(Pytorch 版本 - 1.9.0)

implementation details

Model Structure

I use All Convolution Net, the detail model structure is as shown below.

First, build each layer according to paper.(total 9 convolution layer & 6 ReLU function)
Then, add 2 dropout layer to prevent overfitting.
Finally, use adaptiveavgpool2d layer to let it output 10 value.(the larger this value, the greater probability that it belong to this class)

(according to paper **STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET**)

image From paper

All-CNN-C
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 conv. 96 ReLU with stride $r = 2$
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
3×3 conv. 192 ReLU with stride $r = 2$

image from my code

```python
class AllConvNet(nn.Module):
    def __init__(self, input_size, n_classes=10, **kwargs):
        super(AllConvNet, self).__init__()

        self.conv1 = nn.Conv2d(input_size, 96, 3, padding=1, stride=1)
        self.relu1 = nn.ReLU()

        self.conv2 = nn.Conv2d(96, 96, 3, padding=1, stride=1)
        self.relu2 = nn.ReLU()

        self.conv3 = nn.Conv2d(96, 96, 3, padding=1, stride=2)
        self.dropout3 = nn.Dropout(p=0.5)

        self.conv4 = nn.Conv2d(96, 192, 3, padding=1, stride=1)
        self.relu4 = nn.ReLU()

        self.conv5 = nn.Conv2d(192, 192, 3, padding=1, stride=1)
        self.relu5 = nn.ReLU()

        self.conv6 = nn.Conv2d(192, 192, 3, padding=1, stride=2)
        self.dropout6 = nn.Dropout(p=0.5)

        self.conv7 = nn.Conv2d(192, 192, 3, padding=1, stride=1)
        self.relu7 = nn.ReLU()

        self.conv8 = nn.Conv2d(192, 192, 3, stride=1, padding=1)
        self.relu8 = nn.ReLU()

        self.class_conv = nn.Conv2d(192, n_classes, 1, padding=0, stride=1)
        self.poolout = nn.AdaptiveAvgPool2d(1)

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv1_out = self.relu1(conv1_out)

        conv2_out = self.conv2(conv1_out)
        conv2_out = self.relu2(conv2_out)

        conv3_out = self.conv3(conv2_out)
        conv3_out_drop = self.dropout3(conv3_out)

        conv4_out = self.conv4(conv3_out_drop)
        conv4_out = self.relu4(conv4_out)

        conv5_out = self.conv5(conv4_out)
        conv5_out = self.relu5(conv5_out)

        conv6_out = self.conv6(conv5_out)
        conv6_out_drop = self.dropout6(conv6_out)

        conv7_out = self.conv7(conv6_out_drop)
        conv7_out = self.relu7(conv7_out)

        conv8_out = self.conv8(conv7_out)
        conv8_out = self.relu8(conv8_out)

        class_out = self.class_conv(conv8_out)

        pool_out = self.poolout(class_out)

        pool_out.squeeze_(-1)
        pool_out.squeeze_(-1)
        return pool_out
```

Regularization:

Use Dropout layer as a way to regularization

Data augmentation and Preprocess:

Train_data will do data augmentation including horizontal flip and affine.

Then, do normalize.(test_data only do normalize)

```
train_data = GetLoader(x_train, y_train,transforms=transform_train)
test_data = GetLoader(x_test, y_test,transforms=transform_show)
```

```
transform_train = transforms.Compose([

        transforms.ToPILImage(),
        transforms.RandomHorizontalFlip(),
        transforms.RandomAffine(degrees=0, translate=(0.15, 0.15)),
        transforms.ToTensor(),
        transforms.Normalize((0.4915,0.4823, 0.4469), (0.2023, 0.2433, 0.2616))
    ])

transform_show = transforms.Compose([

        transforms.ToPILImage() ,
        transforms.ToTensor(),
        transforms.Normalize((0.4915,0.4823, 0.4469), (0.2023, 0.2433, 0.2616))
    ])
```

Hyperparameters

batch size=64/640

Learning rate =0.01

Weight_decay = 1e-3

Monmentun = 0.9

Nesterove = true

(Using SGD as optimizer and using Cross Entropy as loss function)

```
    criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.SGD([
        {'params': weight_p, 'weight_decay': 1e-3},
        {'params': bias_p, 'weight_decay': 0}
], lr=1e-2, momentum=0.9, nesterov=True)
```

Training method

First, I use batch size=64 to train my model 200 epoch.
Then, use batch size=640 to continue training.
In this way, I found best result, test_set Accuracy > 90%.

Result

```
Accuracy of my model on test-set:  0.9056
```