Open in Colab (https://colab.research.google.com/github/lucetre/MachineLearning/blob/master/Assignment3/AS3-CNN.ipynb)

M2608.001300 Machine Learning
Assignment #3 Training Convolutional Neural Networks (Pytorch)

For understanding of this work, please carefully look at given PPT file.

Now, you're going to leave behind your implementations and instead migrate to one of popular deep learning frameworks, **PyTorch**.
In this notebook, you will learn how to train convolutional neural networks (CNNs) for classifying images in the CIFAR-10 dataset.
There are **2 sections**, and in each section, you need to follow the instructions to complete the skeleton codes and explain them.

Note: certain details are missing or ambiguous on purpose, in order to test your knowledge on the related materials. However, if you really feel that something essential is missing and cannot proceed to the next step, then contact the teaching staff with clear description of your problem.

Some helpful tutorials and references for assignment #3:

- [1] Pytorch official documentation. [link] (https://pytorch.org/docs/stable/index.html)
- [2] Stanford CS231n lectures. [link] (http://cs231n.stanford.edu/)
- [3] Szegedy et al., "Going deeper with convolutions", CVPR 2015. [pdf] (http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf)

1. Load datasets

The CIFAR-10 dataset will be downloaded automatically if it is not located in the *data* directory.

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

In [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=transfo
rm)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=8,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transfor
m)
testloader = torch.utils.data.DataLoader(testset, batch_size=8,
                                         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
```

In [3]:

```
# function to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    print(np.transpose(npimg, (1, 2, 0)).shape)
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

In [4]:

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

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%s' % classes[labels[j]] for j in range(8)))
# print size of single image
print(images[1].shape)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f3bf14e6b90>
(36, 274, 3)
```


```
plane frog bird truck deer frog cat dog
torch.Size([3, 32, 32])
```

2. Training a small CNN model

CNN architecture in order:

- 7x7 Convolutional layer with 8 filters, strides of 1, and ReLU activation
- 2x2 Max pooling layer with strides of 2
- 4x4 Convolutional layer with 16 filters, strides of 1, and ReLU activation
- 2x2 Max pooling layer with strides of 2
- Fully connected layer with 100 output units and ReLU activation
- Fully connected layer with 80 output units and ReLU activation
- Fully connected layer with 10 output units and linear activation
- You can use any padding option.

Training setup:

- Loss function: Softmax cross entropy
- Optimizer: Gradient descent with 0.001 learning rate
- Batch size: 8
- Training epoch: 2

In [20]:
```
# Define a CNN model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        ##############################################################
##############
        #                          IMPLEMENT YOUR CODE
#
        ##############################################################
##############
        self.conv1 = nn.Conv2d(3, 8, 7)
        self.conv2 = nn.Conv2d(8, 16, 4)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 5 * 5, 100)
        self.fc2 = nn.Linear(100, 80)
        self.fc3 = nn.Linear(80, 10)
        ##############################################################
##############
        #                          END OF YOUR CODE
#
        ##############################################################
##############

    def forward(self, x):
        ##############################################################
##############
        #                          IMPLEMENT YOUR CODE
#
        ##############################################################
##############
        x = self.pool(F.relu(self.conv1(x)))
        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)
        ##############################################################
##############
        #                          END OF YOUR CODE
#
        ##############################################################
##############
        return x

net = Net()
```

In [21]:
```
# Training on GPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# device = torch.device("cpu")
net = net.to(device)
print(device)
```

```
cuda:0
```

In [22]:
```
# Define a Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [23]:
```
# Function to train the network

def train(net, trainloader, max_epoch, crit, opt, model_path='./cifar_ne
t.pth'):

    for epoch in range(max_epoch):  # loop over the dataset multiple tim
es

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

            # Training on GPU
            inputs = inputs.to(device)
            labels = labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = crit(outputs, labels)
            loss.backward()
            opt.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))
                running_loss = 0.0

    print('Finished Training')
    torch.save(net.state_dict(), model_path)
    print('Saved Trained Model')
```

In [24]:
```
PATH = './cifar_net.pth'
epoch = 2
train(net, trainloader, epoch, criterion, optimizer, PATH)
```

```
[1,  2000] loss: 2.149
[1,  4000] loss: 1.750
[1,  6000] loss: 1.585
[2,  2000] loss: 1.494
[2,  4000] loss: 1.429
[2,  6000] loss: 1.378
Finished Training
Saved Trained Model
```

In [25]:
```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%s' % classes[labels[j]] for j in range
(4)))
net = Net()
net.load_state_dict(torch.load(PATH))
outputs = net(images)
_, predicted = torch.max(outputs, 1)

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

```
(36, 274, 3)
```


```
GroundTruth:  cat ship ship plane
Predicted:  cat ship ship plane
```

In [26]:
```
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
        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


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 : 64 %
Accuracy of  bird : 39 %
Accuracy of   cat : 43 %
Accuracy of  deer : 30 %
Accuracy of   dog : 38 %
Accuracy of  frog : 71 %
Accuracy of horse : 65 %
Accuracy of  ship : 56 %
Accuracy of truck : 47 %
```

In [30]:
```
# function to calculate accuracy
def print_accuracy(net, dataloader):
    correct = 0
    total = 0

    with torch.no_grad():
        for data in dataloader:
            images, labels = data
            # Inference on GPU
            images = images.to(device)
            labels = labels.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 %d test images: %d %%' % (tota
l,
        100 * correct / total))
```

In [31]:
```
# load trained model then test
net.load_state_dict(torch.load(PATH))
print_accuracy(net, testloader)
```

```
Accuracy of the network on the 10000 test images: 51 %
```

3. Design a better model on CIFAR-10

Now it's your job to experiment with CNNs to train a model that achieves **>= 70% accuracy on the test set** of CIFAR-10. You can use the implemented *inception class* below.

Things you can try to change:

- Batch size (input parameter of dataloader)
- Filter size
- Number of filters
- Pooling vs Strided Convolution
- Network architectures
- Optimizers
- Activation functions
- Regularizations
- Model ensembles
- Data augmentation

In [11]:

```python
class Inception(nn.Module):
    def __init__(self, in_planes, n1x1, n3x3red, n3x3, n5x5red, n5x5, po
ol_planes):
        super(Inception, self).__init__()
        # 1x1 conv branch
        self.b1 = nn.Sequential(
            nn.Conv2d(in_planes, n1x1, kernel_size=1),
            nn.BatchNorm2d(n1x1),
            nn.ReLU(True),
        )

        # 1x1 conv -> 3x3 conv branch
        self.b2 = nn.Sequential(
            nn.Conv2d(in_planes, n3x3red, kernel_size=1),
            nn.BatchNorm2d(n3x3red),
            nn.ReLU(True),
            nn.Conv2d(n3x3red, n3x3, kernel_size=3, padding=1),
            nn.BatchNorm2d(n3x3),
            nn.ReLU(True),
        )

        # 1x1 conv -> 5x5 conv branch
        self.b3 = nn.Sequential(
            nn.Conv2d(in_planes, n5x5red, kernel_size=1),
            nn.BatchNorm2d(n5x5red),
            nn.ReLU(True),
            nn.Conv2d(n5x5red, n5x5, kernel_size=5, padding=2),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
            nn.Conv2d(n5x5, n5x5, kernel_size=5, padding=2),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
        )

        # 3x3 pool -> 1x1 conv branch
        self.b4 = nn.Sequential(
            nn.MaxPool2d(3, stride=1, padding=1),
            nn.Conv2d(in_planes, pool_planes, kernel_size=1),
            nn.BatchNorm2d(pool_planes),
            nn.ReLU(True),
        )

    def forward(self, x):
        y1 = self.b1(x)
        y2 = self.b2(x)
        y3 = self.b3(x)
        y4 = self.b4(x)
        return torch.cat([y1,y2,y3,y4], 1)
```

In [15]:

```
# Define a CNN model
class GoogLeNet(nn.Module):
    def __init__(self):
        super(GoogLeNet, self).__init__()
        ################################################################
##############
        #                           IMPLEMENT YOUR CODE
#
        ################################################################
##############

        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 192, kernel_size=3, padding=1),
            nn.BatchNorm2d(192),
            nn.ReLU(True),
        )

        self.a3 = Inception(192,  64,  96, 128, 16, 32, 32)
        self.b3 = Inception(256, 128, 128, 192, 32, 96, 64)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(480, 192,  96, 208, 16,  48,  64)
        self.b4 = Inception(512, 160, 112, 224, 24,  64,  64)
        self.c4 = Inception(512, 128, 128, 256, 24,  64,  64)
        self.d4 = Inception(512, 112, 144, 288, 32,  64,  64)
        self.e4 = Inception(528, 256, 160, 320, 32, 128, 128)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a5 = Inception(832, 256, 160, 320, 32, 128, 128)
        self.b5 = Inception(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(8, stride=1)
        self.linear = nn.Linear(1024, 10)

        ################################################################
##############
        #                           END OF YOUR CODE
#
        ################################################################
##############

    def forward(self, x):
        ################################################################
##############
        #                           IMPLEMENT YOUR CODE
#
        ################################################################
##############

        out = self.pre_layers(x)
        out = self.a3(out)
        out = self.b3(out)
        out = self.maxpool(out)
        out = self.a4(out)
        out = self.b4(out)
        out = self.c4(out)
        out = self.d4(out)
        out = self.e4(out)
        out = self.maxpool(out)
        out = self.a5(out)
        out = self.b5(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

        ################################################################
```

In [16]:
```
import time
googlenet = GoogLeNet()
googlenet = googlenet.to(device)

# Define a Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(googlenet.parameters(), lr=0.001, momentum=0.9)

start_time = time.time()
PATH = './google_net.pth'

# Train
train(googlenet, trainloader, 2, criterion, optimizer, PATH)
print("Elapsed time: {}s".format(time.time() - start_time))
```

```
[1,  2000] loss: 1.579
[1,  4000] loss: 1.138
[1,  6000] loss: 0.938
[2,  2000] loss: 0.757
[2,  4000] loss: 0.706
[2,  6000] loss: 0.662
Finished Training
Saved Trained Model
Elapsed time: 484.98631286621094s
```

In [22]:
```
# Test
googlenet.load_state_dict(torch.load(PATH))
print_accuracy(googlenet, testloader)
```

```
Accuracy of the network on the 10000 test images: 78 %
```

In [9]:
```
# Define a CNN model
class BetterNet(nn.Module):
    def __init__(self):
        super(BetterNet, self).__init__()
        ################################################################
##############
        #                          IMPLEMENT YOUR CODE
#
        ################################################################
##############

        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
        )

        self.a3 = Inception(128,  32,  48,  64,  8, 16, 16)
        self.b3 = Inception(128,  64,  96, 128, 16, 32, 32)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(256, 160,  96, 256, 16,  64,  64)
        self.b4 = Inception(544, 256, 128, 256, 64, 128, 128)
        self.c4 = Inception(768, 256, 128, 256, 64, 128, 128)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a5 = Inception(768, 256, 256, 512, 64, 128, 128)
        self.b5 = Inception(1024, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(8, stride=1)
        self.linear = nn.Linear(1024, 10)

        ################################################################
##############
        #                          END OF YOUR CODE
#
        ################################################################
##############

    def forward(self, x):
        ################################################################
##############
        #                          IMPLEMENT YOUR CODE
#
        ################################################################
##############

        out = self.pre_layers(x)
        out = self.a3(out)
        out = self.b3(out)
        out = self.maxpool(out)
        out = self.a4(out)
        out = self.b4(out)
        out = self.c4(out)
        out = self.maxpool(out)
        out = self.a5(out)
        out = self.b5(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

        ################################################################
##############
        #                          END OF YOUR CODE
#
        ################################################################
```

In [13]:
```
import time
betternet = BetterNet()
betternet = betternet.to(device)

# Define a Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(betternet.parameters(), lr=0.001, momentum=0.9)

start_time = time.time()
PATH = './better_net.pth'

# Train
train(betternet, trainloader, 2, criterion, optimizer, PATH)
print("Elapsed time: {}s".format(time.time() - start_time))
```

```
[1,  2000] loss: 1.533
[1,  4000] loss: 1.098
[1,  6000] loss: 0.905
[2,  2000] loss: 0.747
[2,  4000] loss: 0.674
[2,  6000] loss: 0.652
Finished Training
Saved Trained Model
Elapsed time: 357.80253648757935s
```

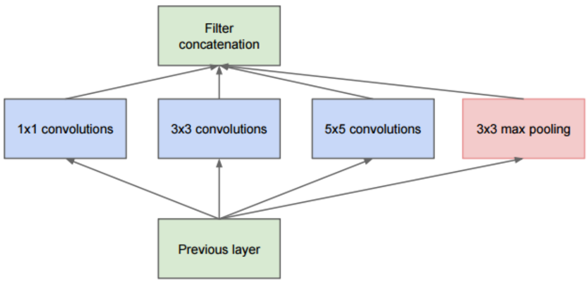
In [16]:
```
# Test
betternet.load_state_dict(torch.load(PATH))
print_accuracy(betternet, testloader)
```

```
Accuracy of the network on the 10000 test images: 79 %
```

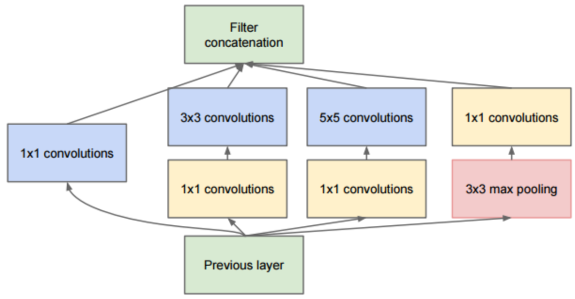
Describe what you did here

In this cell you should also write an explanation of what you did, any additional features that you implemented, and any visualizations or graphs that you make in the process of training and evaluating your network.

Inception module



(a) Inception module, naïve version



(b) Inception module with dimension reductions

GoogLeNet structure

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

BetterNet structure

- Based on GoogLeNet architecture, I've rearranged the order of inceptions and resized convolutional layer sizes. Since GoogLeNet is specialized with 336 * 336 px sized image classification, I thought the number of operations to learn parameters is too expensive. To minimize computation time duration while keeping the accuracy higher than 70%, I used the following inception layers.
    1. Conv2d
    2. BatchNorm2d
    3. ReLU
    4. Inception
        - n1x1(48), n3x3(8), n5x5(16), pool(16)
    5. Inception
        - n1x1(96), n3x3(16), n5x5(32), pool(32)
    6. MaxPool2d
    7. Inception
        - n1x1(160), n3x3(256), n5x5(64), pool(64)
    8. Inception
        - n1x1(256), n3x3(256), n5x5(128), pool(128)
    9. Inception
        - n1x1(256), n3x3(256), n5x5(128), pool(128)
    10. MaxPool2d
    11. Inception
        - n1x1(256), n3x3(512), n5x5(128), pool(128)
    12. Inception
        - n1x1(384), n3x3(384), n5x5(128), pool(128)
    13. AvgPool2d
    14. Linear
- Result (GTX 1080 Ti)

Neural Network	Time	Accuracy
GoogLeNet	484.98s	78%
BetterNet	357.80s	79%