CS398-Deep Learning
Homework 4
Lingyi Xu (lingyix2)

1. Description of Implementation

First, I implement the deep convolutional neural network following the architecture of lecture 6 slides.

Example of convolution network for CIFAR10:

- Convolution layer 1: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 2: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling: $s = 2$, $k = 2$.
- Dropout
- Convolution layer 3: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 4: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling → Dropout

(Continued)

- Convolution layer 5: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 6: 64 channels, $k = 3, s = 1, P = 0$.
- Dropout
- Convolution layer 7: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization
- Convolution layer 8: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization, Dropout
- Fully connected layer 1: 500 units.
- Fully connected layer 2: 500 units.
- Linear → Softmax function

```
def forward(self,x,extract_features=False):
    x = F.relu(self.bn1(self.conv1(x)))
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, kernel_size=2, stride=2)
    x = self.drop1(x)
    x = F.relu(self.bn2(self.conv3(x)))
    x = F.relu(self.conv4(x))
    x = F.max_pool2d(x, kernel_size=2, stride=2)
    x = F.relu(self.bn3(self.conv5(x)))
    x = F.relu(self.conv6(x))
    x = self.drop3(x)
    x = F.relu(self.bn4(self.conv7(x)))
    x = F.relu(self.bn5(self.conv8(x)))
    x = self.drop4(x)
    x = x.view(-1, 4*4*64)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

Then I load and prepare the training data and test data of CIFAR10. When loading data, I used Transforms to normalize and enlarge the dataset to help training. For Transforms, I used random flip, random resize and random color jitter.

```
transform_train = transforms.Compose([
    transforms.RandomResizedCrop(DIM,scale=(0.7,1.0),ratio=(1.0,1.0)),
    transforms.ColorJitter(
            brightness = 0.1*torch.randn(1),
            contrast = 0.1*torch.randn(1),
            saturation = 0.1*torch.randn(1),
            hue = 0.1*torch.randn(1)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)),
])

transform_test = transforms.Compose([
    transforms.CenterCrop(DIM),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)),
])

trainset = torchvision.datasets.CIFAR10(root='./', train=True, download=True, transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./', train=False, download=False, transform=transform_test)

trainloader = torch.utils.data.DataLoader(trainset, batch_size= batch_size, shuffle=True, num_workers=8)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=8)
```

Train the model with GPU acceleration. For this particular dataset, I use the following parameters:

Iteration = 30, Batch size = 120, Learning rate = 0.001/（0.1*itr+1）

```python
for epoch in range(0,num_epochs):

    for param_group in optimizer.param_groups:
        param_group['lr'] = learning_rate/(1+epoch*0.1)

    train_accu = []
    model.train()

    for batch_idx, (X_train_batch,Y_train_batch) in enumerate(trainloader):

        if(Y_train_batch.shape[0]<batch_size):
            continue

        X_train_batch = Variable(X_train_batch).cuda()
        Y_train_batch = Variable(Y_train_batch).cuda()
        output = model(X_train_batch)

        loss = criterion(output, Y_train_batch)
        optimizer.zero_grad()

        loss.backward()
        optimizer.step()

        prediction = output.data.max(1)[1]
        accuracy = (float(prediction.eq(Y_train_batch.data).sum())/float(batch_size))*100.0

        train_accu.append(accuracy)
    accuracy_epoch = np.mean(train_accu)
    print(epoch,accuracy_epoch)
```

In the end, apply Monte Carlo methods, heuristic test methods, and normal test methods:

```python
    # Heuristic Prediction Rule
    model.eval()
    # Normal Prediction
    with torch.no_grad():
        test_accu = []
        for batch_idx, (x_test_batch,y_test_batch) in enumerate(testloader):
            X_test_batch, Y_test_batch = Variable(x_test_batch).cuda(),Variable(y_test_batch).cuda()
            output = model(X_test_batch)
            prediction = output.data.max(1)[1]
            accuracy = (float(prediction.eq(Y_test_batch.data).sum())/float(batch_size))*100.0
            test_accu.append(accuracy)
            accuracy_test = np.mean(test_accu)
    print("Testing",accuracy_test)

#------------when using monte carlo prediction, use this part
#    # Monte Carlo Prediction
#    with torch.no_grad():
#        test_accu = []
#        for batch_idx, (X_test_batch,Y_test_batch) in enumerate(testloader):
#            X_test_batch, Y_test_batch = Variable(X_test_batch).cuda(),Variable(Y_test_batch).cuda()
#            prediction_vector = Variable(torch.tensor(np.zeros((50,120,10)))).cuda()
#            for i in range(monte_carlo_size):
#                output = model(X_test_batch)
#                temp_prediction = output.data
#                prediction_vector[i] = temp_prediction
#            prediction = (prediction_vector.mean(0)).max(1)[1]
#            accuracy = (float(prediction.eq(Y_test_batch.data).sum())/float(batch_size))*100.0
#            test_accu.append(accuracy)
#        mc_test_acc = np.mean(test_accu)
#    print("Monte Carlo test accuracy:",mc_test_acc)
```

2. **Final Test Accuracy:**
   Normal: 83.9369%

Monte Carlo Method: 84.9326%
Heuristic Method: 84.0257%

3. **Structure of CNN (from lecture slides):**

Example of convolution network for CIFAR10:

- Convolution layer 1: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 2: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling: $s = 2$, $k = 2$.
- Dropout
- Convolution layer 3: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 4: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling $\rightarrow$ Dropout

(Continued)

- Convolution layer 5: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 6: 64 channels, $k = 3, s = 1, P = 0$.
- Dropout
- Convolution layer 7: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization
- Convolution layer 8: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization, Dropout
- Fully connected layer 1: 500 units.
- Fully connected layer 2: 500 units.
- Linear $\rightarrow$ Softmax function