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

In [2]:
```
# Device configuration
device = torch.device('cuda:0' if torch.cuda.is_available() else 'c
pu')

# Hyper parameters
num_epochs = 6
num_classes = 10
batch_size = 100
learning_rate = 0.003

# Input pipeline
data_transform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),#numpy to tensorに変える
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229
, 0.224, 0.225])
    ])
# train data読み込み
train_dataset = torchvision.datasets.ImageFolder(root='animal/train
', #修正
                                           transform=data_transform
)
test_dataset = torchvision.datasets.ImageFolder(root='animal/test',
#修正
                                           transform=data_transform
)
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

In [4]:

```
# Convolutional neural network (two convolutional layers)

# Pretrained model
import torchvision.models as models

#for idx, m in enumerate(models.resnet18().modules()):
        #print(idx, '->', m)

class Resnet(nn.Module):

    def __init__(self, num_classes=10):
        super(Resnet,self).__init__()
        resnet = models.resnet18(pretrained=True) #
        resnet.fc = nn.Linear(512, 10)            #
        self.resnet = resnet

    def forward(self,x):
        x = self.resnet(x)
        return x

model = Resnet(num_classes).to(device)


# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

In [5]:

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        print(i)

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, lo
ss.item()))
```

```
0
1
2
```

```
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
```

In [6]:

```
# Test the model
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        print(total)

    print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))

# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```

```
100
Test Accuracy of the model on the 10000 test images: 42.0 %
```

再学習

In [7]:

```
# Hyper parameters 再設定
num_epochs = 7
num_classes = 10
batch_size = 100
learning_rate = 0.002
```

In [8]:

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        print(i)

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, lo
ss.item()))
```

```
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
```

```
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
```

In [9]:

```
# Test the model
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        print(total)

    print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))

# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```

```
100
Test Accuracy of the model on the 10000 test images: 10.0 %
```

In [1]:

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

In [4]:

```
# Device configuration
device = torch.device('cuda:0' if torch.cuda.is_available() else 'c
pu')

# Hyper parameters
num_epochs = 8
num_classes = 10
batch_size = 100
learning_rate = 0.002

# Input pipeline
data_transform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),#numpy to tensorに変える
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229
, 0.224, 0.225])
    ])
# train data読み込み
train_dataset = torchvision.datasets.ImageFolder(root='animal/train
', #修正
                                           transform=data_transform
)
test_dataset = torchvision.datasets.ImageFolder(root='animal/test',
#修正
                                           transform=data_transform
)
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

In [5]:

```
# Convolutional neural network (two convolutional layers)

# Pretrained model
import torchvision.models as models

#for idx, m in enumerate(models.resnet18().modules()):
        #print(idx, '->', m)

class Resnet(nn.Module):

    def __init__(self, num_classes=10):
        super(Resnet,self).__init__()
        resnet = models.resnet18(pretrained=True) #
        resnet.fc = nn.Linear(512, 10)            #
        self.resnet = resnet

    def forward(self,x):
        x = self.resnet(x)
        return x

model = Resnet(num_classes).to(device)


# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

In [6]:

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        print(i)

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, lo
ss.item()))
```

```
0
1
2
```

```
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
0
1
2
```

```
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
```

In [7]:

```python
# Test the model
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        print(total)

    print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))

# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```

```
100
Test Accuracy of the model on the 10000 test images: 65.0 %
```

感想

学習率を少し変えただけで全く学習してなかったり、実行するのに長い時間がかかったのは驚きだった。一番の驚きは、少しでも早くなるかと思いAIXのGPUを使おうとしたが、こちらの問題のせいか、うまく繋げず、プログラムを実行できなかったことである。（問い合わせたところ他の人は使えたらしい）

参考文献

「Pythonによる深層学習」のサンプルコード

http://pr.cei.uec.ac.jp/kobo2018/index.php?Pythonによる深層学習 (http://pr.cei.uec.ac.jp/kobo2018/index.php?Pythonによる深層学習)

「PyTorch-tutorial」

https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/02-intermediate/convolutional_neural_network (https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/02-intermediate/convolutional_neural_network)