernel_width//
↪2)+1]*kernel))
      if img_out[row][column] > 255:
        img_out[row][column] = 255

  return img_out # Each pixel must be of type np.float32
```

Test your linear_filter() function with the following commands.

[6]:
```
# Here is an example smoothing filter,
#  approximating a 2D Gaussian function with sigma = 1.
gaussian5x5 = np.array([
          [1, 4, 7, 4, 1],
          [4, 16, 26, 16, 4],
          [7, 26, 41, 26, 7],
          [4, 16, 26, 16, 4],
          [1, 4, 7, 4, 1,]], dtype=np.float32) / 273.0

# Apply the smoothing filter
img_result = linear_filter(img_grayscale, gaussian5x5)

# display the kernel
plt.imshow(gaussian5x5)
plt.show()

# Plot both images to make it easy to see that they are the same size
cv2_imshow(img_grayscale)
print('\n')
```

```
cv2_imshow(img_result)
```



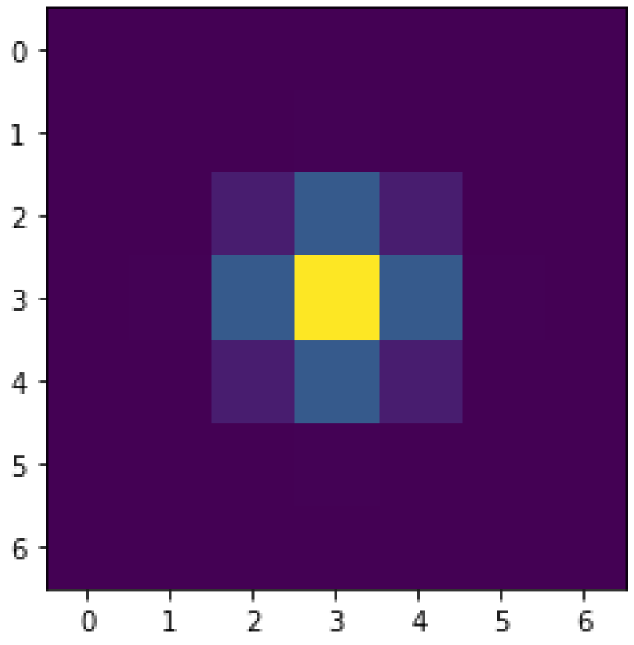


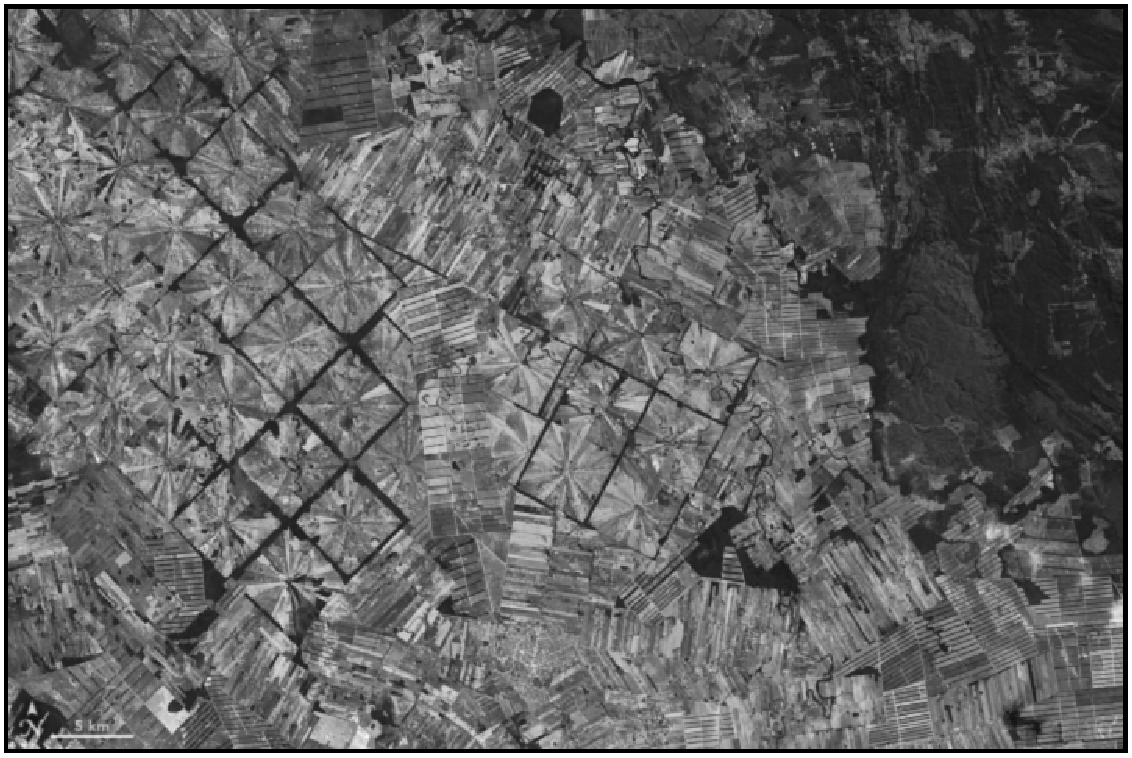
b) Write a function gaussian_kernel() that returns a new kernel of size 7x7, approximating a 2D Gaussian function with sigma = 0.63. Apply this kernel to img_grayscale, and plot the result. (You must calculate these kernel coefficients yourself. It is okay to calculate them at run-time; it is also okay to calculate them in advance and hard-code them into your function. You can use basic Python/NumPy math functions if you wish, but do not use any special OpenCV functions to create a kernel.)

[7]:
```
from ctypes import sizeof
###################################
# TO DO: write the code

def gaussian_kernel():
    """
    Return a Gaussian kernel, to be used for image filtering

    Input parameters:
     none

    Return value:
     a kernel of size 7x7,
     approximating a 2D Gaussian function with sigma = 0.63
    """
    size = 7
    sigma = 0.63
    sample_points = np.linspace(-(size - 1) / 2., (size - 1) / 2., size)
    gauss1d = np.exp(-0.5 * np.square(sample_points) / np.square(sigma))
    kernel = np.outer(gauss1d, gauss1d)
    return kernel / np.sum(kernel)
```

Test your functions gaussian_kernel() and linear_filter() with the following commands.

[8]:
```
gaussian7x7 = gaussian_kernel()
# display the kernel; the maximum value should be at the center
plt.imshow(gaussian7x7)
plt.show()

# display the filtered image
img_result = linear_filter(img_grayscale, gaussian7x7)
cv2_imshow(img_result)
```





---

4 Machine Problem 2: Compass Edge Detection (10 points)

Write a Python function that will perform ”compass” edge detection. You are encouraged (but not required) to write your code so that it uses your linear_filter() function from the previous problem.

For this problem, do not use any OpenCV functions other than basic operations for loading/saving/displaying image files. Your code must access pixel values directly.

[9]:
```
# GETTING STARTED
# Verify that you can input another image from your working directory
# and convert it to grayscale format.
# The resulting img_grayscale will be the input to the edge-detection code.

# Image source:
# https://digitalmedia.fws.gov/digital/collection/natdiglib/id/30095/rec/41

filename = "Achillea millefolium.jpg"
img_color2 = cv2.imread(filename, cv2.IMREAD_COLOR)
cv2_imshow(img_color2)

print ('\n')
```

```
img_grayscale2 = cv2.cvtColor(img_color2, cv2.COLOR_BGR2GRAY)
cv2_imshow(img_grayscale2)
```

Write a Python function compass_edge() that will input a grayscale image and perform ”compass” edge detection. The function must use the 4 generalized Sobel masks that are shown in packet 5 of the lecture slides.

Your function should compute responses for all 4 of the generalized Sobel masks; take the absolute value of each response; and then assign to each output pixel the maximum of all 4 absolute-value responses at that pixel location. Finally, make a change to this output image that will aid in displaying your result: perform a linear mapping from the range [minimum edge value, maximum edge value] to the range [0, 255].

[10]:
```
####################################
# TO DO: write the code

def compass_edge(img_in):
  """
  Detect edges in an image

  Input parameters:
    img_in: a grayscale image

  Return value:
    an "edge image" with the same row/column size as img_in
  """

  sobel_right  = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
  sobel_left   = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
  sobel_top    = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
  sobel_bottom = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

  img_in_sobel_right = linear_filter(img_in, sobel_right)
  img_in_sobel_left = linear_filter(img_in, sobel_left)
  img_in_sobel_top = linear_filter(img_in, sobel_top)
  img_in_sobel_bottom = linear_filter(img_in, sobel_bottom)

  img_out = np.zeros(img_in.shape)

  rows, columns = img_out.shape

  for row in range(rows):
    for column in range(columns):
      img_out[row][column] = max(img_in_sobel_right[row][column],␣
↪img_in_sobel_left[row][column], img_in_sobel_top[row][column],␣
↪img_in_sobel_bottom[row][column])

  minimum_edge_value, maximum_edge_value = np.min(img_out), np.max(img_out)
```

```
def linear_mapping(x):
  """
  Linear Mapping function f(x) for [a,b] to [c,d]:

  f(x) =  c + ((d-c)/(b-a))*(x-a)

  In our case:
     [a, b] = [minimum_edge_value, maximum_edge_value]
     [c, d] = [0, 255]
  """

  result = ((255)/
↪(maximum_edge_value-minimum_edge_value))*(x-minimum_edge_value)
  return result

for row in range(rows):
  for column in range(columns):
    img_out[row][column] = linear_mapping(img_out[row][column])

return img_out
```

Test your function compass_edge() with the following commands.

[11]:
```
img_result2 = compass_edge(img_grayscale2)
cv2_imshow(img_result2.astype(np.uint8))
```

5 Machine Problem 3: Gaussian Pyramid (10 points)

Write a function named generate_pyramid() that generates a Gaussian Pyramid. For a given grayscale image, your function should apply your 7x7 Gaussian filter and downscale (subsample) the result. In your code, use a loop to perform these steps successively to generate multiple layers of a pyramid, and display all of those images.

For full credit, you must use your gaussian_kernel() and linear_filter() functions from Machine Problem 1. The subsampling step should result in a new image that is half the width and half the height of the input. (For odd-numbered dimensions, simply ignore the fractional part when computing the new width and height.)

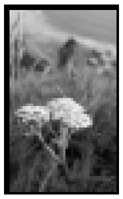
The final pyramid should consist of the original grayscale image along with at least 5 filtered and subsampled images. Display all of the images in your pyramid.

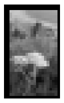
For this problem, do not use external or imported functions for filtering or downscaling. Do not use any OpenCV functions other than basic operations for loading/saving/displaying image files.

[12]:
```
###################################
# TO DO: write the code

def generate_pyramid(img_in):
  """
  Generate and display a Gaussian pyramid

  Input parameters:
    img_in: a grayscale image

  Return value:
    none
  """

  # place your code here
  gaussian7x7 = gaussian_kernel()


  def blur_and_subsample(img_in):
    filtered_image = linear_filter(img_in, gaussian7x7)
    [input_height, input_width] = img_in.shape
    f = 2 # Down Sampling rate
    img_out = np.zeros((input_height//f, input_width//2))

    for i in range(0, input_height, f):
      for j in range(0, input_width, f):
        try:
          img_out[i//f][j//f] = filtered_image[i][j]
        except IndexError:
          pass
    return img_out

  def generate_levels(img_in, iterations):

    previous = img_in
    for i in range(iterations):
      next = blur_and_subsample(previous)
      cv2_imshow(next)
      previous = next

  #Display
  cv2_imshow(img_in)
  generate_levels(img_in, 5)
```

Test your generate_pyramid() function with the following commands.

[13]:
```
filename = "Achillea millefolium.jpg"
img_color2 = cv2.imread(filename, cv2.IMREAD_COLOR)
img_grayscale2 = cv2.cvtColor(img_color2, cv2.COLOR_BGR2GRAY)

generate_pyramid(img_grayscale2)
```





---

6 Creating a PDF version of your current notebook

[ ]:
```
#The following two installation steps are needed to generate a PDF version of␣
↪the notebook
#(These lines are needed within Google Colab, but are not needed within a local␣
↪version of Jupyter notebook)
!apt-get -qq install texlive texlive-xetex texlive-latex-extra pandoc
!pip install --quiet pypandoc
```

```
Extracting templates from packages: 100%
Preconfiguring packages …
Selecting previously unselected package fonts-droid-fallback.
(Reading database … 155569 files and directories currently installed.)
Preparing to unpack …/00-fonts-droid-fallback_1%3a6.0.1r16-1.1_all.deb …
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1) …
Selecting previously unselected package fonts-lato.
Preparing to unpack …/01-fonts-lato_2.0-2_all.deb …
Unpacking fonts-lato (2.0-2) …
Selecting previously unselected package poppler-data.
Preparing to unpack …/02-poppler-data_0.4.8-2_all.deb …
Unpacking poppler-data (0.4.8-2) …
Selecting previously unselected package tex-common.
Preparing to unpack …/03-tex-common_6.09_all.deb …
Unpacking tex-common (6.09) …
Selecting previously unselected package fonts-lmodern.
Preparing to unpack …/04-fonts-lmodern_2.004.5-3_all.deb …
Unpacking fonts-lmodern (2.004.5-3) …
Selecting previously unselected package fonts-noto-mono.
Preparing to unpack …/05-fonts-noto-mono_20171026-2_all.deb …
Unpacking fonts-noto-mono (20171026-2) …
Selecting previously unselected package fonts-texgyre.
Preparing to unpack …/06-fonts-texgyre_20160520-1_all.deb …
Unpacking fonts-texgyre (20160520-1) …
```

[ ]:
```
# TO DO: Provide the full path to your Jupyter notebook file
!jupyter nbconvert --to PDF "/content/drive/MyDrive/5554/HW2/
↪Homework2_ankitparekh.ipynb"
```

```
#os.chdir('/content/drive/MyDrive/5554/HW2')
```