Lab 3: CNNs and Deep Learning

(version 1.0)

TBMI26/732A55 Neural Networks and Learning Systems 2020

Michael Felsberg

Hint: Use the provided test cases to check if your solutions are valid.

2D Convolution

It is widely used with 2D signals such as images. For the further steps, we often need to visualize an image and we define a shortcut for that:

In [1]:
```
from matplotlib import pyplot as plt

def visualize(img, title=''):
    plt.imshow(img,'gray')
    plt.colorbar()
    plt.title(title)
    plt.show()
    print('Image size:', img.shape)
```

In [ ]:
```

```

Task 1: Convolution can be performed in 2D using the function `scipy.signal.convolove2d()` . Use this function to generate a 2D kernel of size 33×33 by five times cascading 2D convolutions of H with itself, starting with

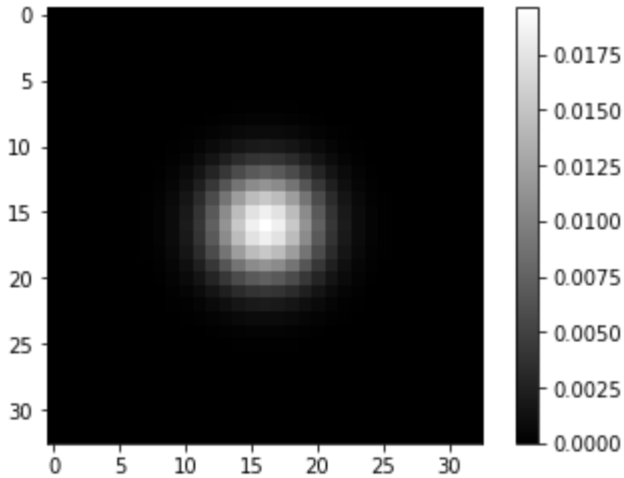
$$H = \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} /4.$$

Visualize the kernel using `visualize` function defined above.

In [6]:
```
# YOUR CODE HERE
import scipy
import numpy as np
from scipy import signal
from scipy.signal import convolve2d

init_H = np.array([[1,1], [1,1]])/4
H=signal.convolve2d(init_H,init_H)
for i in range(1,5):
    H = signal.convolve2d(H,H)

visualize(H)
```



```
Image size: (33, 33)
```

In [7]:
```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(H.sum().round()==1)
```

Task 2: Now, load the image `'MR15^044.JPG'` (a sample from ImageNet), **sum** its RGB-channels, **normalize** it to the range [0,1], and convolve it with H from task 3 under the options `'valid'` and `'same'`.

What differences do you observe regarding the size of the output?

In [9]:
```
img = plt.imread('MR15^044.JPG')
```

Out[9]:
```
array([[[ 96, 128, 151],
        [ 96, 128, 151],
        [ 98, 130, 153],
        ...,
        [  6,  38,  49],
        [  4,  36,  49],
        [ 17,  49,  62]],

       [[ 96, 128, 151],
        [ 97, 129, 152],
        [ 98, 130, 153],
        ...,
```

```
        [  8,  38,  48],
        [ 18,  48,  59],
        [ 31,  61,  72]],

       [[ 97, 129, 152],
        [ 98, 130, 153],
        [ 99, 131, 154],
        ...,
        [  2,  31,  39],
        [ 12,  39,  50],
        [  8,  35,  46]],

       ...,

       [[ 73, 105, 143],
        [ 73, 108, 140],
        [ 83, 119, 141],
        ...,
        [193,  79,  27],
        [182,  66,  25],
        [137,  23,   0]],

       [[ 74, 108, 146],
        [ 64,  99, 131],
        [ 71, 107, 129],
        ...,
        [237, 124,  48],
        [245, 131,  60],
        [217, 103,  33]],

       [[127, 161, 199],
        [ 87, 122, 154],
        [ 74, 112, 135],
        ...,
        [217, 102,  13],
        [232, 118,  32],
        [220, 106,  20]]], dtype=uint8)
```

In [5]:

```
# YOUR CODE HERE
img = plt.imread('MR15^044.JPG')



visualize(img_gray, 'The normalized grayscale input image')
visualize(omg_sc_valid, 'The convolved image in "valid" mode')
visualize(omg_sc_same, 'The convolved image in "same" mode')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-b164f5a2899c> in <module>
      3
      4
----> 5 visualize(img_gray, 'The normalized grayscale input image')
      6 visualize(omg_sc_valid, 'The convolved image in "valid" mode')
      7 visualize(omg_sc_same, 'The convolved image in "same" mode')

NameError: name 'img_gray' is not defined
```

In [141…

```
img = plt.imread('MR15^044.JPG')
R = img.sum(axis=0)
R.shape
```

Out[141…

```
(248, 3)
```

In [ ]:

```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(img_gray.max() == 1.0)
assert(omg_sc_valid.shape == (214, 216))
assert(omg_sc_same.shape == (246, 248))
```

Strided convolution

In strided convolution, samples are removed based on the stride. According to the Nyquist theorem, this can generate aliasing artifacts.

Task 3: Visualize the input image and the second output image from task 4, `omg_sc_same` , while only keping every *fifth* row and column.

Hint: Use Python extended slicing, read this guid on extended slices

In [ ]:

```
# YOUR CODE HERE
raise NotImplementedError()

visualize(img_gray_ds, 'Input image with stride of 5')
visualize(omg_sc_same_ds, 'Filtered input image with stride of 5')
```

What do you observe, in particular at the ski?

YOUR ANSWER HERE

Convolution in PyTorch

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing. It is primarily developed by Facebook's AI Research lab.

We will start by utilizing PyTorch to perform convolution operations in 2D.

Task 4: Apply the cascaded 33×33 filter from task 1 to the image using a `torch.nn.Conv2d` layer.

Compare the results from *Scipy* in task 2 and *PyTorch* in this task by subtracting the output images.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()

# Hint: Convert the output tensor to numpy array
visualize(out_2d_np, 'Filtered image using PyTorch')
diff = np.abs(out_2d_np-omg_sc_same)
visualize(diff, 'Diff. between Scipy and PyTorch')
```

In [ ]:
```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(diff.mean()<1e-7)
```

Task 5: Repeat the previous task with stride 5. Compare with `omg_sc_same_ds` from task 3.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()

# Hint: Convert the output tensor to numpy array
visualize(out_2d_s5_np, 'Filtered image using PyTorch with stride=5')
diff_s5 = np.abs(out_2d_s5_np-omg_sc_same_ds)
visualize(diff_s5, 'Diff. between Scipy and PyTorch')
```

In [ ]:
```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(diff_s5.mean()<1e-6)
```

Task 6: Repeat task 5 with stride of 5 and a 1×1 filter. Compare with `img_gray_ds` from task 3.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()

# Hint: Convert the output tensor to numpy array
visualize(out_2d_s5_1_np, 'Filtered image using PyTorch with stride=5')
diff_s5_1 = np.abs(out_2d_s5_1_np-img_gray_ds)
visualize(diff_s5_1, 'Diff. between Scipy and PyTorch')
```

In [ ]:
```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(diff_s5_1.mean()<1e-7)
```

Training a PyTorch Convolution layer

Now, we want the network to learn the convolution filter given the input and the filtered output.

Task 7: Considering the input image tensor `inp_2d` from task 4 as a *batch* and the filtered output `out_2d_t` as a label, use `torch.optim.SGD` to learn the the filter H.

Hints:

- Use the L1 loss from `torch.nn.functional.l1_loss` .
- Use a small learning rate.
- Detach `out_2d_t` from the model graph in task 4 to avoid errors.
- Iterate for 500 iterations.
- Clip the weights after each iteration to $[0, \infty[$ for stable convergence.
- Print the loss every 100 steps.

In [ ]:

```
# Define the model and the optimizer
# YOUR CODE HERE
raise NotImplementedError()

# Visualize the trained filter
visualize(model_1layer.weight[0,0,:,:].detach().cpu().numpy())
```

Task 8: To make the transitiong to the next task easier, redo task 7 by defining a custom PyTorch module which includes only 1 convolution layer.
You can follow this tutorial.
A custom module class inherits `torch.nn.Module` class and needs to have two mandatory functions:

- `__init__(self):` where you define layers included in your module.
- `forward(self, x):` where you define the inference steps of your network.

The built-in auto-differentiation module in PyTorch will keep track of the operations that you perform in the inference steps and calculates their derivatives when you back-propagate the loss function during training.

In [ ]:

```
# YOUR CODE HERE
raise NotImplementedError()

# Visualize the trained filter
visualize(net.conv1.weight[0,0,:,:].detach().cpu().numpy())
```

Training a whole network

So far, we have experimented with training a single convolution layer. Now we try to train a whole network to perform the task of image classification on CIFAR-10 dataset.
But first, make sure that CUDA is available by running the following command:

In [ ]:

```
import torch
print("CUDA Available: ",torch.cuda.is_available())
```

Task 9: We will train on CIFAR10, which is readily available at `torchvision.datasets.CIFAR10` .

Create a dataloader for the *training* and the *test* sets of CIFAR10 using `torch.utils.data.DataLoader` , then show some examples from the training set using `torchvision.utils.make_grid` and print out their labels.

Hints :

- The `imshow` function for visualizing the images is provided below.
- Use `torchvision.transforms` to perform whitening on images (normalization using the mean and the standard deviation).
- Use a batch size of 64.

In [ ]:

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

def imshow(img):
    img = img * 0.2 + 0.5  # Un-Normalize, Change according to your normaliz
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
    return npimg.mean()

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 's

# YOUR CODE HERE
raise NotImplementedError()

# Show some random images
dataiter = iter(trainloader)
images, labels = dataiter.next()
grid_img = torchvision.utils.make_grid(images)
imshow(grid_img)

# Print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```

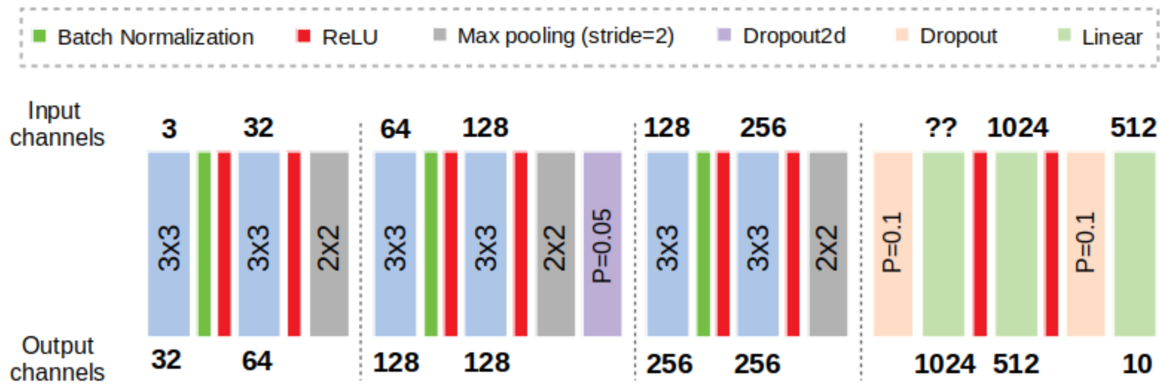
In [ ]:

```
### TEST CELL. PLEASE DON'T CHANGE ###
assert(grid_img.std()>0.8)
```

Baseline Model

Task 10: Build the depicted LeNet5-inspired model using PyTorch standard components. Assume a **padding** with `same` mode for all convolution layers.

Try to figure out the missing dimension at the first fully connected layer.



In [ ]:

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

class LeNet5(nn.Module):
    def __init__(self):
        super().__init__()

        # Define the network
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, x):
        # Perform Inference
        # YOUR CODE HERE
        raise NotImplementedError()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = LeNet5().to(device)
```

Task 11: Train the LeNet5 model for 40 epochs using a suitable batch size and display the result.

Hints:

- Define an optimizer, e.g. SGD optimizer.
- Define a suitable loss function.
- Iterate for 40 epochs and at each epoch calculate a running loss and accuracy on the training set.
- After each epoch, evaluate the model on the test set. You can achieve this by **completing** the `test` function below that performs *only* inference on the test set and calculates the accuracy.

In [ ]:

```
# A function to plot the accuracy training history
def plot_model_history(history):
    plt.figure(0)
    plt.plot(history['train'],'r', lw=3)
    plt.plot(history['test'],'b', lw=3)
    plt.rcParams['figure.figsize'] = (8, 6)
    plt.xlabel("Epoch number")
    plt.ylabel("Accuracy")
    plt.title("Training Accuracy vs Test Accuracy")
    plt.legend(['Training','Test'])
    plt.grid(True)

# Test function that runs only inference
def test(model, testloader):
    correct = 0
    total = 0
    # YOUR CODE HERE
    raise NotImplementedError()
    print('Test Accuracy: %d %%' % (100 * correct / total))
    return correct / total
```

In [ ]:

```
NUM_EPOCHS = 40
LR = 0.01

# Define a proper optimizer and a proper loss function
# YOUR CODE HERE
raise NotImplementedError()

acc_history = {'train':[], 'test':[]}

# Iterate for N epochs
# YOUR CODE HERE
raise NotImplementedError()

print('Finished Training!')

plot_model_history(acc_history)

# Let's quickly save our trained model:
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

== MANDATORY QUESTIONS END HERE ==

Baseline + Decaying Learning Rate

In most papers, the learning rate is successively reduced in order to boost the final performance, e.g. divided by two after 20 and 30 epochs.

[EXTRA] Task 12: Define a suitable function and train the previous model with decaying learning rate. Plot the result and compare it to the baseline.

In [ ]:
```
def adjust_learning_rate(optimizer, epoch):
    for param_group in optimizer.param_groups:
        lrate = param_group["lr"]
        # YOUR CODE HERE
        raise NotImplementedError()
```

In [ ]:
```
net_lr = LeNet5().to(device)

# Define a proper optimizer and a proper loss function
# YOUR CODE HERE
raise NotImplementedError()

acc_history_lr = {'train':[], 'test':[]}
# Iterate for N epochs
# YOUR CODE HERE
raise NotImplementedError()
print('Finished Training!')

plot_model_history(acc_history_lr)

# Let's quickly save our trained model:
PATH = './cifar_net_lr.pth'
torch.save(net_lr.state_dict(), PATH)
```

Baseline + Decaying Learning rate + Data Augmentation

[EXTRA] Task 13: Data augmentation is known to reduce overfitting. Use `torchvision.transforms` to perform additional augmentation with flipping and random cropping. Adjust the number of epochs and the learning rate schedule if needed. What do you observe?

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:

```
net_lr_wr_aug = LeNet5().to(device)

# Define a proper optimizer and a proper loss function
# YOUR CODE HERE
raise NotImplementedError()

acc_history_lr_wr_aug = {'train':[], 'test':[]}
# Iterate for N epochs
# YOUR CODE HERE
raise NotImplementedError()
print('Finished Training!')

plot_model_history(acc_history_lr_wr_aug)

# Let's quickly save our trained model:
PATH = './cifar_net_lr_wr_aug.pth'
torch.save(net_lr_wr_aug.state_dict(), PATH)
```