実装演習 2-5.最新のCNN

以下サイトを参考に、PyTorchでAlexNetを実装しCIFAR-10の学習を行う。
[http://cedro3.com/ai/pytorch-alexnet/ (http://cedro3.com/ai/pytorch-alexnet/)](http://cedro3.com/ai/pytorch-alexnet/)

In [1]:
```
import torch
import torchvision
import torch.nn as nn
import torch.nn.init as init
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [2]:
```
batch_size = 64
num_classes = 10
num_epochs = 20
```

In [3]:
```
# GPUならcuda、CPUならcpuを表示する
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

CIFAR-10のデータを読み込む

In [4]:
```
# 学習用データセット
train_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=True, transform=transforms.ToTensor(), download=True)
```

```
Files already downloaded and verified
```

In [5]:
```
# テスト用データセット
test_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=False, transform=transforms.ToTensor(), download=True)
```

```
Files already downloaded and verified
```

In [6]:
```
print(f'train_dataset: {len(train_dataset)}')
print(f'test_dataset: {len(test_dataset)}')
```

```
train_dataset: 50000
test_dataset: 10000
```

In [7]:
```
# データセットの中身を表示
image, label = train_dataset[0]
print(image.size())
print(label)
```

```
torch.Size([3, 32, 32])
6
```

In [8]:
```
# データローダーの設定
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True, nu
m_workers=2)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False, nu
m_workers=2)
```

AlexNetを構築する

ILSVRCの画像は224*224だが、CIFAR-10は32*32のため、入力層を以下のように変更する。

- kernel_sizeを11から3にする
- paddingを2から1にする

また、MaxPoolingのkernel_sizeを3から2にする。
そうすることで、画像サイズの変更が32→16→8→4になる。

In [9]:
```python
class AlexNet(nn.Module):
    """AlexNetモデルの構築"""
    def __init__(self, num_classes):
        super(AlexNet, self).__init__()
        # 畳み込み層（第1層）
        self.block1 = nn.Sequential(
            # CIFAR-10は3*32*32
            nn.Conv2d(3, 96, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))
        # 畳み込み層（第2層）
        self.block2 = nn.Sequential(
            nn.Conv2d(96, 256, kernel_size=(5, 5), padding=(2, 2)),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))
        # 畳み込み層（第3層）
        self.block3 = nn.Sequential(
            nn.Conv2d(256, 384, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(inplace=True))
        # 畳み込み層（第4層）
        self.block4 = nn.Sequential(
            nn.Conv2d(384, 384, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(inplace=True))
        # 畳み込み層（第5層）
        self.block5 = nn.Sequential(
            nn.Conv2d(384, 256, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)))

        # 全結合層
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 4 * 4, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = x.view(x.size(0), 256 * 4 * 4)
        x = self.classifier(x)
        return x
```

In [10]:
```python
net = AlexNet(num_classes).to(device)
```

In [11]:
```python
# 交差エントロピー誤差
criterion = nn.CrossEntropyLoss()
# 最適化手法
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4)
```

学習を行う

In [12]:
```python
train_loss_list = []
train_acc_list = []
val_loss_list = []
val_acc_list = []
```

In [13]:

```
for epoch in range(num_epochs):
    train_loss, train_acc, val_loss, val_acc = 0, 0, 0, 0

    # 学習モード
    net.train()
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()  # 勾配初期化
        outputs = net(images)  # モデル学習

        loss = criterion(outputs, labels)
        train_loss += loss.item()
        train_acc += (outputs.max(1)[1] == labels).sum().item()
        # 逆誤差伝搬
        loss.backward()
        optimizer.step()

    avg_train_loss = train_loss / len(train_loader.dataset)
    avg_train_acc = train_acc / len(train_loader.dataset)

    # 検証モード
    net.eval()
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = net(images)  # モデル検証
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            val_acc += (outputs.max(1)[1] == labels).sum().item()

    avg_val_loss = val_loss / len(test_loader.dataset)
    avg_val_acc = val_acc / len(test_loader.dataset)

    print('Epoch [{}/{}], Loss: {loss:.4f}, val_loss: {val_loss:.4f}, val_acc: {val_acc:.4
f}'.format(
        epoch+1, num_epochs, i+1, loss=avg_train_loss, val_loss=avg_val_loss, val_acc=avg
_val_acc))

    train_loss_list.append(avg_train_loss)
    train_acc_list.append(avg_train_acc)
    val_loss_list.append(avg_val_loss)
    val_acc_list.append(avg_val_acc)
```

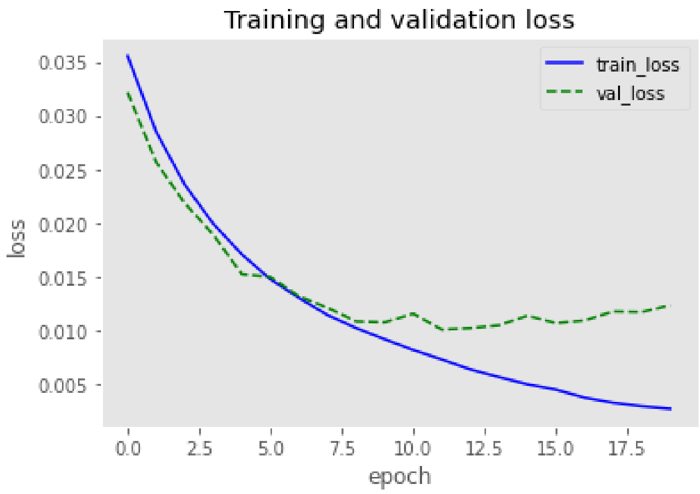
```
Epoch [1/20], Loss: 0.0355, val_loss: 0.0322, val_acc: 0.2342
Epoch [2/20], Loss: 0.0285, val_loss: 0.0256, val_acc: 0.3966
Epoch [3/20], Loss: 0.0236, val_loss: 0.0218, val_acc: 0.4832
Epoch [4/20], Loss: 0.0199, val_loss: 0.0189, val_acc: 0.5551
Epoch [5/20], Loss: 0.0171, val_loss: 0.0152, val_acc: 0.6605
Epoch [6/20], Loss: 0.0148, val_loss: 0.0150, val_acc: 0.6726
Epoch [7/20], Loss: 0.0130, val_loss: 0.0132, val_acc: 0.7092
Epoch [8/20], Loss: 0.0114, val_loss: 0.0121, val_acc: 0.7356
Epoch [9/20], Loss: 0.0102, val_loss: 0.0108, val_acc: 0.7597
Epoch [10/20], Loss: 0.0092, val_loss: 0.0108, val_acc: 0.7690
Epoch [11/20], Loss: 0.0082, val_loss: 0.0115, val_acc: 0.7579
Epoch [12/20], Loss: 0.0073, val_loss: 0.0101, val_acc: 0.7839
Epoch [13/20], Loss: 0.0064, val_loss: 0.0102, val_acc: 0.7865
Epoch [14/20], Loss: 0.0057, val_loss: 0.0105, val_acc: 0.7873
Epoch [15/20], Loss: 0.0050, val_loss: 0.0114, val_acc: 0.7786
Epoch [16/20], Loss: 0.0045, val_loss: 0.0107, val_acc: 0.7872
Epoch [17/20], Loss: 0.0037, val_loss: 0.0109, val_acc: 0.7902
Epoch [18/20], Loss: 0.0033, val_loss: 0.0118, val_acc: 0.7859
Epoch [19/20], Loss: 0.0029, val_loss: 0.0117, val_acc: 0.7940
Epoch [20/20], Loss: 0.0027, val_loss: 0.0123, val_acc: 0.7864
```

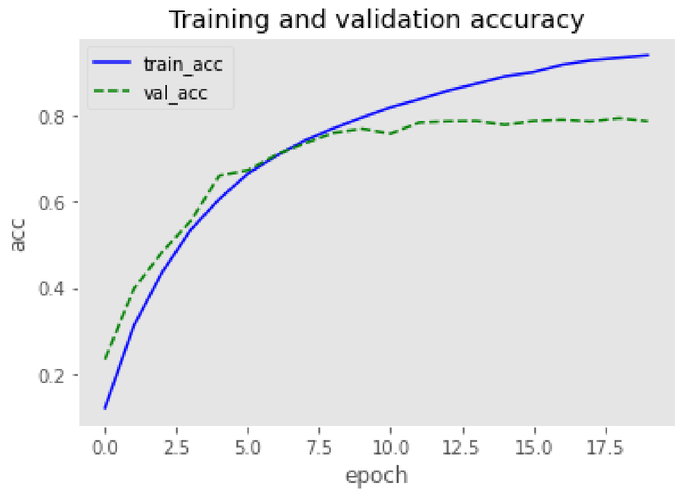
グラフの描画

In [14]:

```
plt.figure()
plt.plot(range(num_epochs), train_loss_list, color='blue', linestyle='-', label='train_loss')
plt.plot(range(num_epochs), val_loss_list, color='green', linestyle='--', label='val_loss')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.title('Training and validation loss')
plt.grid()

plt.figure()
plt.plot(range(num_epochs), train_acc_list, color='blue', linestyle='-', label='train_acc')
plt.plot(range(num_epochs), val_acc_list, color='green', linestyle='--', label='val_acc')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('acc')
plt.title('Training and validation accuracy')
plt.grid()
plt.show()
```





In [ ]: