hw3.1

November 19, 2019

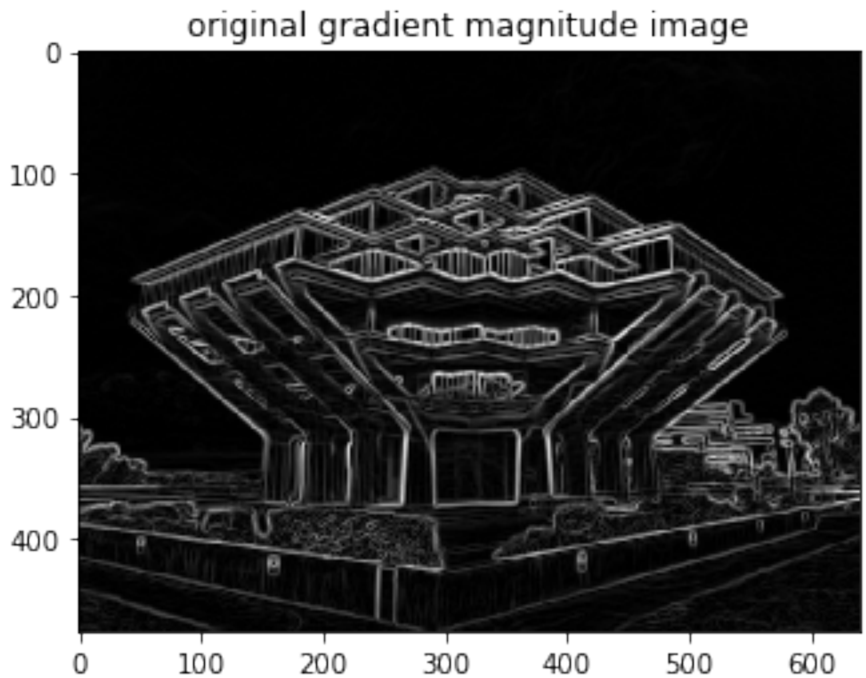
[1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy.ndimage.filters import sobel
```

[2]:
```
img = cv2.imread("geisel.jpg", 0)
img_pad = np.pad(img, (2, 2), mode = 'reflect')
# smoothing image
k = np.array([[2,4,5,4,2],
              [4,9,12,9,4],
              [5,12,15,12,5],
              [4,9,12,9,4],
              [2,4,5,4,2]]) * 1/159
img_smooth = np.zeros(img.shape)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_smooth[i][j] = np.sum(img_pad[i : i + 5, j : j + 5] * k)
```

[3]:
```
# gradient image
kx = np.array([[-1,0,1],
               [-2,0,2],
               [-1,0,1]])
ky = np.array([[-1,-2,-1],
               [0,0,0],
               [1,2,1]])
img_s_pad = np.pad(img_smooth, (1, 1), mode = 'reflect')
img_x = np.zeros(img.shape)
img_y = np.zeros(img.shape)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_x[i][j] = np.sum(img_s_pad[i : i + 3, j : j + 3] * kx)
        img_y[i][j] = np.sum(img_s_pad[i : i + 3, j : j + 3] * ky)
```

[4]:
```
gradient = np.sqrt(np.square(img_x) + np.square(img_y))
gradient *= 255 / gradient.max()
plt.title("original gradient magnitude image")
plt.imshow(gradient, cmap = 'gray')
```

```
plt.savefig("gradient_image.jpg")
plt.show()
```

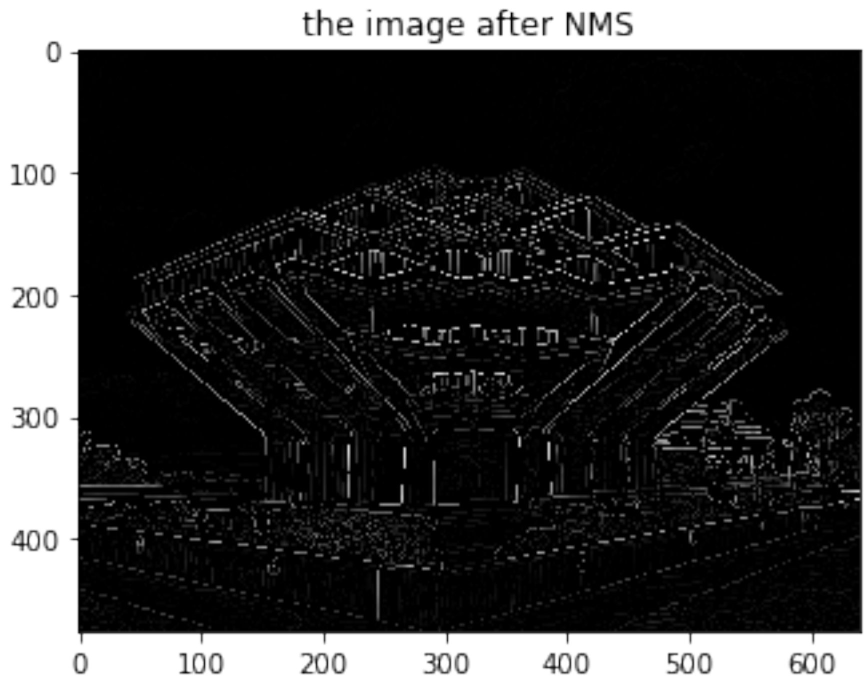


[5]:
```
angle = np.zeros(img.shape)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if (img_x[i][j] == 0):
            angle[i][j] = np.pi / 2
        else:
            angle[i][j] = np.arctan(img_y[i][j] / img_x[i][j])
```

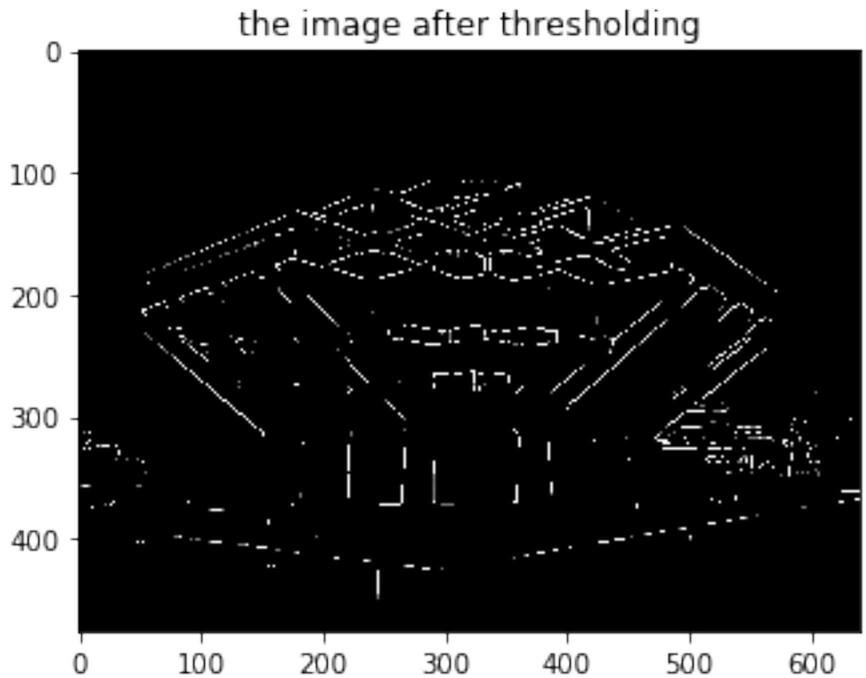
[6]:
```
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M,N))
    angle = D * 180. / np.pi
    angle[angle < 0] += 180


    for i in range(1,M-1):
        for j in range(1,N-1):
            try:
                q = 255
                r = 255
```

```
                #angle 0
                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                    q = img[i, j+1]
                    r = img[i, j-1]
                #angle 45
                elif (22.5 <= angle[i,j] < 67.5):
                    q = img[i+1, j+1]
                    r = img[i-1, j-1]
                #angle 90
                elif (67.5 <= angle[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                #angle 135
                elif (112.5 <= angle[i,j] < 157.5):
                    q = img[i-1, j+1]
                    r = img[i+1, j-1]

                if (img[i,j] >= q) and (img[i,j] >= r):
                    Z[i,j] = img[i,j]
                else:
                    Z[i,j] = 0

            except IndexError as e:
                pass

    return Z
```

[7]:
```
img_nms = non_max_suppression(gradient, angle)
plt.title("the image after NMS")
plt.imshow(img_nms, cmap = 'gray')
plt.savefig("aft_NMS.jpg")
plt.show()
```



[8]:
```
img_threshold = img_nms.copy()
img_threshold[img_threshold >= 130] = 255
img_threshold[img_threshold < 120] = 0
plt.title("the image after thresholding")
plt.imshow(img_threshold, cmap = 'gray')
plt.savefig("aft_threshold.jpg")
plt.show()
```



The original gradient magnitude image, the image after NMS, and the final edge image after thresholding is shown above. Here I choose two threshold. One is 130. The value above 130 will be changed to 255. One is 120. The value below 120 will be changed to 0.

hw3.2

November 19, 2019

[1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import math
```

0.0.1 2(i)

[2]:
```
img = cv2.imread("Car.tif", 0)
# img = int(img)
img_pad = np.zeros((512, 512))
p = int((512 - img.shape[0]) / 2)
q = int((512 - img.shape[1]) / 2)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_pad[i + p][j + q] = img[i][j]
```

[3]:
```
f = np.fft.fft2(img_pad)
fshift = np.fft.fftshift(f)
log_magnitude_spectrum = np.log(np.abs(fshift))
```

[4]:
```
uK = [91, 176, 344, 429]
vK = [168, 166, 166, 169]
uK  = uK - np.ones((4, )) * 256
vK = vK - np.ones((4, )) * 256
print(uK, vK)
x_axis = np.linspace(-256,255,512)
y_axis = np.linspace(-256,255,512)
[v,u] = np.meshgrid(x_axis,y_axis)
```

```
[-165.  -80.   88.  173.] [-88. -90. -90. -87.]
```

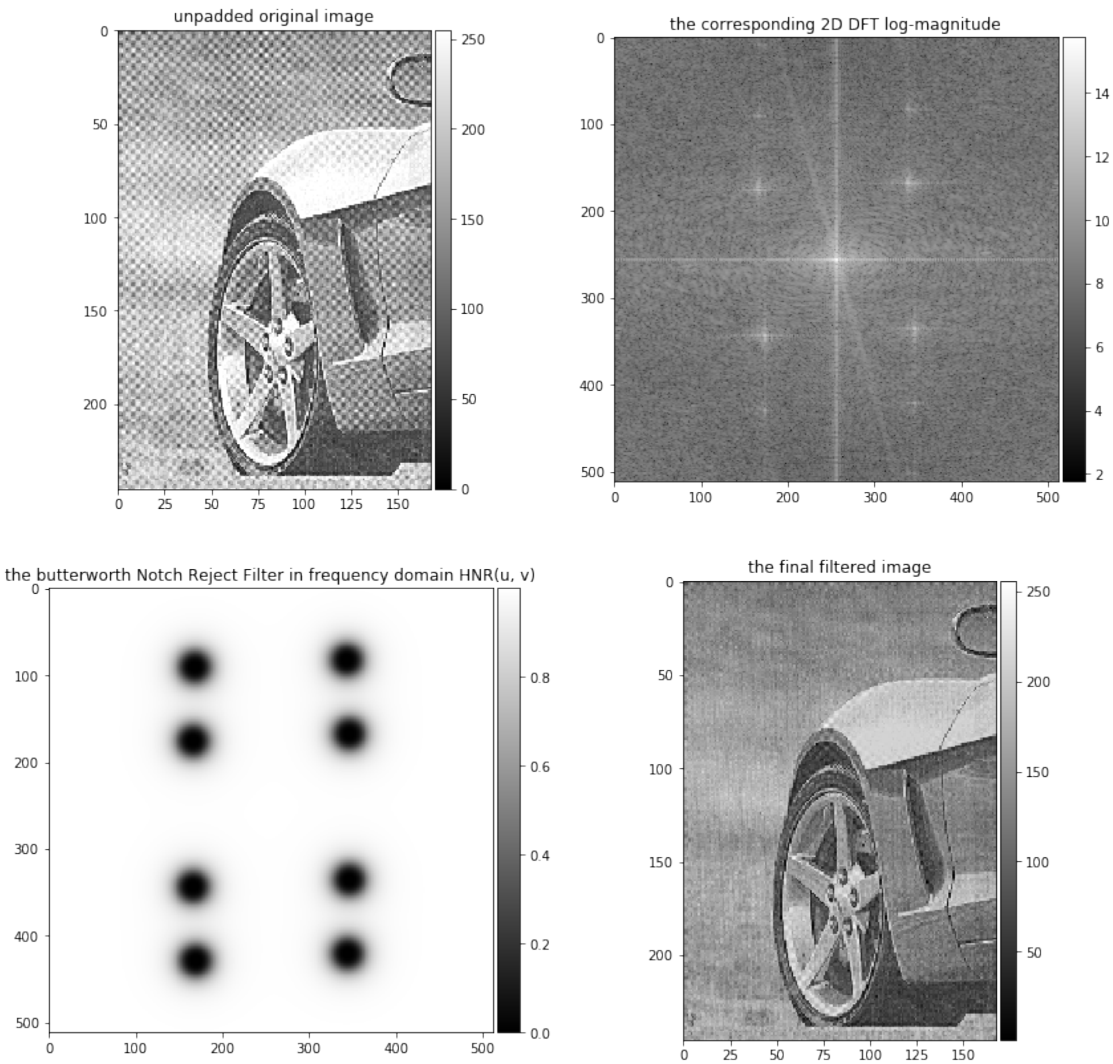
[5]:
```
img_h = np.zeros(img_pad.shape)
n = 2
d0 = 20
for i in range(img_pad.shape[0]):
```

```
    for j in range(img_pad.shape[1]):
        h = 1
        uc = u[i][j]
        vc = v[i][j]
        for k in range(4):
            dk = np.sqrt(np.square(uc - uK[k]) + np.square(vc - vK[k]))
            d_k = np.sqrt(np.square(uc + uK[k]) + np.square(vc + vK[k]))
            if (dk == 0 or d_k == 0):
                h *= 0
            else:
                h *= 1/(1 + math.pow(d0 / dk, 2 * n)) * 1/(1 + math.pow(d0 /␣
↪d_k, 2 * n))
        img_h[i][j] = h
```

[6]:

```
img_no_shift = np.fft.ifftshift(fshift * img_h)
img_back = np.abs(np.fft.ifft2(img_no_shift))
img_back -= img_back.min()
img_back = img_back[p : img_back.shape[0] - p, q : img_back.shape[1] - q] * 256␣
↪/ img_back.max()

f = plt.figure(figsize=(14,14))
f_ax1 = f.add_subplot(221)
f_ax2 = f.add_subplot(222)
f_ax3 = f.add_subplot(223)
f_ax4 = f.add_subplot(224)

img1_1 = f_ax1.imshow(img, cmap = 'gray')
f_ax1.title.set_text("unpadded original image")
divider = make_axes_locatable(f_ax1)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_1, cax, orientation='vertical')

img1_2 = f_ax2.imshow(log_magnitude_spectrum, cmap = 'gray')
f_ax2.title.set_text("the corresponding 2D DFT log-magnitude")
divider = make_axes_locatable(f_ax2)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_2, cax, orientation='vertical')

img2_1 = f_ax3.imshow(img_h, cmap = 'gray')
f_ax3.title.set_text("the butterworth Notch Reject Filter in frequency domain␣
↪HNR(u, v)")
divider = make_axes_locatable(f_ax3)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_1, cax, orientation='vertical')

img2_2 = f_ax4.imshow(img_back, cmap = 'gray')
f_ax4.title.set_text("the final filtered image")
```

```python
divider = make_axes_locatable(f_ax4)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_2, cax, orientation='vertical')
plt.savefig("result_car.jpg")
plt.show()
```



(i) The 10 parameters I choose here are shown as below:

$$n = 2$$

$$D_0 = 20$$

$$u_1 = 91 - 256 = -165, v_1 = 168 - 256 = -88$$

$$u_2 = 176 - 256 = -80, v_2 = 166 - 256 = -90$$

$$u_3 = 344 - 256 = 88, v_3 = 166 - 256 = -90$$

$$u_4 = 429 - 256 = 173, v_4 = 169 - 256 = -87$$

0.0.2 2(ii)

[7]:
```
img = cv2.imread("Street.png", 0)
# img = int(img)
img_pad = np.zeros((512, 512))
p = int((512 - img.shape[0]) / 2)
q = int((512 - img.shape[1]) / 2)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_pad[i + p][j + q] = img[i][j]
```

[8]:
```
f = np.fft.fft2(img_pad)
fshift = np.fft.fftshift(f)
log_magnitude_spectrum = np.log(np.abs(fshift))
```

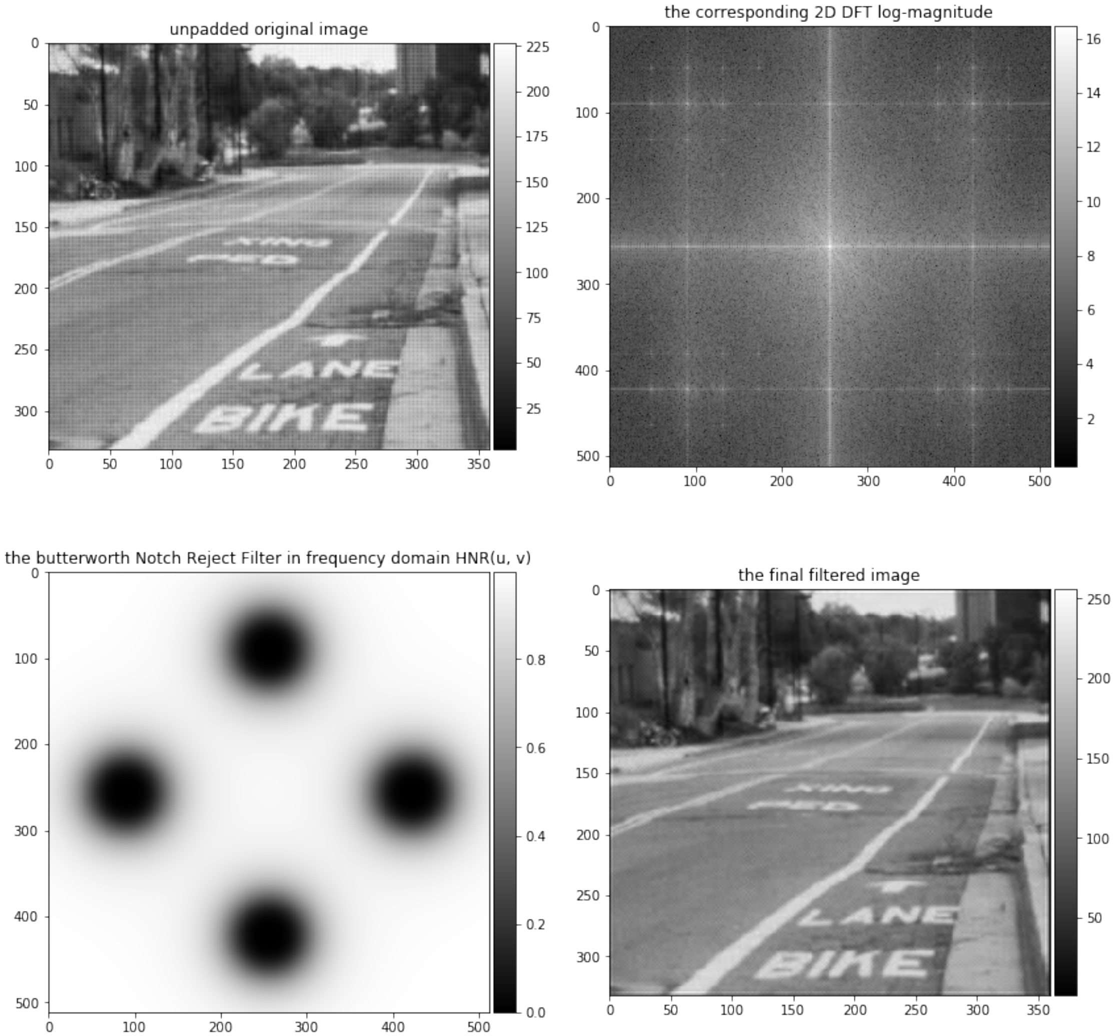
[9]:
```
uK = [256, 90]
vK = [90, 256]
uK  = uK - np.ones((2, )) * 256
vK = vK - np.ones((2, )) * 256
print(uK, vK)
x_axis = np.linspace(-256,255,512)
y_axis = np.linspace(-256,255,512)
[v,u] = np.meshgrid(x_axis,y_axis)
```

```
[   0. -166.] [-166.    0.]
```

[10]:
```
img_h = np.zeros(img_pad.shape)
n = 2
d0 = 50
for i in range(img_pad.shape[0]):
    for j in range(img_pad.shape[1]):
        h = 1
        uc = u[i][j]
        vc = v[i][j]
        for k in range(2):
            dk = np.sqrt(np.square(uc - uK[k]) + np.square(vc - vK[k]))
            d_k = np.sqrt(np.square(uc + uK[k]) + np.square(vc + vK[k]))
            if (dk == 0 or d_k == 0):
                h *= 0
            else:
                h *= 1/(1 + math.pow(d0 / dk, 2 * n)) * 1/(1 + math.pow(d0 /
 ↪d_k, 2 * n))
        img_h[i][j] = h
```

[11]:
```
img_no_shift = np.fft.ifftshift(fshift * img_h)
img_back = np.abs(np.fft.ifft2(img_no_shift))
img_back -= img_back.min()
```

```
img_back = img_back[p : img_back.shape[0] - p, q : img_back.shape[1] - q] * 256␣
↪/ img_back.max()

f = plt.figure(figsize=(14,14))
f_ax1 = f.add_subplot(221)
f_ax2 = f.add_subplot(222)
f_ax3 = f.add_subplot(223)
f_ax4 = f.add_subplot(224)

img1_1 = f_ax1.imshow(img, cmap = 'gray')
f_ax1.title.set_text("unpadded original image")
divider = make_axes_locatable(f_ax1)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_1, cax, orientation='vertical')

img1_2 = f_ax2.imshow(log_magnitude_spectrum, cmap = 'gray')
f_ax2.title.set_text("the corresponding 2D DFT log-magnitude")
divider = make_axes_locatable(f_ax2)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_2, cax, orientation='vertical')

img2_1 = f_ax3.imshow(img_h, cmap = 'gray')
f_ax3.title.set_text("the butterworth Notch Reject Filter in frequency domain␣
↪HNR(u, v)")
divider = make_axes_locatable(f_ax3)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_1, cax, orientation='vertical')

img2_2 = f_ax4.imshow(img_back, cmap = 'gray')
f_ax4.title.set_text("the final filtered image")
divider = make_axes_locatable(f_ax4)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_2, cax, orientation='vertical')
plt.savefig("result_street.jpg")
plt.show()
```



(ii) The 6 parameters I choose here are shown as below:

$$n = 2$$

$$D_0 = 50$$

$$u_1 = 256 - 256 = 0, v_1 = 90 - 256 = -166$$

$$u_2 = 90 - 256 = -166, v_2 = 256 - 256 = 0$$

hw3.3

November 19, 2019

[1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

[2]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

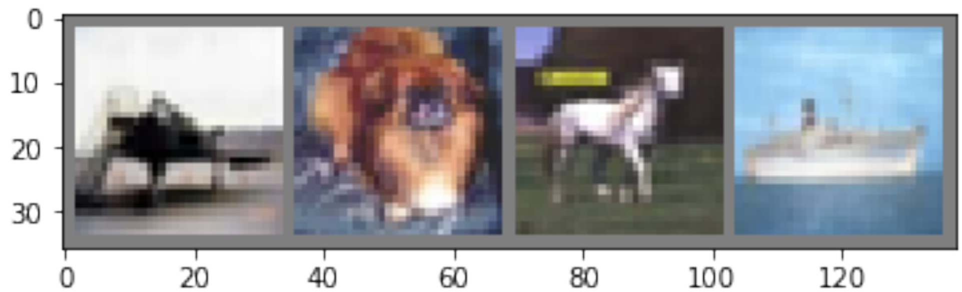
[3]:
```
print(len(trainset))
print(len(trainloader))
```

```
50000
12500
```

(ii) 50000 images and 12500 batches are been used to train the network

[19]:
```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
```

```python
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()


# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
plane   dog horse  ship
```

(iii) Here instead normalize the image, we unnormalize the image by set the range of image from -1 to 1 to 0 to 1, by dividing each pixel value of the image by 2 and add 0.5

[5]:
```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

```
    def forward(self, x):
        x1 = self.conv1(x)
        x = self.pool(F.relu(x1))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x, x1


net = Net()
```

[6]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[7]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)
```

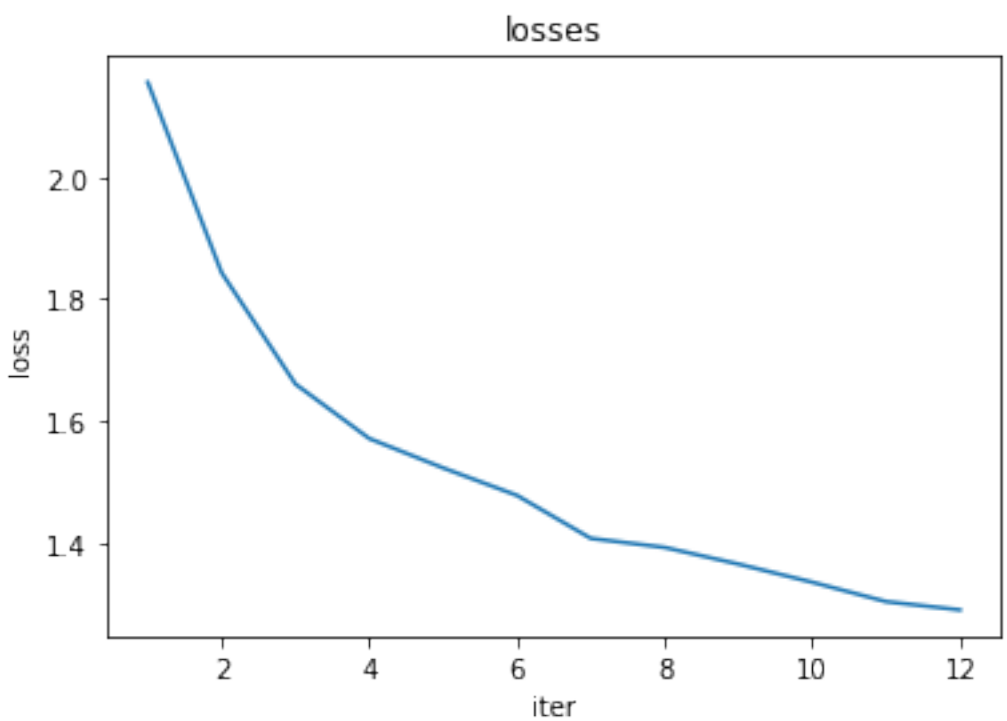
```
cuda:0
```

[8]:
```
losses = []
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)
        net.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs,_ = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
```

```
            losses.append(running_loss / 2000)
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.155
[1,  4000] loss: 1.843
[1,  6000] loss: 1.661
[1,  8000] loss: 1.571
[1, 10000] loss: 1.523
[1, 12000] loss: 1.478
[2,  2000] loss: 1.408
[2,  4000] loss: 1.393
[2,  6000] loss: 1.365
[2,  8000] loss: 1.336
[2, 10000] loss: 1.304
[2, 12000] loss: 1.290
Finished Training
```

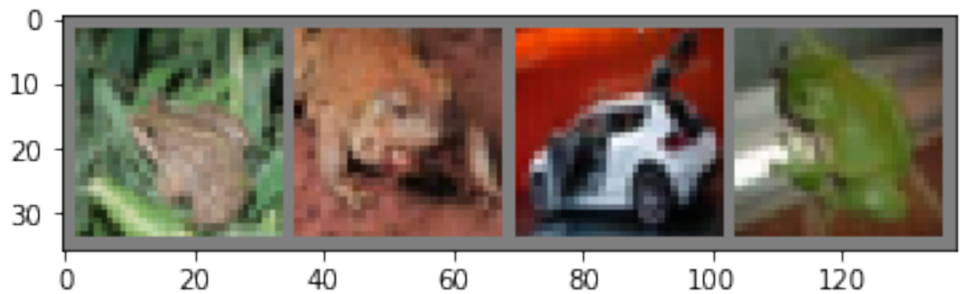
[9]:
```
xrange = np.arange(1, len(losses) + 1, 1)
plt.title('losses')
plt.plot(xrange, losses)
plt.xlabel('iter')
plt.ylabel('loss')
plt.show()
```



(iv) Here I plot the dropping losses

[10]:
```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

[11]:
```
dataiter = iter(testloader)
images, labels = dataiter.next()
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
GroundTruth:   frog  frog   car  frog
```

[12]:
```
net = Net()
net.load_state_dict(torch.load(PATH))
```

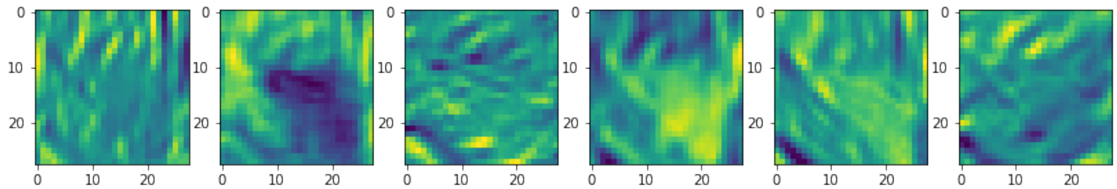
[12]:
```
<All keys matched successfully>
```

[13]:
```
outputs,middle = net(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
middle = middle.detach().numpy()
```

```
Predicted:    cat  frog   car  deer
```

(v) Here we can show the right result from 2nd and 3rd images as the label of ground truth and prediction are the same. We can also see the wrong result of 1st and 4th image as the label of ground truth and prediction are not matched.

[18]:
```
image_one = middle[0]
f = plt.figure(figsize=(14,14))
for i in range(image_one.shape[0]):
    ax = f.add_subplot(1, image_one.shape[0], i + 1)
    ax.imshow(image_one[i])
plt.show()
```



(vi) Here I plot the output of the 1st layer of CNN using one image from the training set as above

[16]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        net.to(device)
        outputs,_ = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 54 %
```

[17]:
```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        net.to(device)
        outputs,_ = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
```

```python
for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 63 %
Accuracy of   car : 79 %
Accuracy of  bird : 25 %
Accuracy of   cat : 30 %
Accuracy of  deer : 47 %
Accuracy of   dog : 36 %
Accuracy of  frog : 72 %
Accuracy of horse : 71 %
Accuracy of  ship : 54 %
Accuracy of truck : 61 %
```