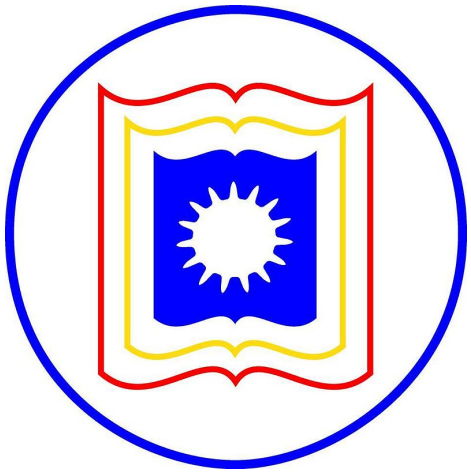
University of Rajshahi



Assignments of DIP Lab

Course Name: Digital Image Processing Lab
Course Code: CSE-4182
Date: September 16, 2022

Prepared by
Name: Md. Meem Mursalin Chowdhury
ID: 1810276103
Year: 4th, Semester: Odd
Session: 2017-18

Submitted to
Md. Khademul Islam Molla, Professor
MD. Rokanujjaman, Professor
Sangeeta Biswas, Associate Professor

Dept. of Computer Science and Engineering,
University of Rajshahi

Abstract

Digital Image Processing or DIP is a very well-known term to most of the people. It refers to the processing of different images. This processing is done images on basis of the need and demand. In this lab all the images are processed using python3 and the grayscale images or 2D images are considered.

Contents

1 Assignment-12

1.1 Introduction

Problem: Write a report on what different convolution layers of a CNN based image classifier see. You may use the attached code. You need to install Tensorflow in your computer. You can run it in Google Colab for GPU support if you do not have GPU or enough memory.

Solution: Here we have provide a report on what happens on different convolution layers of a CNN based image classifier. As the CNN model uses different layers, we can see different things happens to the images in different layers. The result and code of it given as follows.

1.2 Required Software

Here we have to install tensorflow which should be installed in a virtual environment. So at fist we have to create a virtual environment then install tensorflow. Again in the virtual environment, we have to install matplotlib, opencv and tk for generating the output. For the first execution of the program, the tensorflow model has to be loaded, so it will take some time to download. After that the program will execute fast.

1.3 Procedure

Step-1: Install tensorflow, matplotlib, opencv libraries.
Step-2: Run the following code.

1.4 Code

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
import cv2
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Model

DIR = '/media/mursalin/New Volume/Image-Processing/code_13/'
# DIR = './'

def main():
  # Load a pre-trained model.
  baseModel = VGG16()
  baseModel.summary()

  # Prepare a new model having the desired layer as the output layer.
  for i in range(10):
    layer_number = i +1# *** Change layer number to see different convolution layer's output.
    print(layer_number)
    inputs = baseModel.input
    outputs = baseModel.layers[layer_number].output
    model = Model(inputs, outputs)

    # Prepare data.
    img = prepare_data()

    # Predict output of a specific layer.
    outputs = model.predict(img)

    # Display what different channls see.
    display_channels(outputs, layer_number)
```

```
31
32      def display_channels(chSet, layer_no):
33        plt.figure(figsize = (20, 20))
34        plt.suptitle('Layer-' + str(layer_no))
35        for i in range(9):
36          plt.subplot(3, 3, i + 1)
37          plt.title('Channel-' + str(i))
38          plt.imshow(chSet[0, :, :, i], cmap = 'gray')
39          plt.axis('off')
40
41        plt.savefig(DIR+'fig-'+str(layer_no)+'.jpg')
42        plt.show()
43        plt.close()
44
45      def prepare_data():
46        # Load an image
47        imgPath = DIR + 'rose.jpg'  #'Elephant.jpg' #'Baby.jpeg' #'Rose.jpeg' #'Boat.
    jpeg' #
48        bgrImg = cv2.imread(imgPath)
49        print(bgrImg.shape)
50
51        # Convert the image from BGR into RGB format
52        rgbImg = cv2.cvtColor(bgrImg, cv2.COLOR_BGR2RGB)
53
54        # Reshape the image so that it can fit into the model.
55        #display_img(rgbImg)
56        rgbImg = cv2.resize(rgbImg, (224, 224))
57        display_img(rgbImg)
58
59        # Expand dimension since the model accepts 4D data.
60        print(rgbImg.shape)
61        rgbImg = np.expand_dims(rgbImg, axis = 0)
62        print(rgbImg.shape)
63
64        # Preprocess image
65        rgbImg = preprocess_input(rgbImg)
66
67        return rgbImg
68
69      def display_img(img):
70        plt.imshow(img)
71        plt.show()
72        plt.close()
73
74      if __name__ == '__main__':
75        main()
76
77
```

Listing 1: Code for using pre trained model(VGG16)

1.5 Result Discussion

As we are using the VGG16 model as CNN model, there are 16 different layers are produced here. As different convolution layer are present there, so if we give input of an image, we will get different kinds of output for each kind of layers. For this experiment, we have taken output of 9 picture for each layer and found out the difference. The output picture of 10 different layers is given below. From the 1st layer, we can see that the output picture can easily be recognised. Here the edges of the object can easily be understood and here, sobel filter is perhaps used. From layer-2's output, the picture continues to become blur and at the end of layer 10, the picture can no more be understood. After the finished layer output, the picture can no more be recognised.

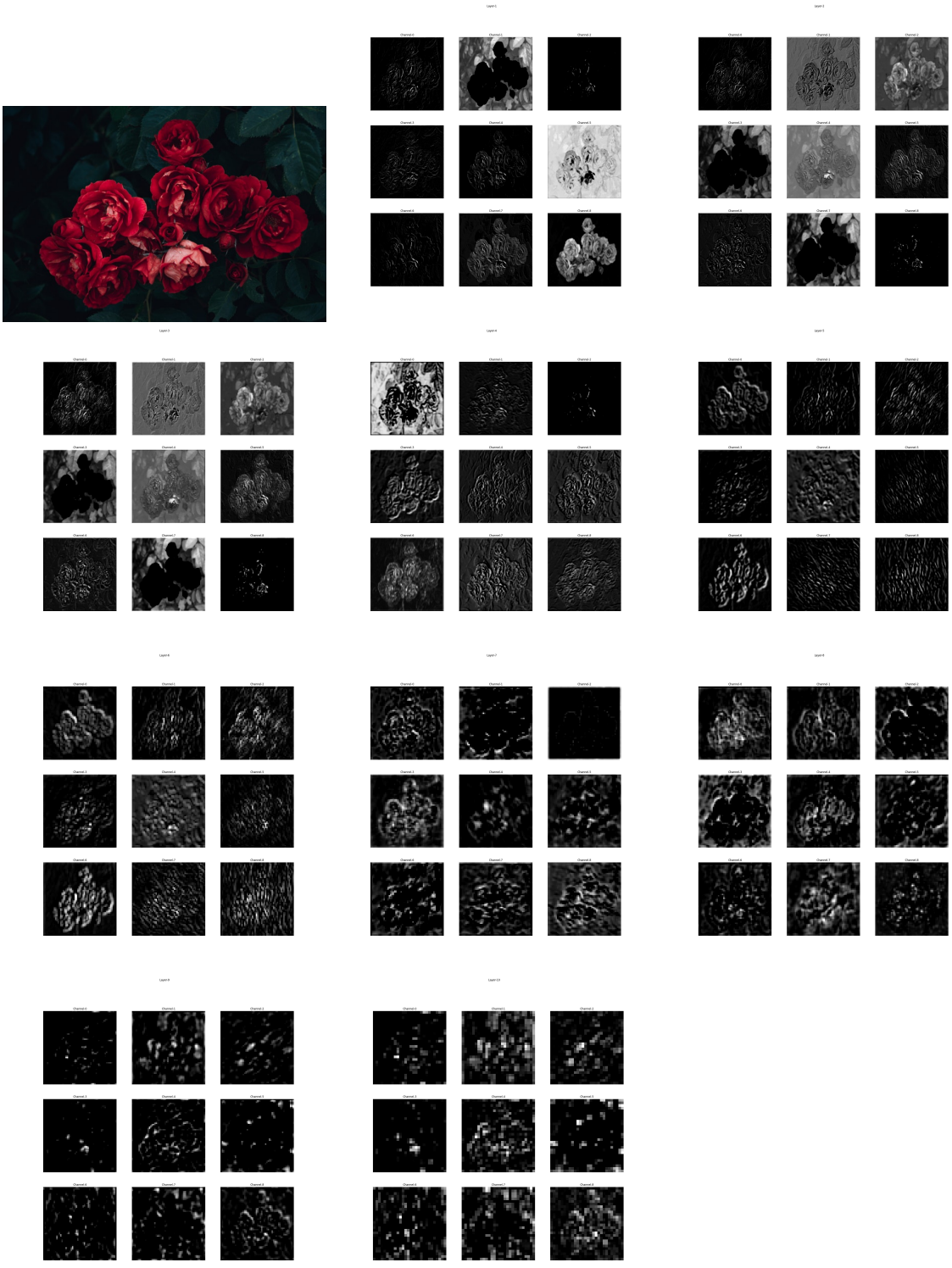


Figure 1: Input image, Output of Different Layers of VGG16 model from layer 1-10

2 Assignment-13

2.1 Introduction

Problem: Write a program to turn a JPEG image into a PNG image and vice-versa.

Solution: Here we have to load a jpeg typed image at first, then we have to convert it to png and again, we have to load a png image and convert it to jpeg format. At the time time of conversion, we have to check the metadata of the header of the image and find the difference. So, we need two codes here for each of the conversion and showing the metadata of image header for better understanding. The codes, images and outputs are given below.

2.2 Required Software

Here we have to at first, install Pillow for image conversion and also for getting the metadata of the image header file. We specifically import the Image from PIL and PIL.ExifTags for the metadata extraction. We can also use imghdr which is very much helpful in showing the format of the image.

2.3 Procedure

Step-1: Install PILLOW and imghdr, import them in required program.
Step-2: Show image type. (using imghdr.what())
Step-3: In code, read image of type jpeg or png. (usning Image.open())
Step-4: Extract metadata from the image and save it in a dictionary. Then show the metadata in terminal.
Step-5: Convert the image in png or jpeg format (using .save(), .covert()) and save it.

2.4 Code

```
    from PIL import Image
    from PIL.ExifTags import TAGS
    import imghdr

    def main():
        # path to the image or video
        img_path = './tower.jpg'
        print(img_path)
        print(imghdr.what(img_path))

        # read the image data using PIL
        image = Image.open(img_path)
        # print(image)

        # extract other basic metadata
        info_dict = {
            "Filename" : image.filename,
            "Image Size" : image.size,
            "Image Height" : image.height,
            "Image Width" : image.width,
            "Image Format" : image.format,
            "Image Mode" : image.mode,
            "Animated Image" : getattr(image, "is_animated", False),
            "Frames in Image" : getattr(image, "n_frames", 1)
        }

        # showing basic information
        for label, value in info_dict.items():
```

```
            print(f"{label:25}: {value}")

        # extract EXIF data
        exifdata = image.getexif()

        # iterating over all EXIF data fields
        for tag_id in exifdata:
            # get the tag name, instead of human unreadable tad id
            tag = TAGS.get(tag_id, tag_id)
            data = exifdata.get(tag_id)
            # decode bytes
            if isinstance(data, bytes):
                data = data.decode()
            print(f"{tag:25}: {data}")

        # jpg to png conversion
        image.save("tower.png")


    if __name__ == '__main__':
        main()


```

Listing 2: Code for JPEG to PNG Image Conversion

```
./tower.jpg
jpeg
Filename                  : ./tower.jpg
Image Size                : (640, 480)
Image Height              : 480
Image Width               : 640
Image Format              : JPEG
Image Mode                : RGB
Animated Image            : False
Frames in Image           : 1

```

Listing 3: Output of JPEG to PNG Image Conversion

```
from PIL import Image
from PIL.ExifTags import TAGS
import imghdr

def main():
    # path to the image or video
    img_path = './tower.png'
    print(img_path)
    print(imghdr.what(img_path))

    # read the image data using PIL
    image = Image.open(img_path)
    print(image)

    # extract other basic metadata
    info_dict = {
        "Filename" : image.filename,
        "Image Size" : image.size,
        "Image Height" : image.height,
        "Image Width" : image.width,
        "Image Format" : image.format,
        "Image Mode" : image.mode,
        "Animated Image" : getattr(image, "is_animated", False),
        "Frames in Image" : getattr(image, "n_frames", 1)
    }

    # showing basic information
    for label, value in info_dict.items():
```

```
            print(f"{label:25}: {value}")

        # extract EXIF data
        exifdata = image.getexif()

        # iterating over all EXIF data fields
        for tag_id in exifdata:
            # get the tag name, instead of human unreadable tad id
            tag = TAGS.get(tag_id, tag_id)
            data = exifdata.get(tag_id)
            # decode bytes
            if isinstance(data, bytes):
                data = data.decode()
            print(f"{tag:25}: {data}")

        # jpg to png conversion
        image2 = image.convert('RGB')
        image2.save("tower2.jpeg")

    if __name__ == '__main__':
        main()


```

Listing 4: Code for PNG to JPEG Image Conversion

```
./tower.png
png
Filename                  : ./tower.png
Image Size                : (640, 480)
Image Height              : 480
Image Width               : 640
Image Format              : PNG
Image Mode                : RGB
Animated Image            : False
Frames in Image           : 1

```

Listing 5: Output of PNG to JPEG Image Conversion

2.5 Result Discussion

Here in the output, we can see that the data is showing accordingly to the result. No inconsistency or error is observed. The input and output images are given below accordingly.

Figure 2: Input image of jpeg to png image conversion, Output image of jpeg to png image conversion and input image of png to jpeg image conversion, Output image of png to jpeg image conversion

3 Assignment-14

3.1 Introduction

Problem: Write a program to perform JPEG compression on frequency domain.

Solution: Here we have to perform JPEG compression on frequency domain. It may be noted here that if we do JPEG compression based on frequency domain, the dimension of image is not changed, rather the size of image is changed. This happens because as we are compressing the image based on the frequency, the highest frequencies are kept in the image and other lower frequencies are cut off. The phrase cut off means that frequencies energy becomes zero. So no information of that lower frequencies are saved. And we call that lower frequency as noise. Here we have do the same things. The code for image compression and its outcome is given below accordingly.

3.2 Required Software

Here for image compression, we have to use numpy, matplotlib and opencv(cv2) libraries for the code. These libraries are enough for our workings. We use matplotlib.pyplot for image taking input image and save output images, opencv(cv2) for image conversion and numpy for all other calculation.

3.3 Procedure

Step-1: Install numpy, matplotlib and opencv and import them in required program.
Step-2: Read image (using plt.imread()) and convert it to grayscale (using cv2.cvtColor())
Step-3: Fourier transform the image and sort it by highest magnitude.
Step-4: Find out the threshold for compressing the image.
Step-5: Find out the indices of small frequency.
Step-6: Threshold the small indices.
Step-7: Compressed the image by Inverse Fourier Transform.
Step-8: Save the images and plot them.

3.4 Code

```
    import matplotlib.pyplot as plt
    import numpy as np
    import cv2 as cv

    def main():
        img_path = './trees_in_water.jpg'
        print(img_path)

        img_rgb = plt.imread(img_path)
        print(img_rgb.shape)

        img_gray = cv.cvtColor(img_rgb, cv.COLOR_RGB2GRAY)
        print(img_gray.shape)
        cnt = 0
        # plt.imshow(img_gray, cmap='gray')
        # plt.savefig('fig-0.jpg')
        # plt.show()
        plt.imsave('./fig-' + str(cnt) + '.jpg', img_gray, format='jpg', cmap='gray')

        img_set = [img_gray]
        img_title = ['Grayscale']

```

```
        ft = np.fft.fft2(img_gray)
        ftSort = np.sort(np.abs(ft.reshape(-1))) # sort by highest magnitude

        cnt = 0
        # Zero out all the small co-efficients and inverse transformation
        for keep in (0.75 , 0.5 , 0.1 , 0.05 , 0.01, 0.001 , 0.0001):
            thresh = ftSort[int(np.floor((1-keep)*len(ftSort)))]
            ind = np.abs(ft) > thresh           # find small indices
            Atlow = ft * ind                    # threshold small indices
            ftlow = np.fft.ifft2(Atlow).real    # compressed image
            img_set.append(ftlow)
            img_title.append('Image at ' + str(keep*100) + '%' + ' compression')
            cnt += 1
            plt.imsave('./fig-' + str(cnt) + '.jpg', ftlow, format='jpg', cmap='gray'
    )
            print(ftlow.shape)

        img_plot(img_set, img_title)

    def img_plot(img_set, img_title):
        n = len(img_set)
        plt.figure(figsize = (20, 20))
        for i in range(n):
            plt.subplot(2, 4, i+1)
            plt.imshow(img_set[i], cmap='gray')
            plt.title(img_title[i])

        plt.savefig('output.jpg')
        plt.show()

    if __name__ == '__main__':
        main()

    # image save option
    # import matplotlib.image as mpimg

    # img = mpimg.imread("src.png")
    # mpimg.imsave("out.png", img)


```

Listing 6: Code for Image Compression on Frequency Domain

3.5 Result Discussion

Here in the output, we can see that the size of image changes with the compression percentage but the height and width remains same. Sometimes the size is increasing and other times, it is decreasing. In the output figure, at first the grayscale image is shown which is used for compression. It has size of 1.3 MB (1,324,171 bytes). The 1st output image of 75% image compression has size of 1.3 MB (1,335,669 bytes). So, the size has increased. The 2nd output image of 50% image compression has size of 1.3 MB (1,323,616 bytes). So, the size has decreased. The 3rd output image of 10% image compression has size of 1.5 MB (1,489,565 bytes). So, the size has increased. The 4th output image of 5% image compression has size of 1.5 MB (1,507,115 bytes). So, the size has increased. The 5th output image of 1% image compression has size of 1.1 MB (1,114,207 bytes). So, the size has decreased. The 6th output image of 0.1% image compression has size of 766.9 kB (766,909 bytes). So, the size has decreased. The 7th output image of 0.01% image compression has size of 636.4 kB (636,408 bytes). So, the size has decreased. This has happened because perhaps when the compression technique is applied, as for unique values, the storing values increases which results in increasing the size of output image.

Figure 3: Input image of grayscale, output images of compression of 75%, 50%, 10%, 5%, 1%, 0.1%, 0.01% respectively