```
from PIL import Image
import numpy as np
import math
from scipy import signal
from conda import exceptions


# Part 1.1

# paramenter: n -> dimension of array matrix
# return a numpy array with dimension n
def boxfilter(n):
    if n % 2 == 1:            # check if n is odd
        try:
            a = np.ones((n,n))      # Create a matrix of nxn with all elements
             = 1
            b = np.size(a)
            return a / b            # Divide each element of matrix by size to
             get total sum = 1


        # Throw error if n is odd
        except ValueError:
            print("Assertion Error: Dimensions must be odd")

    else:
         raise exceptions.ArgumentError("Dimensions must be odd")


# Part 1.2

# paramenter: Sigma -> dimension of array matrix
# return a numpy array with dimension n

def gauss1d(sigma):
    if sigma > 0:  # check if sigma is greater than 0
        try:

            array_length = int(math.ceil(float(sigma) * 6))

            # Check if array_length is even
            if (array_length % 2 == 0):
                array_length += 1

            centre = array_length / 2
            # print("Center - ", centre)

            array = np.arange(-centre + 1, centre + 1)

            array = np.floor(array)
            # print("Floor array -  ", array)
```

```
            result = np.exp(-1 * (array ** 2) / (2 * sigma ** 2))

            result = result / np.sum(result)
            # print("Normalized result - ", result)

            return result

        except ValueError:
            print("Sigma is a negative ")


# Part 1.3

# Funtion returns a 2D gaussian filter

def gauss2d(sigma):
    g2d_array = gauss1d(sigma)[np.newaxis]
    #print("G2D array - ", g2d_array)

    g2d_array_transpose = g2d_array.T
    #print("G2D array transpose - ", g2d_array_transpose)

    result = signal.convolve2d(g2d_array, g2d_array_transpose)
    #print("Result - ", result)

    return result


# Part 1.4

def gaussconvolve2d(array, sigma):
    if sigma > 0:
        try:
            gauss2d_filter = gauss2d(sigma)

            # Apply gaussian convolution to a 2D array
            convolve_array = signal.convolve2d(array, gauss2d_filter, 'same')

            # dog_image = Image.open('/dog.jpg')
            # dog_image.show()

            # print("test -", list(np.asarray(dog_image)))

            return convolve_array

        except ValueError:
            print("Sigma is negative")


# Part 5
```

With a seperable 2D Gaussian filter, there are 2m multiplications at each pixel (X,Y). Also there are n x n pixels in (X,Y). Hence there are 2m * n^2 multiplications.
However, the convolution can be sped up by taking a natural log on both sides. At the expense of two ln() and one exp() computations,
multiplication is reduced to admission

```
# Part 2.1

dog_image = Image.open('/Users/gautamsoni/Desktop/CPSC 425/assignment1/dog.jpg')
dog_image.show()


r, g, b = dog_image.split()

b_array = np.asarray(b)
g_array = np.asarray(g)
r_array = np.asarray(r)

b_gauss = gaussconvolve2d(b_array, 7)[:,:,np.newaxis]
g_gauss = gaussconvolve2d(g_array, 7)[:,:,np.newaxis]
r_gauss = gaussconvolve2d(r_array, 7)[:,:,np.newaxis]

new_blurr_dog = np.concatenate((r_gauss, g_gauss, b_gauss), axis=2)

blurr_dog_image = Image.fromarray(new_blurr_dog.astype('uint8'))
blurr_dog_image.show()

# Part 2.2

cat_image = Image.open("/Users/gautamsoni/Desktop/CPSC 425/assignment1/hw1/0a_cat.bmp")
cat_image.show()

cat_image_array = np.asarray(cat_image)

r_cat, g_cat, b_cat = cat_image.split()

b_array_cat = np.asarray(b_cat)
g_array_cat = np.asarray(g_cat)
r_array_cat = np.asarray(r_cat)

b_gauss_cat = gaussconvolve2d(b_array_cat, 7)[:,:,np.newaxis]
g_gauss_cat = gaussconvolve2d(g_array_cat, 7)[:,:,np.newaxis]
r_gauss_cat = gaussconvolve2d(r_array_cat, 7)[:,:,np.newaxis]

new_blurr_cat = np.concatenate((r_gauss_cat, g_gauss_cat, b_gauss_cat), axis=2)
```

```
new_blurr_cat_image = Image.fromarray(new_blurr_cat.astype('uint8'))
new_blurr_cat_image.show()

high_frequency_cat = np.subtract(cat_image_array,new_blurr_cat)

high_frequency_cat_image = Image.fromarray(high_frequency_cat.astype('uint8') +
 128)
high_frequency_cat_image.show()

# Part 2.3

hybrid_image_array = np.add(high_frequency_cat, new_blurr_dog)

hybrid_image = Image.fromarray(hybrid_image_array.astype('uint8'))
hybrid_image.show()
```