ICE-8 Image Filtering

(Tutorial) CNN

Feature detectors can be set up with different values to get different results.

For example, a filter can be applied that can sharpen and focus an image or blur an image. That would give equal importance to all the values. You can do edge enhancement, edge detection, and more. You would do that by applying different feature detectors to create different feature maps. The computer is able to determine which filters make the most sense and apply them.

The primary purpose here is to find features in your image, put them into a feature map, and still preserve the spatial relationship between pixels. That's important so that the pixels don't get all jumbled up.

- use pip install opencv-python to install cv2

Use this image https://drive.google.com/file/d/1OFAD7GGVQeKYPyp-0XZn3cxR4h9FblIz/view?usp=sharing

We're going to use this above given image for our input image.

We'll make it black and white

Upload the image above as cat.jpg in your notebook for the tutorial to work

```
!pip install opencv-python
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
img_path = 'cat.jpg'
# Load color image
bgr_img = cv2.imread(img_path)
# Convert to grayscale
gray_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY)
# Normalize, rescale entries to lie in [0,1]
gray_img = gray_img.astype("float32")/255
# Plot image
plt.imshow(gray_img, cmap='gray')
plt.show()
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.7.0.7
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from op
```

```
#Let’s define and visualize our filters

import numpy as np
filter_vals = np.array([[-1, -1, 1, 1], [-1, -1, 1, 1], [-1, -1, 1, 1], [-1, -1, 1, 1]])
print('Filter shape: ', filter_vals.shape)
```

```
Filter shape:  (4, 4)
```

```
# Define four different filters, all of which are linear combinations of the `filter_vals` defined ab
filter_1 = filter_vals
filter_2 = -filter_1
filter_3 = filter_1.T
filter_4 = -filter_3
filters = np.array([filter_1, filter_2, filter_3, filter_4])
# Print out the values of filter 1 as an example
print('Filter 1: \n', filter_1)
```

```
Filter 1:
 [[-1 -1  1  1]
 [-1 -1  1  1]
 [-1 -1  1  1]
 [-1 -1  1  1]]
```

```
#Now let’s define a convolutional layer (using PyTorch)



import torch
import torch.nn as nn
import torch.nn.functional as F

# Neural network with one convolutional layer with four filters
class Net(nn.Module):

    def __init__(self, weight):
        super(Net, self).__init__()
        # Initializes the weights of the convolutional layer to be the weights of the 4 defined filte
        k_height, k_width = weight.shape[2:]
        # Assumes there are 4 grayscale filters
        self.conv = nn.Conv2d(1, 4, kernel_size=(k_height, k_width), bias=False)
        self.conv.weight = torch.nn.Parameter(weight)
    def forward(self, x):
            # Calculates the output of a convolutional layer pre- and post-activation
            conv_x = self.conv(x)
            activated_x = F.relu(conv_x)

            # Returns both layers
            return conv_x, activated_x

# Instantiate the model and set the weights
weight = torch.from_numpy(filters).unsqueeze(1).type(torch.FloatTensor)
model = Net(weight)
# Print out the layer in the network
print(model)
```

```
     Net(
       (conv): Conv2d(1, 4, kernel_size=(4, 4), stride=(1, 1), bias=False)
     )
```

```
def viz_layer(layer, n_filters= 4):
    fig = plt.figure(figsize=(20, 20))

    for i in range(n_filters):
        ax = fig.add_subplot(1, n_filters, i+1, xticks=[], yticks=[])
        # Grab layer outputs
        ax.imshow(np.squeeze(layer[0,i].data.numpy()), cmap='gray')
        ax.set_title('Output %s' % str(i+1))
```

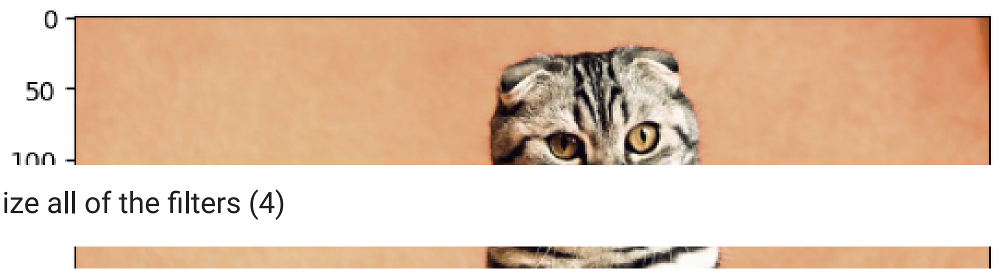
Task 1. Write code for the following tasks below :

- Plot original image
- Visualize all of the filters (4)
- Convert the image into an input tensor

Plot original image

```
#code here
plt.imshow(cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB))
```

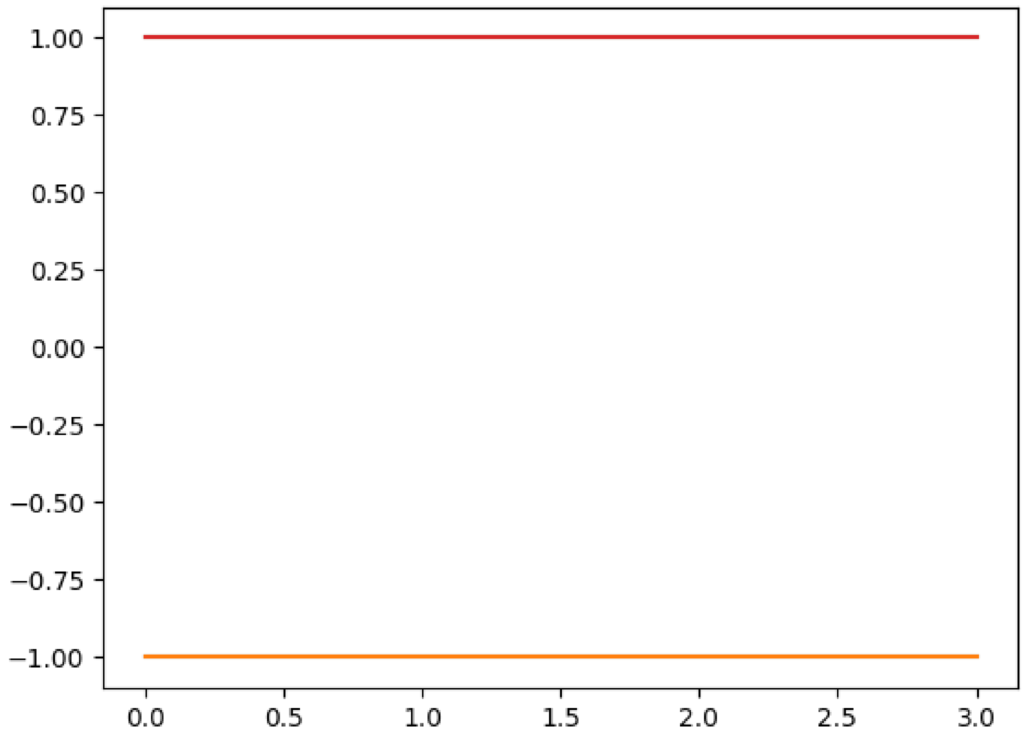
```
<matplotlib.image.AxesImage at 0x7b7e906af250>
```



Visualize all of the filters (4)

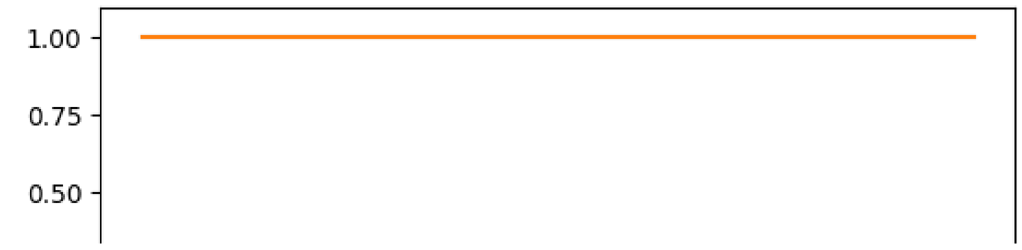
```
plt.plot(filter_1)
```

```
[<matplotlib.lines.Line2D at 0x7b7e8fb02c50>,
 <matplotlib.lines.Line2D at 0x7b7e8fb02cb0>,
 <matplotlib.lines.Line2D at 0x7b7e8fb02ce0>,
 <matplotlib.lines.Line2D at 0x7b7e8fb02dd0>]
```



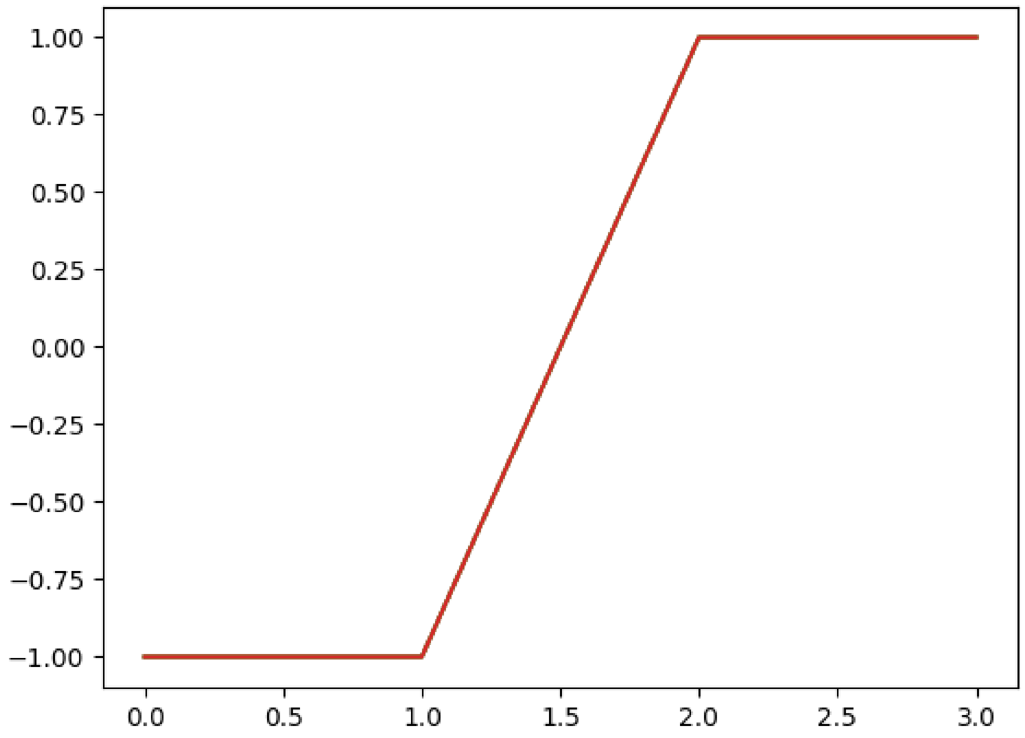
```
plt.plot(filter_2)
```

```
[<matplotlib.lines.Line2D at 0x7b7e8f99b400>,
 <matplotlib.lines.Line2D at 0x7b7e8f99b460>,
 <matplotlib.lines.Line2D at 0x7b7e8f99b490>,
 <matplotlib.lines.Line2D at 0x7b7e8f99b580>]
```



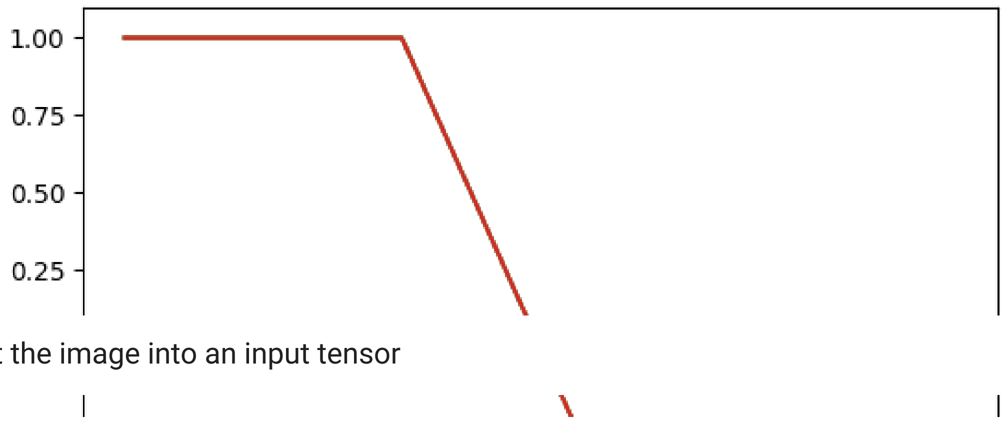
```
plt.plot(filter_3)
```

```
[<matplotlib.lines.Line2D at 0x7b7e8fa2b6d0>,
 <matplotlib.lines.Line2D at 0x7b7e8fa2b730>,
 <matplotlib.lines.Line2D at 0x7b7e8fa2b760>,
 <matplotlib.lines.Line2D at 0x7b7e8fa2b850>]
```



```
plt.plot(filter_4)
```

```
[<matplotlib.lines.Line2D at 0x7b7e8f8b73a0>,
 <matplotlib.lines.Line2D at 0x7b7e8f8b7400>,
 <matplotlib.lines.Line2D at 0x7b7e8f8b7430>,
 <matplotlib.lines.Line2D at 0x7b7e8f8b7520>]
```



Convert the image into an input tensor

```
import torch
from PIL import Image
from torchvision import transforms

img = Image.open("cat.jpg").convert("L")
convert_tensor = transforms.ToTensor()
tensor = convert_tensor(img)

tensor.shape
```

```
torch.Size([1, 426, 640])
```

Task 2. Write code for the following tasks below :

- Get the convolutional layer (pre and post activation)

  variables to use are **conv_lyr, changed_lyr**

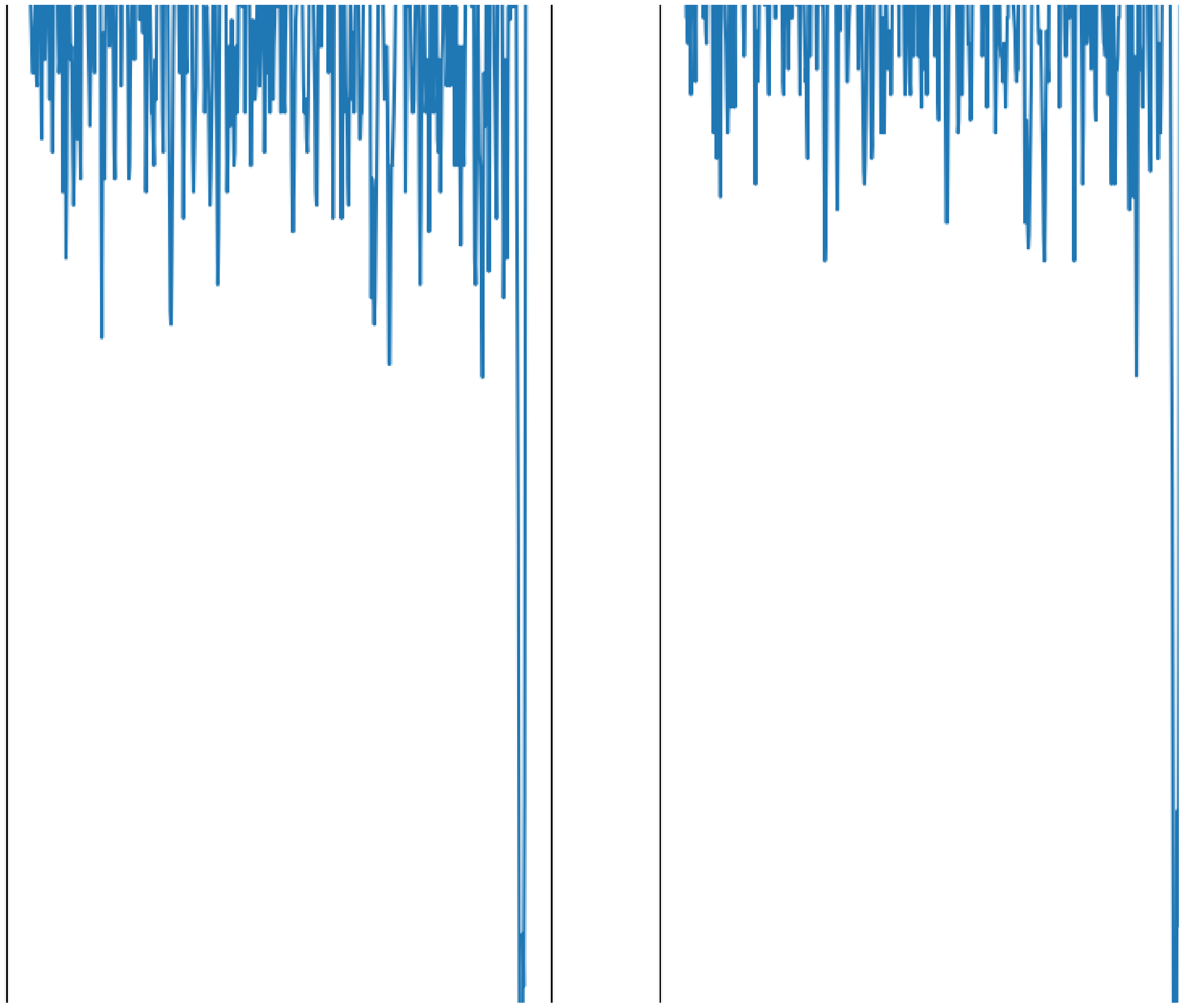
- Visualize the output of a convolutional layer

Get the convolutional layer (pre and post activation)

variables to use are conv_lyr, changed_lyr

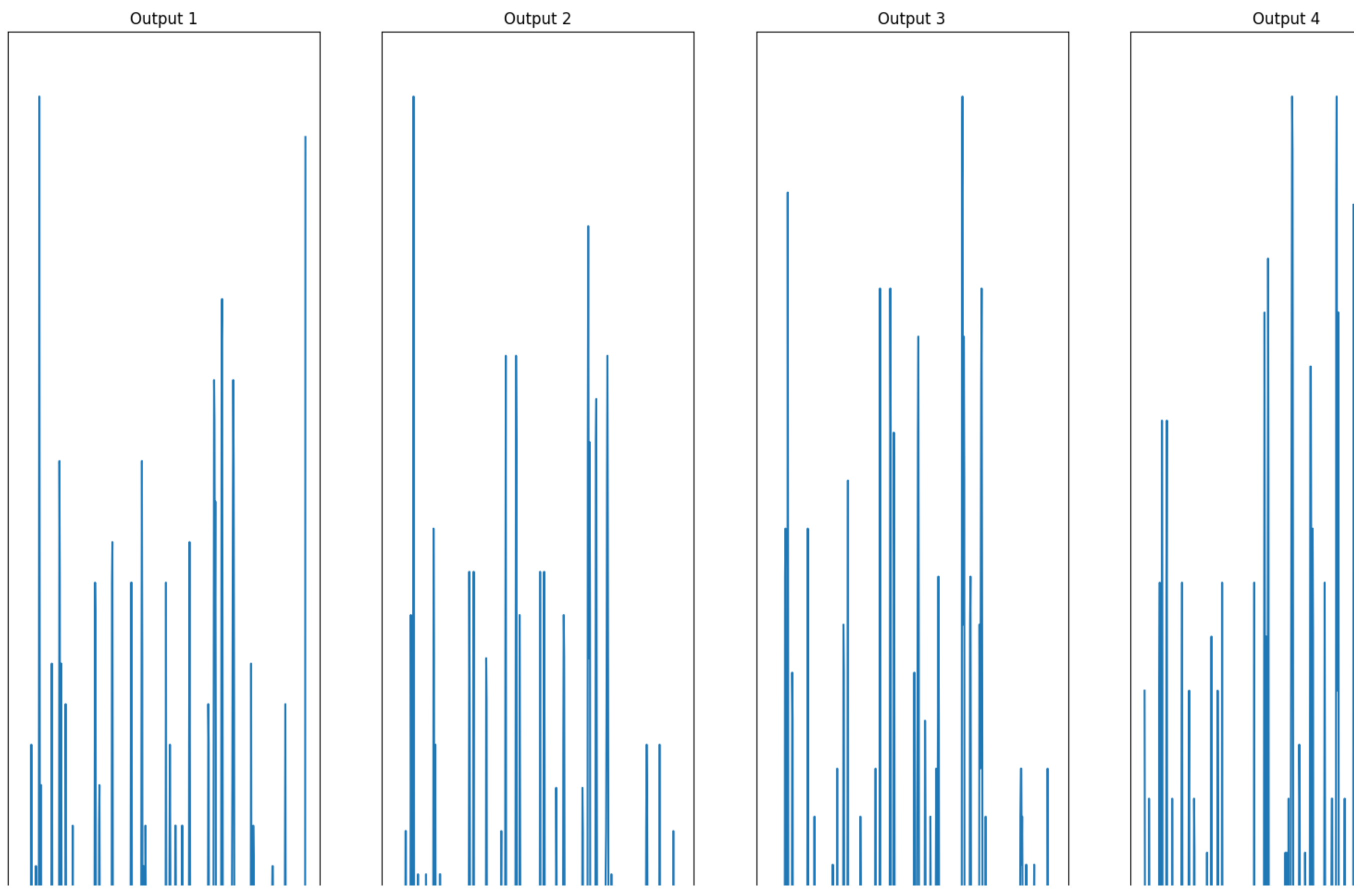
```
#code here
conv_lyr, changed_lyr = model(tensor)


def viz_layerr(layer, n_filters= 4):
    fig = plt.figure(figsize=(20, 20))

    for i in range(n_filters):
        ax = fig.add_subplot(1, n_filters, i+1, xticks=[], yticks=[])
        # Grab layer outputs
        ax.plot(np.squeeze(layer[0,i].data.numpy()))
        ax.set_title('Output %s' % str(i+1))
```

```
viz_layerr(conv_lyr, n_filters= 4)
```



Visualize the output of a convolutional layer

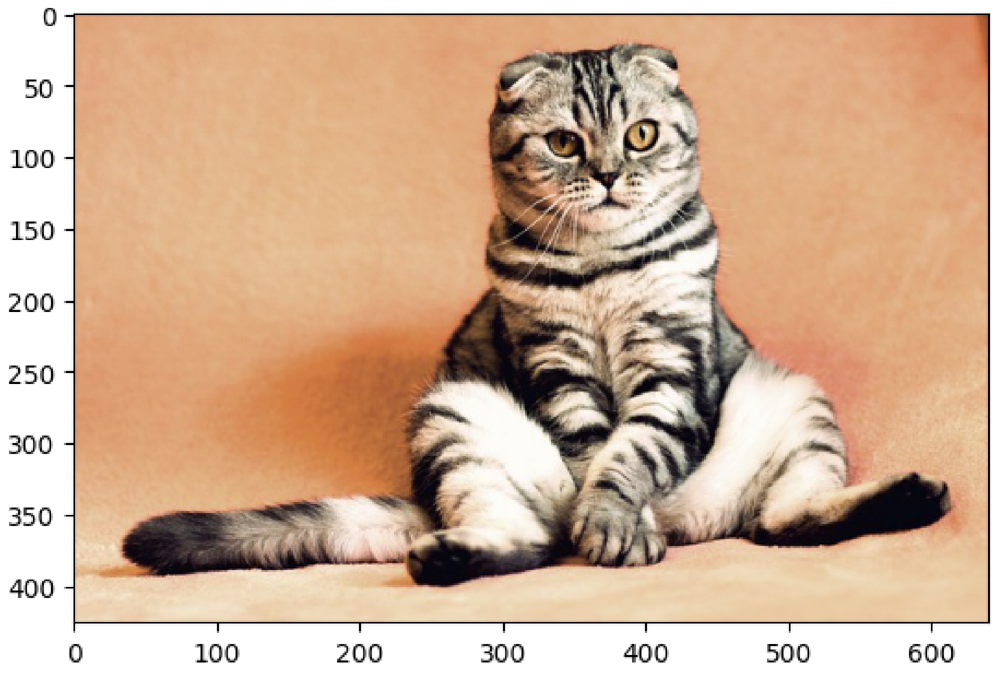
```
viz_layerr(changed_lyr, n_filters= 4)
```



Task 3. Write code for the following tasks below :

- convert image to grayscale for filtering and print it

```
#code here
import cv2

imag = cv2.imread("cat.jpg")
gry = cv2.cvtColor(imag, cv2.COLOR_BGR2GRAY)

plt.imshow(cv2.cvtColor(imag, cv2.COLOR_BGR2RGB))
plt.show()

plt.imshow(gry, cmap='gray')
plt.show()
```



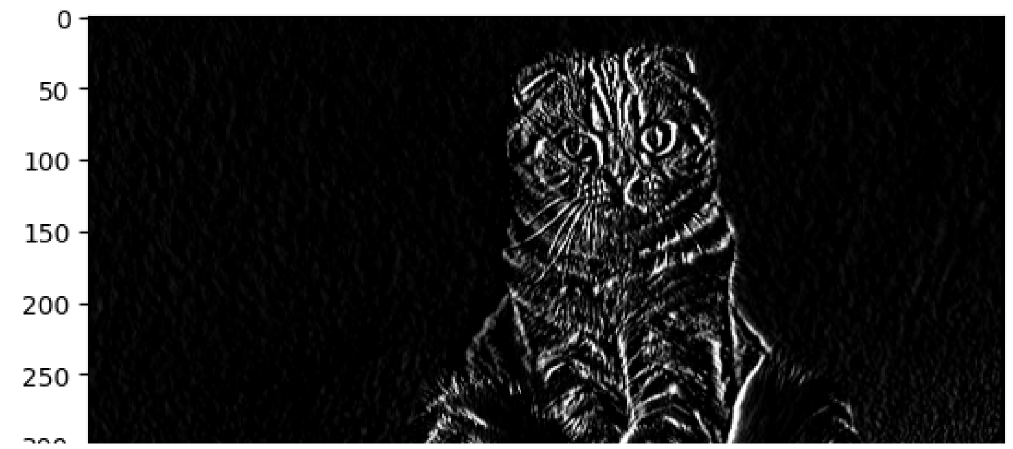
Task 4. Write code for the following tasks below :

- create a custom kernel and apply edge detection filter and print it

HINT : Sobel operator

```
#code here
sobel = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])

edges = cv2.filter2D(src=gry, ddepth=-1, kernel=sobel)
plt.imshow(edges, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7b7e8d9458d0>
```



Question 1. What are different types of edge detection filters .

State why sobel operator is useful here .

What did we acheive from it ?

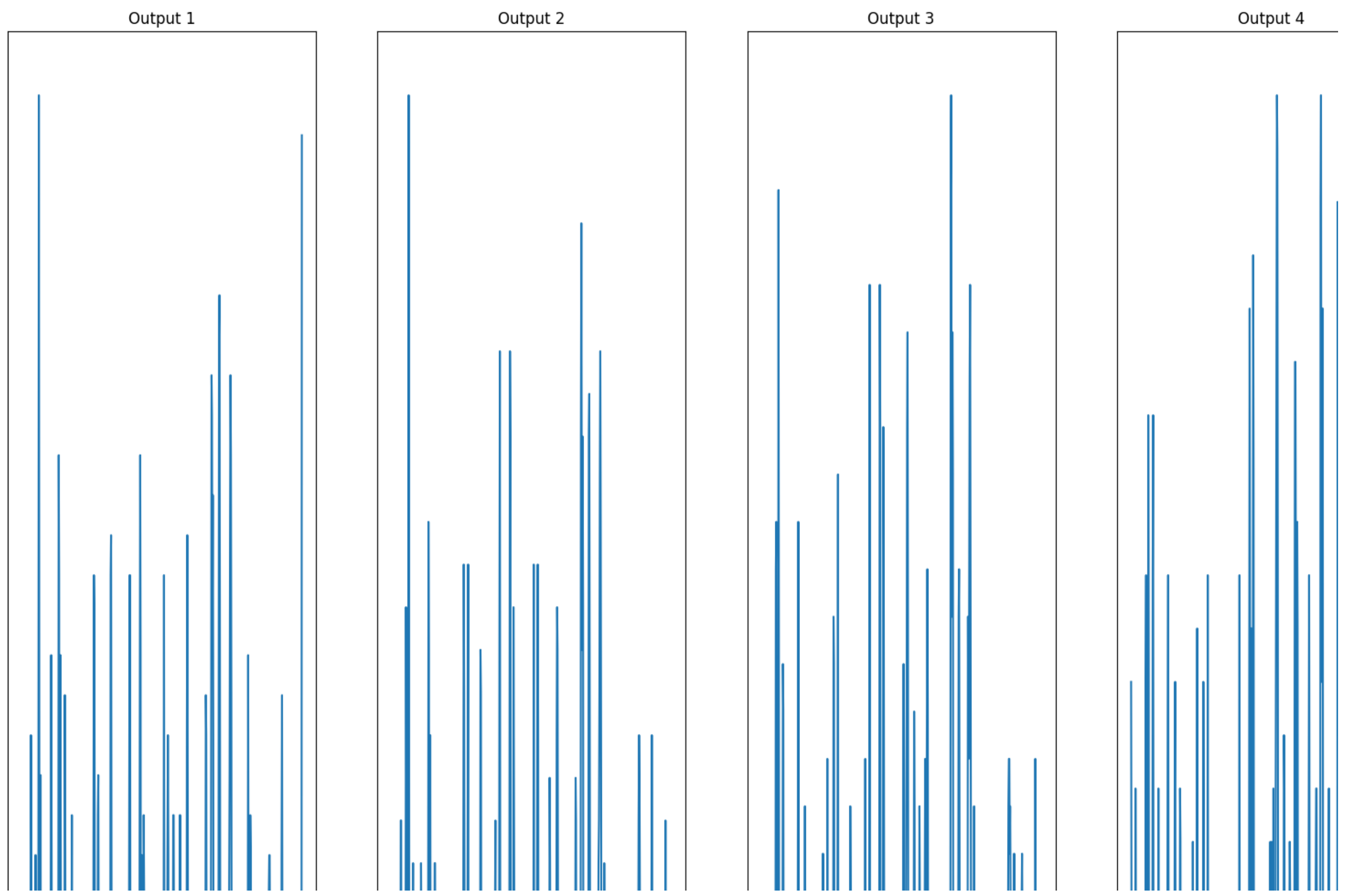
answer here

Task 5. No need to code for this task

If all tasks above are done properly this task will work

HINT : read task 2 question properly as I have pointed out what variable name to use which is used in this task

```
viz_layerr(changed_lyr)
```



Question 2. What are different types of edge detection filters .

HINT : Please refer lecture 8 slides .

Task 5 falls under which **building block of CNN** Explain the importance of that block in CNN. Explain why that block helps in task 5 and what it did .

Answer here