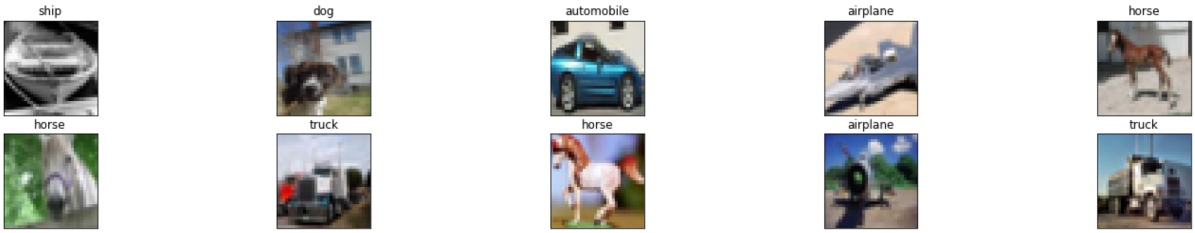
Shengkai Xu
ECG 4106 - RealTime AI
03/29/2022
Homework3

Github: https://github.com/ric3b0wl/RealTimeAI/tree/master/hw3

Problem 1

Build a Convolutional Neural Network, like what we built in lectures (without skip connections), to classify the images across all 10 classes in CIFAR 10. You need to adjust the fully connected layer at the end properly with respect to the number of output classes. Train your network for 300 epochs. Report your training time, training loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare them against a fully connected network (homework 2) on training time, achieved accuracy, and model size.

First, we check if the images are loaded:



Then we define the model architecture:

```python
# define the CNN architecture
class Net1(nn.Module):
    def __init__(self):
        super(Net1, 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):
        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)
        return x
```

```
Net1(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

After 300 epochs of training, we are not able to get very good accuracy, after putting into the test data set for verification we are only getting 65% accuracy for the average of 10 classes.

```
Epoch: 1
training:
100%|██████████| 2000/2000 [00:11<00:00, 181.50batch/s]
validation:
100%|██████████| 500/500 [00:02<00:00, 223.27batch/s]
Training Loss: 1.757462         Validation Loss: 0.389919
Validation loss decreased (inf --> 0.389919).  Saving model ...
```

```
Epoch: 300
training:
100%|██████████| 2000/2000 [00:11<00:00, 171.80batch/s]
validation:
100%|██████████| 500/500 [00:02<00:00, 212.60batch/s]Training Loss: 0.079974    Validation Loss: 1.412274
CPU times: user 1h 7min 6s, sys: 58.3 s, total: 1h 8min 5s
Wall time: 1h 8min 42s
```

Wall time display 1 hour and 8 min for this model.

```
Test Loss: 1.030446

Test Accuracy of airplane: 73% (732/1000)
Test Accuracy of automobile: 75% (752/1000)
Test Accuracy of  bird: 45% (457/1000)
Test Accuracy of   cat: 40% (402/1000)
Test Accuracy of  deer: 60% (601/1000)
Test Accuracy of   dog: 67% (670/1000)
Test Accuracy of  frog: 65% (656/1000)
Test Accuracy of horse: 75% (754/1000)
Test Accuracy of  ship: 76% (767/1000)
Test Accuracy of truck: 71% (712/1000)

Test Accuracy (Overall): 65% (6503/10000)
```

For the second model, we add one more layer and max-pooling function, but the result was still not very pleasing, so I decided to add a 50% of drop out. Finally, we are getting up to 74% of average accuracy across 10 classes of images.

```python
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        # convolutional layer
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)
        # fully connected layers
        self.fc1 = nn.Linear(64 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 64)
        self.fc3 = nn.Linear(64, 10)
        # dropout
        self.dropout = nn.Dropout(p=.5)

    def forward(self, x):
        # add sequence of convolutional and max pooling layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        # flattening
        x = x.view(-1, 64 * 4 * 4)
        # fully connected layers
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

```
Net2(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1024, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

```
Epoch: 1
training:
100%|██████████| 2000/2000 [00:12<00:00, 159.64batch/s]
validation:
100%|██████████| 500/500 [00:02<00:00, 213.12batch/s]
Training Loss: 1.798779         Validation Loss: 0.408045
Validation loss decreased (inf --> 0.408045).  Saving model ...
```

```
Epoch: 300
training:
100%|██████████| 2000/2000 [00:12<00:00, 159.50batch/s]
validation:
100%|██████████| 500/500 [00:02<00:00, 200.72batch/s]Training Loss: 0.035946    Validation Loss: 0.476834
CPU times: user 1h 13min 2s, sys: 1min, total: 1h 14min 2s
Wall time: 1h 14min 33s
```

Wall time displayed 1 hour and 14 mins for the second model which are a little bit longer than the first model by 6 mins.

```
Test Loss: 0.796236

Test Accuracy of airplane: 76% (761/1000)
Test Accuracy of automobile: 82% (829/1000)
Test Accuracy of  bird: 66% (666/1000)
Test Accuracy of   cat: 56% (562/1000)
Test Accuracy of  deer: 67% (675/1000)
Test Accuracy of   dog: 66% (666/1000)
Test Accuracy of  frog: 80% (804/1000)
Test Accuracy of horse: 80% (803/1000)
Test Accuracy of  ship: 82% (829/1000)
Test Accuracy of truck: 83% (831/1000)

Test Accuracy (Overall): 74% (7426/10000)
```

Problem 2

Extend your CNN by adding one more additional convolution layer followed by an activation function and pooling function. You also need to adjust your fully connected layer properly with respect to intermediate feature dimensions. Train your network for 300 epochs. Report your training time, loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare your model size and accuracy over the baseline implementation in Problem1.a. Do you see any over-fitting? Make sure to submit your code by providing the GitHub URL of your course repository for this course.

With Resnet we are building a deeper network than problem 1. Using two-block architecture, we can build up to 100+ layers in a convolution neural network.

```
class ResBlock(nn.Module):
    def __init__(self, n_chans):
        super(ResBlock, self).__init__()
        self.conv = nn.Conv2d(n_chans, n_chans, kernel_size=3,
                              padding=1, bias=False)  # <1>
        self.batch_norm = nn.BatchNorm2d(num_features=n_chans)
        torch.nn.init.kaiming_normal_(self.conv.weight,
                                      nonlinearity='relu')  # <2>
        torch.nn.init.constant_(self.batch_norm.weight, 0.5)
        torch.nn.init.zeros_(self.batch_norm.bias)

    def forward(self, x):
        out = self.conv(x)
        out = self.batch_norm(out)
        out = torch.relu(out)
        return out + x
```

```
class NetResDeep(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

```
model1 = NetResDeep(n_chans1=32, n_blocks=10).to(device=device)
optimizer = optim.SGD(model1.parameters(), lr=3e-3,  weight_decay=1e-4)
loss_fn = nn.CrossEntropyLoss()
```

```
Epoch 1,
Training loss 1.6299726059436799
100%|██████████| 2000/2000 [00:12<00:00, 166.45it/s]
Accuracy train: 0.49
100%|██████████| 500/500 [00:03<00:00, 165.80it/s]
Accuracy val: 0.48
```

```
Epoch 300,
Training loss 0.032437252582235 84
100%|██████████| 2000/2000 [00:11<00:00, 168.84it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.15it/s]Accuracy val: 0.65
CPU times: user 1h 22min 23s, sys: 29.5 s, total: 1h 22min 53s
Wall time: 1h 22min 38s
```

Wall time displayed: 1 hour 22 min, Restnet model a little bit longer time than the problem 1 regular convolution model.

Next we set the weight decay lambada to 0.001

```
model1 = NetResDeep(n_chans1=32, n_blocks=10).to(device=device)
optimizer = optim.SGD(model1.parameters(), lr=3e-3,  weight_decay=1e-4)
loss_fn = nn.CrossEntropyLoss()
```

```
Epoch 300,
Training loss 0.047725999124603104
100%|██████████| 2000/2000 [00:11<00:00, 173.03it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.54it/s]Accuracy val: 0.65
CPU times: user 1h 22min 19s, sys: 27.8 s, total: 1h 22min 47s
Wall time: 1h 22min 33s
```

We dont much difference in terms of training time and we are getting about the same accuracy between train and validaiont set.

Additionally we set a dropout rate to decrease the chance of overfitting

```
class NetResDeepDropout(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc2 = nn.Linear(32, 10)
        self.dropout2d = nn.Dropout2d(p=0.3)
        self.dropout = nn.Dropout(p=0.3)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.dropout2d(out)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = self.dropout2d(out)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1(out))
        out = self.dropout(out)
        out = self.fc2(out)
        return out
```

```
model = NetResDeep(n_chans1=32, n_blocks=10).to(device=device)
optimizer = optim.SGD(model.parameters(), lr=3e-3)
criterion = nn.CrossEntropyLoss() # loss function
```

```
Epoch 300,
Training loss 1.0545503511428833
100%|██████████| 2000/2000 [00:15<00:00, 130.91it/s]
Accuracy train: 0.63
100%|██████████| 500/500 [00:03<00:00, 133.17it/s]Accuracy val: 0.57
CPU times: user 1h 51min 2s, sys: 30.7 s, total: 1h 51min 33s
Wall time: 1h 51min 17s
```

With dropout rate of 30% we are getting a long training rate, I believe with great of dropout rate we can achieve a better accuracy and training time.

Finally we add batch normlization to the model

```
class NetResDeepBatchNorm(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_norm = nn.BatchNorm2d(n_chans1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc1_norm = nn.BatchNorm1d(32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1_norm(self.conv1(x))), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1_norm(self.fc1(out)))
        out = self.fc2(out)
        return out
```

```
model3 = NetResDeepBatchNorm(n_chans1=32, n_blocks=10).to(device=device)
optimizer3 = optim.SGD(model3.parameters(), lr=9e-3)
criterion3 = nn.CrossEntropyLoss()
```

```
Epoch 300,
Training loss 0.08433806079177884
100%|██████████| 2000/2000 [00:12<00:00, 164.83it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:03<00:00, 162.87it/s]Accuracy val: 0.64
CPU times: user 1h 24min 51s, sys: 33.4 s, total: 1h 25min 25s
Wall time: 1h 25min 15s
```

This show batch norlization have much better performance, it dropped training time significantly along with accuracy of the training.