Fourier Transform Model of Image Formation

March 7, 2019

1 Fourier Transform Model of Image Formation

Kenneth V. Domingo
2015-03116
Physics 166 THV - Dr. Maricor Soriano
2nd Semester, A.Y. 2018-19
A live version of this notebook can be accessed on GitHub:
https://github.com/kvdomingo/Physics-166/blob/master/Fourier%20Transform%20Model%20of%20Imag

```
In [4]: import numpy as np
        import matplotlib.pyplot as mp
        import scipy.fftpack as fft
        import scipy.ndimage as img
        import scipy.signal as sig
        from PIL import Image, ImageDraw, ImageFont
        from jupyterthemes import jtplot
```

```
In [5]: jtplot.reset()
        mp.rcParams["figure.figsize"] = (5*16/9, 5)
        mp.rcParams["figure.dpi"] = 100
        mp.rcParams["text.usetex"] = True
        mp.rcParams["font.family"] = "serif"
```

```
In [9]: class ImgModel:
            """
            The ImgModel class makes it easier to generate circular apertures,
            images with text using only Python, and simulating an imaging system
            based on these generated elements. This eliminates the need to keep
            rewriting similar code.
            """


            def generate_aperture(rad=0.5, res=500):
                """
                Generate a binary circular aperture

                Parameters
```

```python
    ----------
    rad : float
        radius of the aperture relative to the size of the image
    res : int
        resolution of the image meshgrid

    Returns
    -------
    Z : array_like
        numpy array containing the aperture
    """
    t = np.linspace(-1, 1, res)
    X, Y = np.meshgrid(t,t)
    R = np.hypot(X, Y)
    Z = np.zeros((res, res))
    Z[R <= rad] = 1
    Z = np.round(Z/Z.max() * 255).astype("uint8")
    return Z


def generate_txtImage(x, y, dims=512, text=None, fontsize=256,
                      fontcolor="white", bgcolor="black",
                      supersample=True, superes=128):
    """
    Generate an image with text in-program

    Parameters
    ----------
    x : int
        x-location of the upper-left corner of text
    y : int
        y-location of the upper-left corner of text
    dims : int
        (square) dimensions of the image
    text : str
        text to be put in the image
    fontsize : float
        size of text
    fontcolor : str
        color of text
    bgcolor : str
        color of image background
    supersample : bool
        whether or not to supersample the image to reduce aliasing.
        Supersampled dimension is given by the dims parameter, and the
        final size is given by the superes parameter
    superes : int
        final downscaled size of image after supersampling. Will be
```

```python
            ignored if supersample is false

        Returns
        -------
        a : array_like
            numpy array containing the image with text, of size dims x dims
            if supersample is True; size superes x superes otherwise
        """
        a = Image.new("L", (dims, dims), color=bgcolor)
        fnt = ImageFont.truetype("C:/Windows/Fonts/Arial.ttf", fontsize)
        d = ImageDraw.Draw(a)
        d.text((x, y), text, font=fnt, fill=fontcolor)
        if supersample:
            a = a.resize((superes, superes), Image.ANTIALIAS)
        a = np.array(a, "uint8")
        return a

    def image_aperture(obj, aper):
        """
        Simulate imaging device. Inputs should ideally be the outputs of
        generate_aperture() and generate_txtImage(), and should be of
        the same size

        Parameters
        ----------
        obj : array_like
            Output of generate_txtImage(), or any other image with the same
            dimensions as aper
        aper : array_like
            Output of generate_aperture, or any other aperture with the
            same dimensions as obj

        Returns
        -------
        FImage : array_like
            Image formed by multiplying the FTs of obj and aper
        """
        Faper = fft.fftshift(aper)
        Fobj = fft.fft2(obj)
        FRA = Faper*Fobj
        IRA = abs(fft.fft2(FRA))
        FImage = np.round(IRA/IRA.max() * 255).astype("uint8")
        return FImage

def uint8(X):
    """
    Convert input data type to 8-bit unsigned integer
    """
```

```python
    return np.round(abs(X)/abs(X).max() * (2**8 - 1)).astype("uint8")

def fftcircle(Z, r, h, k, **kwargs):
    """
    Draw a circle on an existing image

    Parameters
    ----------
    Z : PIL.Image
        An instance of a blank or pre-existing PIL.Image object
    r : int
        radius of circle
    h : int
        x-location of center of circle
    k : int
        y-location of center of circle
    """
    draw = ImageDraw.Draw(Z)
    draw.ellipse((h-r, k-r, h+r, k+r), **kwargs)

def fftsquare(Z, r, h, k, **kwargs):
    """
    Draw a square on an existing image

    Parameters
    ----------
    Z : PIL.Image
        An instance of a blank or pre-existing PIL.Image object
    r : int
        half-length of a side of the square
    h : int
        x-location of center of square
    k : int
        y-location of center of square
    """
    draw = ImageDraw.Draw(Z)
    draw.rectangle((h-r, k-r, h+r, k+r), **kwargs)

def fftgauss(X, Y, mux, muy, sigma):
    """
    Generate a 2D Gaussian

    Parameters
    ----------
    X : array_like
        meshgrid of x values
    Y : array_like
        meeshgrid of y values
```

```
    mux : float
        x-location of mean
    muy : float
        y-location of mean
    sigma : float
        standard deviation of the gaussian
    """
    return np.exp(-((X - mux/len(X))**2 + (Y - muy/len(Y))**2)**2/sigma**2)
```

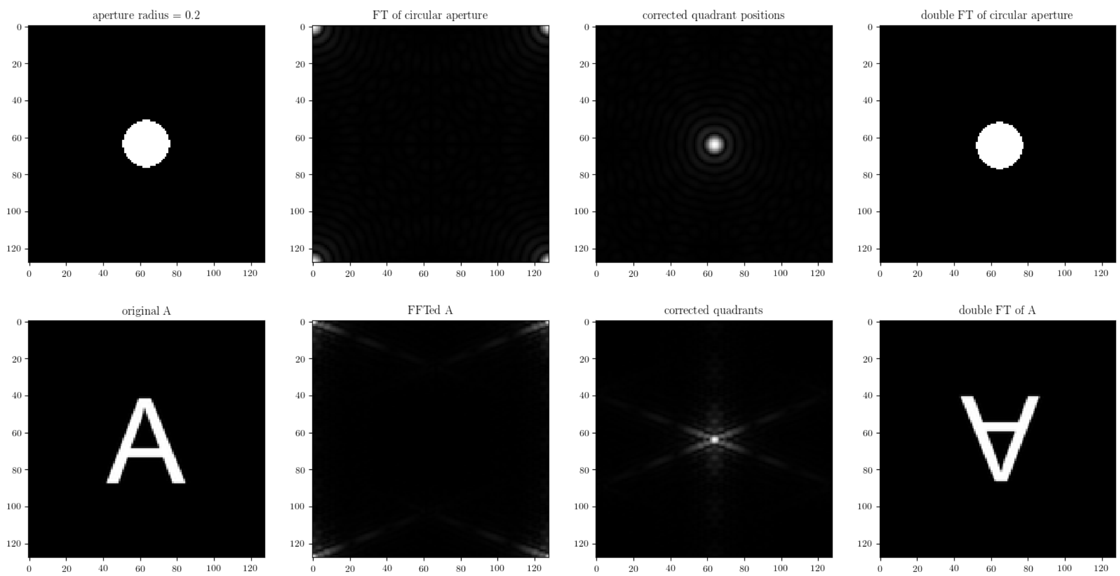
1.1 Activity 1. Familiarization with discrete FFT

A 128×128 Boolean image of a centered circle and a centered letter "A" is created, and the 2D FFT is taken.

```
In [7]: Z = ImgModel.generate_aperture(0.2, 128)
        A = ImgModel.generate_txtImage(170, 120, text="A")
        fig = mp.figure(figsize=(5*4, 5*2))

        ax = fig.add_subplot(241)
        ax.imshow(Z, "gray")
        ax.set_title("aperture radius = 0.2")

        ax = fig.add_subplot(242)
        FZ = fft.fft2(Z)
        ax.imshow(uint8(FZ), "gray")
        ax.set_title("FT of circular aperture")

        ax = fig.add_subplot(243)
        ax.imshow(uint8(fft.fftshift(FZ)), "gray")
        ax.set_title("corrected quadrant positions")

        ax = fig.add_subplot(244)
        ax.imshow(uint8(fft.fft2(FZ)), "gray")
        ax.set_title("double FT of circular aperture")

        ax = fig.add_subplot(245)
        ax.imshow(A, "gray")
        ax.set_title("original A")

        ax = fig.add_subplot(246)
        FA = fft.fft2(A)
        ax.imshow(uint8(FA), "gray")
        ax.set_title("FFTed A")

        ax = fig.add_subplot(247)
        ax.imshow(uint8(fft.fftshift(FA)), "gray")
        ax.set_title("corrected quadrants")
```

```
ax = fig.add_subplot(248)
ax.imshow(uint8(fft.fft2(FA)), "gray")
ax.set_title("double FT of A")

mp.show()
```

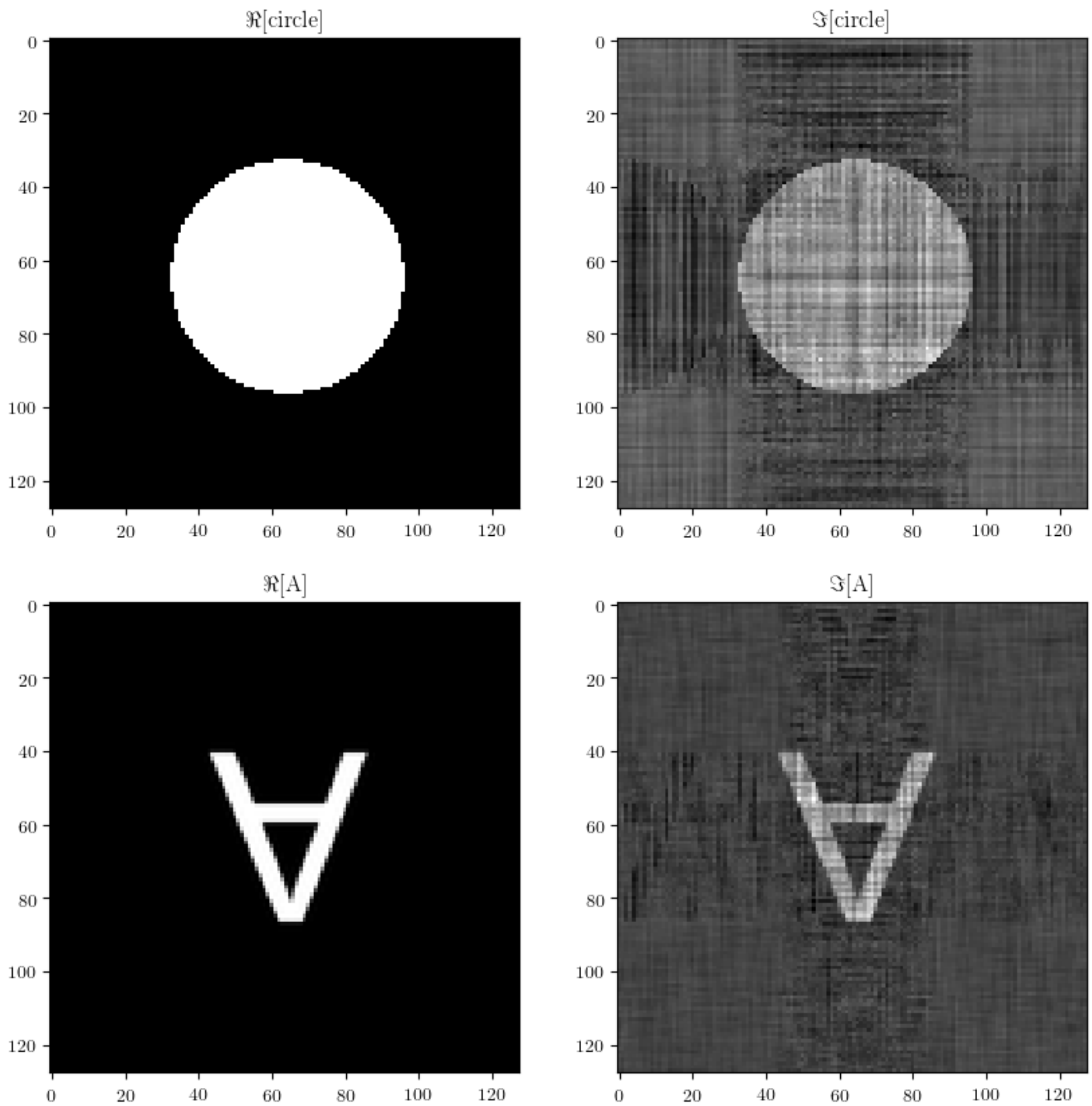
```
D:\ProgramData\Anaconda3\envs\compsense\lib\site-packages\scipy\fftpack\basic.py:159: FutureWa
  z[index] = x
```



In the 2nd column, we see the quarters of a pattern in each corner. Applying fftshift() corrects the quadrant positions and we observe some sort of Airy pattern at the center. Applying fft2() again reverts them back to the spatial domain.

```
In [29]: fig = mp.figure(figsize=(5*2, 5*2))

         ax = fig.add_subplot(221)
         ax.imshow(fft.fft2(FZ).real, "gray")
         ax.set_title(r"$\Re$[circle]")

         ax = fig.add_subplot(222)
         ax.imshow(fft.fft2(FZ).imag, "gray")
         ax.set_title(r"$\Im$[circle]")

         ax = fig.add_subplot(223)
         ax.imshow(fft.fft2(FA).real, "gray")
         ax.set_title(r"$\Re$[A]")

         ax = fig.add_subplot(224)
         ax.imshow(fft.fft2(FA).imag, "gray")
```

```
ax.set_title(r"$\Im$[A]")

mp.show()
```

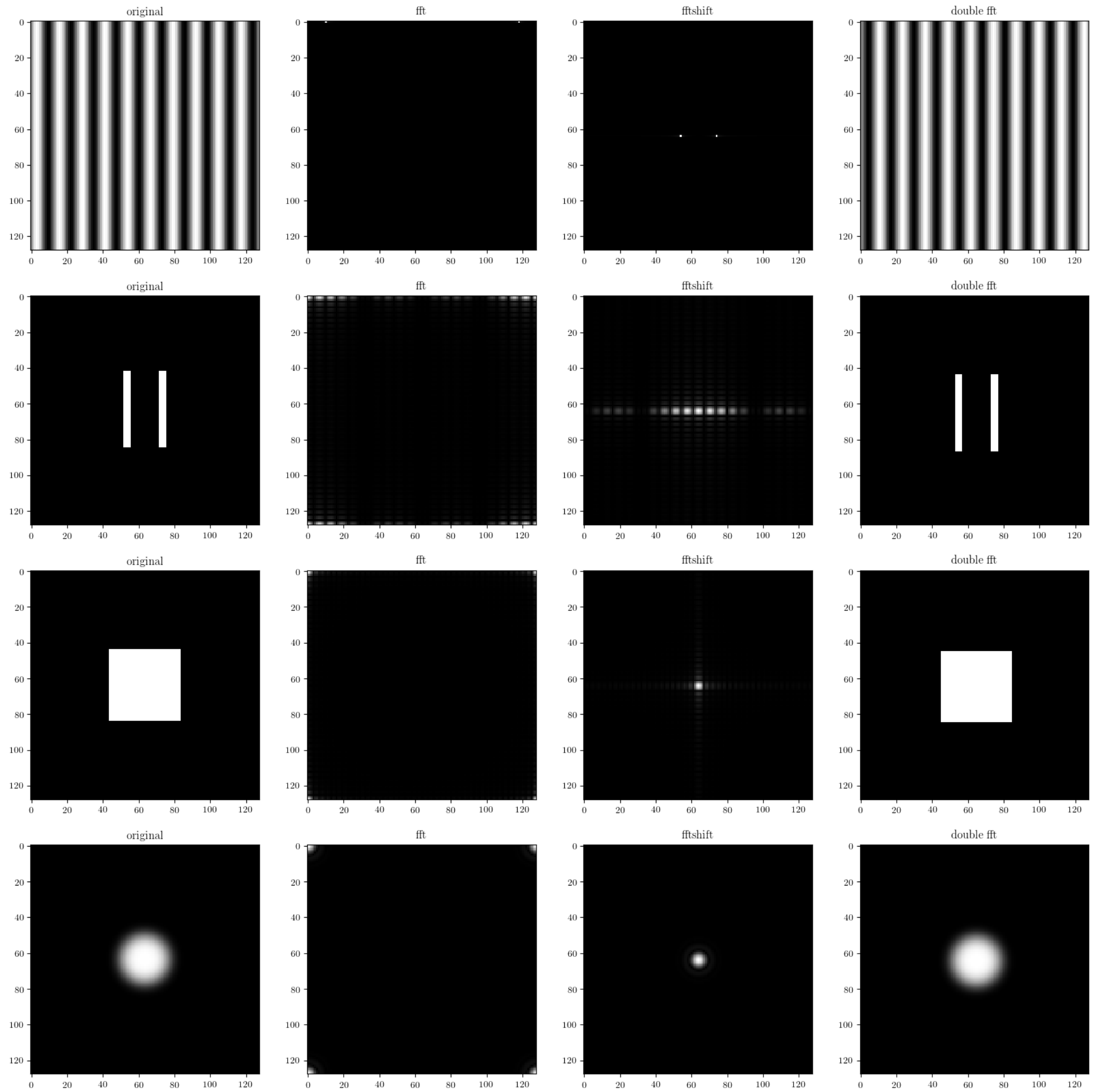


One can see that the FT of an image has both real and imaginary parts. This arises from the definition of the 2D Fourier transform:

$$F(f_x, f_y) = \int \int f(x,y) e^{-2\pi i (f_x x + f_y y)} dx dy$$

```
In [8]: L = 128
        Z = np.zeros((L, L, 4))
        t = np.linspace(-10*np.pi, 10*np.pi, L)
        X,Y = np.meshgrid(t,t)
```

```python
# Sine along x
Z[:,:,0] = np.sin(X)

# Double slit
Z[L//3:2*L//3, L//2-10-2:L//2-10+2, 1] = 1
Z[L//3:2*L//3, L//2+10-2:L//2+10+2, 1] = 1

# Square
Z[L//2-20:L//2+20, L//2-20:L//2+20, 2] = 1

# 2D Gaussian
Z[:,:,3] = fftgauss(X, Y, 0, 0, 50)

fig = mp.figure(figsize=(5*4, 5*4))

for i in range(4):
    ax = fig.add_subplot(4, 4, 4*i + 1)
    ax.imshow(Z[:,:,i], "gray")
    ax.set_title("original")

    ax = fig.add_subplot(4, 4, 4*i + 2)
    FZ = fft.fft2(Z[:,:,i])
    ax.imshow(uint8(FZ), "gray")
    ax.set_title("fft")

    ax = fig.add_subplot(4, 4, 4*i + 3)
    ax.imshow(uint8(fft.fftshift(FZ)), "gray")
    ax.set_title("fftshift")

    ax = fig.add_subplot(4, 4, 4*i + 4)
    ax.imshow((fft.fft2(FZ)).real, "gray")
    ax.set_title("double fft")

mp.show()
```

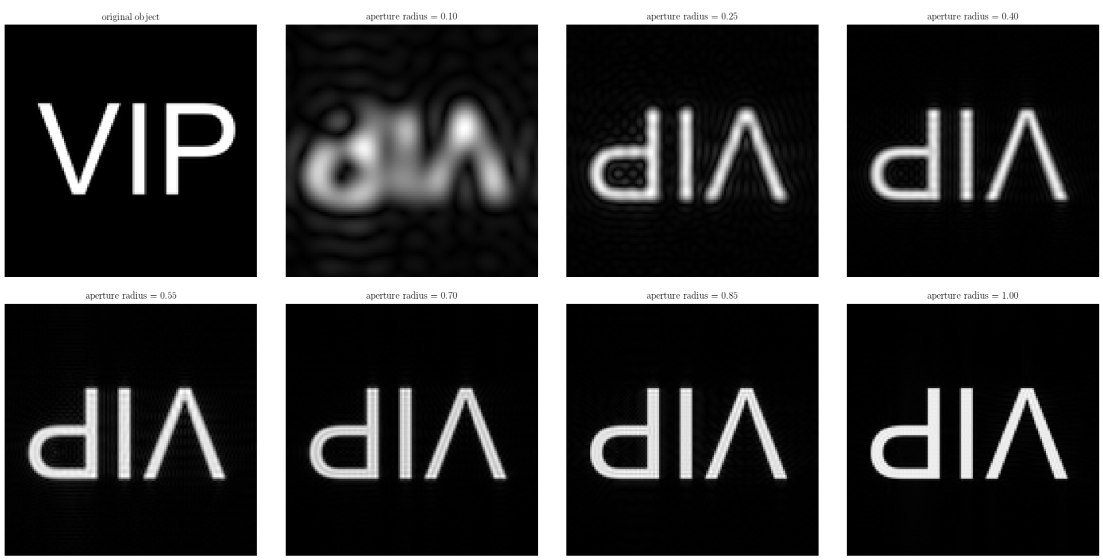


- The fft2() of a 2D sine wave of fixed frequency are dots corresponding to the actual (positive) and aliased (negative) frequencies, similar to the fft() of a temporal signal.
- The fft2() of a double slit is the same as that of a double slit diffraction pattern, i.e. the FT of two dots enveloped by the FT of a rectangle.
- The fft2() of a square is similar to that of a circle, but the patterns in Fourier space follow the shape of the aperture.
- The fft2() of a gaussian is also a gaussian with a different radius.

1.2 Activity 2. Simulation of an imaging device

In [5]:
```
obj = ImgModel.generate_txtImage(66, 112, 512, "VIP", 256, "white",
                                 "black", True, 128)
aper_arr = np.linspace(0.1, 1, 7)
```

```
fig = mp.figure(figsize=(5*4, 5*2))

ax = fig.add_subplot(241)
ax.imshow(obj, "gray")
ax.axis("off")
ax.set_title("original object")

for i in range(7):
    aper = ImgModel.generate_aperture(rad=aper_arr[i], res=128)
    img = ImgModel.image_aperture(obj, aper)
    ax = fig.add_subplot(2, 4, i+2)
    ax.imshow(img, "gray")
    ax.set_title("aperture radius = %.2f"%(aper_arr[i]))
    ax.axis("off")

mp.tight_layout()
mp.show()
```

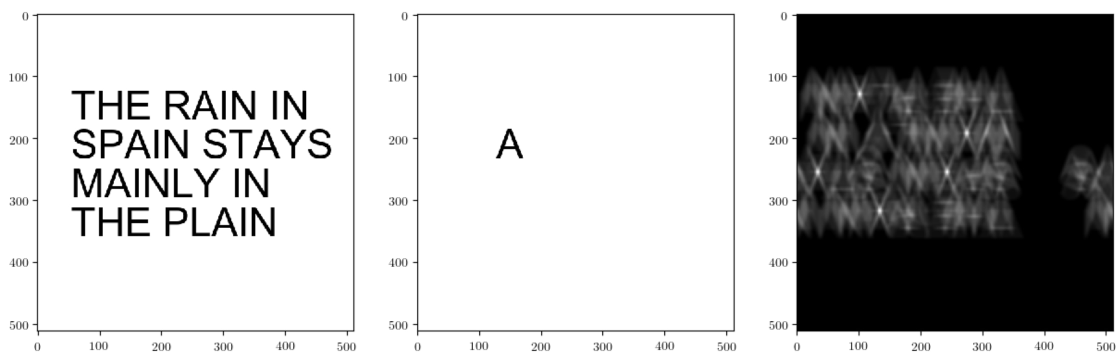


We can observe that a wider aperture images the object more closely to the original. Also notice that the final image is inverted. Recall from optics that a real image that has passed through a lens is always inverted.

1.3 Activity 3. Template matching using correlation

```
In [6]: obj = ImgModel.generate_txtImage(55, 112, 512, "THE RAIN IN\nSPAIN STAYS \
                                          \nMAINLY IN\nTHE PLAIN", 65, "black",
                                          "white", True, 512)
        temp = ImgModel.generate_txtImage(128, 174, 512, "A", 65, "black",
```

```
                                                        "white", True, 512)

fig = mp.figure(figsize=(5*3, 5))

ax = fig.add_subplot(131)
ax.imshow(obj, "gray")

ax = fig.add_subplot(132)
ax.imshow(temp, "gray")

Fobj = fft.fft2(obj)
Ftemp = fft.fft2(temp)
Fmatch = Ftemp*Fobj.conj()
Fimg = fft.fftshift(abs(fft.ifft2(Fmatch)))
ax = fig.add_subplot(133)
ax.imshow(Fimg, "gray")

mp.show()
```

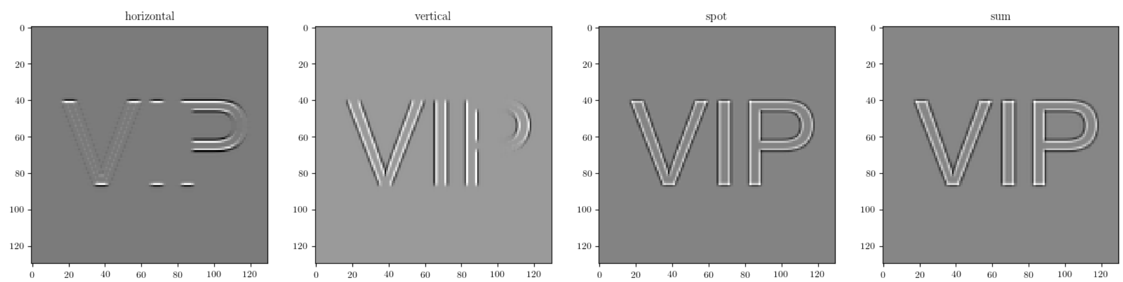


We can observe that peaks arise in locations where there is a letter "A".

1.4 Activity 4. Edge detection using convolution integral

```
In [7]: pat1 = np.array([[-1, -1, -1],
                         [2, 2, 2],
                         [-1, -1, -1]])
        pat2 = pat1.T
        pat3 = np.array([[-1, -1, -1],
                          [-1, 8, -1],
                          [-1, -1, -1]])

        obj = ImgModel.generate_txtImage(66, 112, 512, "VIP", 256, "white",
                                         "black", True, 128)

        fig = mp.figure(figsize=(5*4, 5))
```

```python
img = sig.convolve2d(pat1, obj, "full")
fin = img.copy()
ax = fig.add_subplot(141)
ax.imshow(img, "gray")
ax.set_title("horizontal")

img = sig.convolve2d(pat2, obj, "full")
fin += img
ax = fig.add_subplot(142)
ax.imshow(img, "gray")
ax.set_title("vertical")

img = sig.convolve2d(pat3, obj, "full")
fin += img
ax = fig.add_subplot(143)
ax.imshow(img, "gray")
ax.set_title("spot")

ax = fig.add_subplot(144)
ax.imshow(fin, "gray")
ax.set_title("sum")

mp.show()
```



We can observe that the form of the direction pattern affects the orientation of the edges that are detected. Spot pattern shows the best performance.