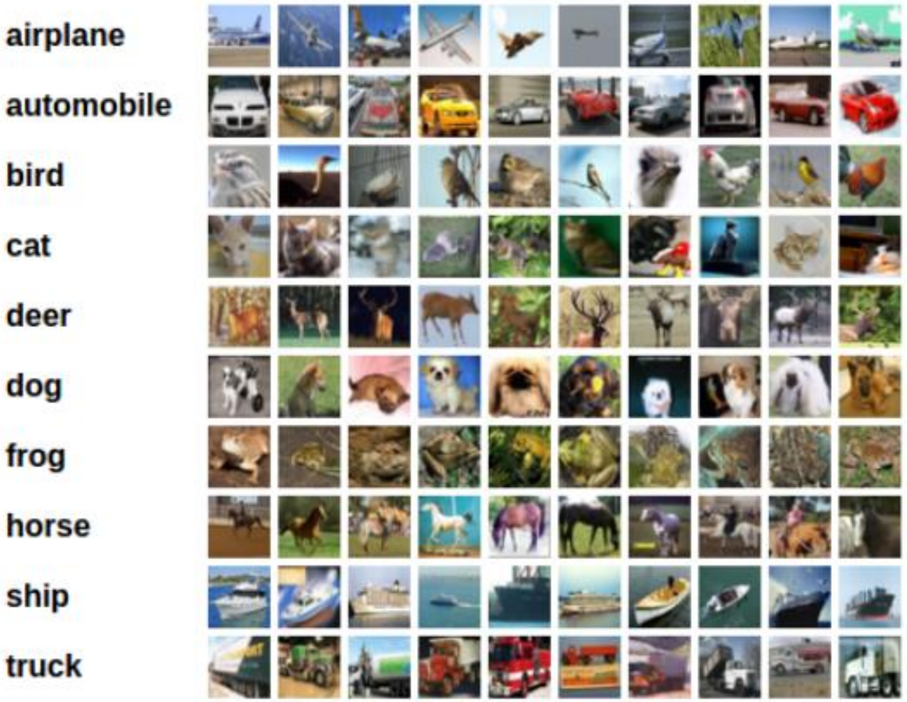
Standalone Deep-Learning

week 7 (lec 21~22)

Boaz 17기 분석 정성경

Data Loader

✔ CIFAR-10 Dataset



✔ torchvision.datasets.CIFAR10

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])

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

partition = {'train': trainset, 'val':valset, 'test':testset}
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
170499072/? [00:11<00:00, 17380553.47it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

✔ 50000 images for train → 40000 : 10000
✔ 10000 images for test

Data Shape

✔ Input shape for Conv2d layer

```
[19]
BATCH_SIZE = 256

trainloader = torch.utils.data.DataLoader(partition['train'],
                                          batch_size=BATCH_SIZE, shuffle=True, num_workers=2)

for (X_train, y_train) in trainloader:
    print('X_train:', X_train.size(),'type:', X_train.type())
    print('y_train:', y_train.size(),'type:', y_train.type())
    break
```

```
X_train: torch.Size([256, 3, 32, 32]) type: torch.FloatTensor
y_train: torch.Size([256]) type: torch.LongTensor
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d

CNN (Convolutional Neural Network)

✔ CNN

```
[20]
class CNN1(nn.Module):

    def __init__(self):
        super(CNN1, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,
                               out_channels = 64,
                               kernel_size = 3,
                               stride = 1,
                               padding = 1)
        self.conv2 = nn.Conv2d(in_channels = 64,
                               out_channels = 256,
                               kernel_size = 5,
                               stride = 1,
                               padding = 2)
        self.act = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size = 2,
                                     stride = 2)
        self.fc = nn.Linear(256*16*16, 10)

    def forward(self, x):
        x = self.conv1(x)          # (N, 64, 32, 32)
        x = self.act(x)
        x = self.conv2(x)          # (N, 256, 32, 32)
        x = self.act(x)
        x = self.maxpool1(x)       # (N, 256, 16, 16)
        x = x.view(x.size(0), -1)  # (N, 256*16*16)
        x = self.fc(x)             # (N, 10)
        return x
```

✔ Pytorch : Object로 구성된 layer → forward 함수

✔ **Conv1**
- In_channels : RGB 3-dim
- Out_channels : # of filter
- Kernel_size : Size of filter
- Stride : filter를 몇 칸 씩 건너뛰는지
- Padding : Zero padding 얼마나 할지

✔ **Conv2**
- In_channels : Out_channels of prev. layer
- Dimension 유지를 위해 kernel_size 5 → padding 2
  - (32 + 2*2 – 5) / 1 + 1 = 32

✔ **Maxpool1**
- Size를 절반으로 줄이기 위해서는
- Kernel_size = 2
- Stride = 2

✔ **Fc (fully connected layer)**

CNN (Convolutional Neural Network)

✔ CNN

```
[20]
class CNN1(nn.Module):

    def __init__(self):
        super(CNN1, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,
                               out_channels = 64,
                               kernel_size = 3,
                               stride = 1,
                               padding = 1)
        self.conv2 = nn.Conv2d(in_channels = 64,
                               out_channels = 256,
                               kernel_size = 5,
                               stride = 1,
                               padding = 2)
        self.act = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size = 2,
                                     stride = 2)
        self.fc = nn.Linear(256*16*16, 10)

    def forward(self, x):
        x = self.conv1(x)           # (N, 64, 32, 32)
        x = self.act(x)
        x = self.conv2(x)           # (N, 256, 32, 32)
        x = self.act(x)
        x = self.maxpool1(x)        # (N, 256, 16, 16)
        x = x.view(x.size(0), -1)   # (N, 256*16*16)
        x = self.fc(x)              # (N, 10)
        return x
```

✔ Fc (fully connected layer) input shape

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d

Shape:

- Input: (N, C, H_{in}, W_{in})
- Output: (N, C, H_{out}, W_{out}), where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

```
[21]
def dimension_check():
    net = CNN1()
    x = torch.randn(2, 3, 32, 32)
    y = net(x)
    print(y.size())
```

VGG (very deep convolutional networks for large-scale image recognition)

```python
class CNN(nn.Module):

    def __init__(self, model_code, in_channels, out_dim, act, use_bn):
        super(CNN, self).__init__()

        if act == 'relu':
            self.act = nn.ReLU()
        elif act == 'sigmoid':
            self.act = nn.Sigmoid()
        elif act == 'tanh':
            self.act = nn.TanH()
        else:
            raise ValueError("Not a valid activation function code")

        self.layers = self._make_layers(model_code, in_channels, use_bn)
        self.classifer = nn.Sequential(nn.Linear(512, 256),
                                       self.act,
                                       nn.Linear(256, out_dim))

    def forward(self, x):
        x = self.layers(x)
        x = x.view(x.size(0), -1)
        x = self.classifer(x)
        return x

    def _make_layers(self, model_code, in_channels, use_bn):
        layers = []
        for x in cfg[model_code]:
            if x == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [nn.Conv2d(in_channels=in_channels,
                                     out_channels=x,
                                     kernel_size=3,
                                     stride=1,
                                     padding=1)]
                if use_bn:
                    layers += [nn.BatchNorm2d(x)]
                layers += [self.act]
                in_channels = x
        return nn.Sequential(*layers)
```

[6]
```python
cfg = {
    'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'VGG13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'],
    'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'],
}
```

- ✔ Layer List → nn.Sequential(*layers)
- ✔ Batch Normalization
- ✔ fc linear layer input shape : (512*1*1)

```python
# ====== Model ====== #
args.model_code = 'VGG11'
args.in_channels = 3
args.out_dim = 10
args.act = 'relu'

# ====== Regularization ======= #
args.l2 = 0.00001
args.use_bn = True

# ====== Optimizer & Training ====== #
args.optim = 'RMSprop' #'RMSprop' #SGD, RMSprop, ADAM...
args.lr = 0.0015
args.epoch = 10

args.train_batch_size = 256
args.test_batch_size = 1024
```

VGG *(very deep convolutional networks for large-scale image recognition)*

- ✔ VGG13, hidden unit 4000, RMSprop, Epoch 150 : 81.8%
- ✔ VGG16, hidden unit 500, Dropout 0.3, RMSprop, Epoch 150 : 82.9%
- ✔ VGG13, hidden unit 500, Dropout 0.3, Adam, Epoch 300 : 79.6%

위 top3 모델을 seed를 다르게 준 후 앙상블 : **86.3%**

Thank you !