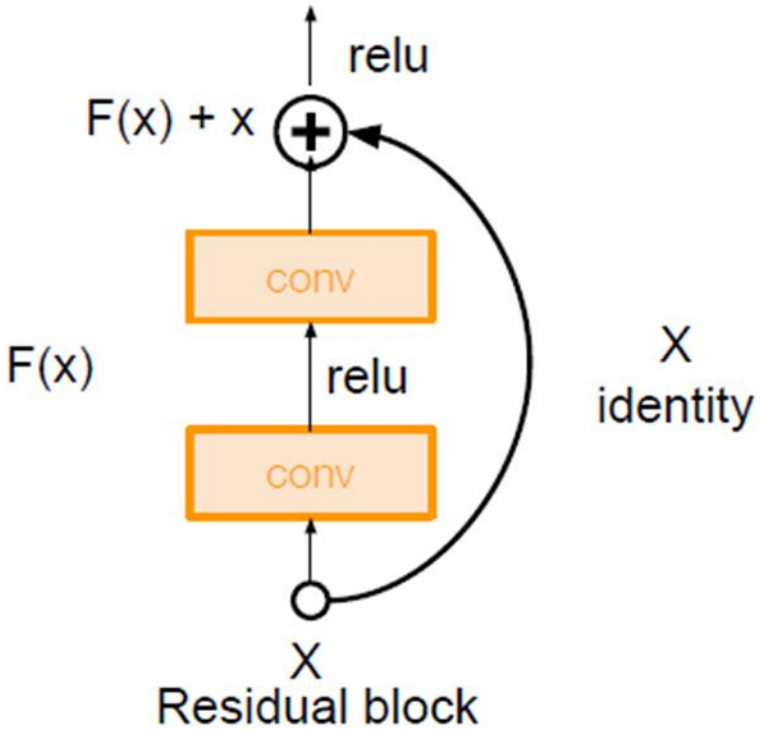
이미지 분류모델 구현 – ResNet

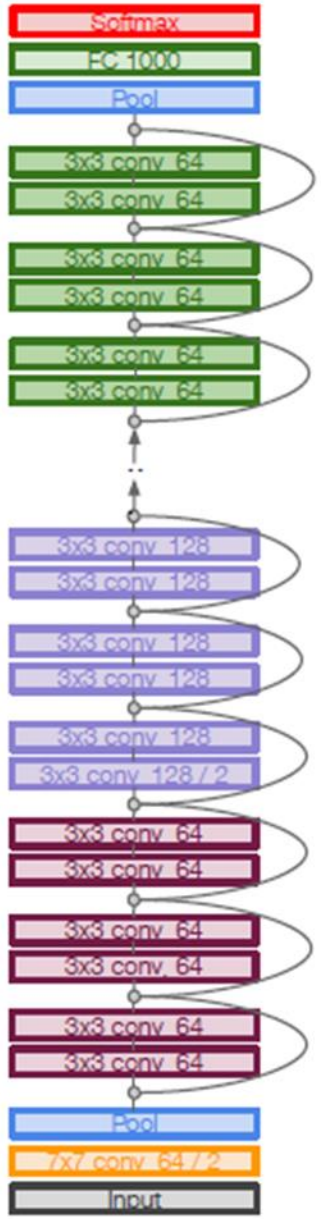
박경규

ResNet

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





CIFAR-10 분류모델 - ResNet

https://github.com/Justin-A/DeepLearning101/blob/master/4-4_CIFAR_ResNet.ipynb

```
class BasicBlock(nn.Module):
    def __init__(self, in_planes, planes, stride = 1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
                in_planes, planes, kernel_size = 3, stride = stride, padding = 1, bias = False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(
                planes, planes, kernel_size = 3, stride = 1, padding = 1, bias = False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size = 1, stride = stride, bias = False),
                nn.BatchNorm2d(planes))
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

CIFAR-10 분류모델 - ResNet

https://github.com/Justin-A/DeepLearning101/blob/master/4-4_CIFAR_ResNet.ipynb

```
class ResNet(nn.Module):
 def __init__(self, num_classes = 10):
     super(ResNet, self).__init__()
     self.in_planes = 16

     self.conv1 = nn.Conv2d(3, 16, kernel_size = 3, stride = 1, padding = 1, bias = False)
     self.bn1 = nn.BatchNorm2d(16)
     self.layer1 = self._make_layer(16, 2, stride = 1)
     self.layer2 = self._make_layer(32, 2, stride = 2)
     self.layer3 = self._make_layer(64, 2, stride = 2)
     self.linear = nn.Linear(64, num_classes)

 def _make_layer(self, planes, num_blocks, stride):
     strides = [stride] + [1] * (num_blocks  - 1)
     layers = []
     for stride in strides:
         layers.append(BasicBlock(self.in_planes, planes, stride))
         self.in_planes = planes
     return nn.Sequential(*layers)
```

CIFAR-10 분류모델 - ResNet

https://github.com/Justin-A/DeepLearning101/blob/master/4-4_CIFAR_ResNet.ipynb

```
def forward(self, x):
    out = F.relu(self.bn1(self.conv1(x)))
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = F.avg_pool2d(out, 8)
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out
```

CIFAR-10 분류모델 - ResNet

print(model)

```
ResNet(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
                                    ..........................

  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (linear): Linear(in_features=64, out_features=10, bias=True)
)
```

CIFAR-10 분류모델 - ResNet

```
Train Epoch: 10 [0/50000 (0%)]  Train Loss: 0.378728
Train Epoch: 10 [6400/50000 (13%)]      Train Loss: 0.287413
Train Epoch: 10 [12800/50000 (26%)]     Train Loss: 0.228947
Train Epoch: 10 [19200/50000 (38%)]     Train Loss: 0.468466
Train Epoch: 10 [25600/50000 (51%)]     Train Loss: 0.539970
Train Epoch: 10 [32000/50000 (64%)]     Train Loss: 0.399627
Train Epoch: 10 [38400/50000 (77%)]     Train Loss: 0.882485
Train Epoch: 10 [44800/50000 (90%)]     Train Loss: 0.332883

[EPOCH: 10],    Test Loss: 0.5320,      Test Accuracy: 82.03 %
```

torchvision.models

https://pytorch.org/vision/stable/models.html

Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet

CIFAR-10 분류모델 – torchvision.models

https://github.com/Justin-A/DeepLearning101/blob/master/4-5_Load-Pre-trained_Model.ipynb

PyTorch 내에서 제공하는 ResNet34 모델 불러온 후 FC 층 추가 및 Output 크기 설정하기

```
import torchvision.models as models

model = models.resnet34(pretrained = False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)
model = model.cuda()

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
criterion = nn.CrossEntropyLoss()

BATCH_SIZE = 32
EPOCHS = 10
```

CIFAR-10 분류모델 – torchvision.models

```
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

CIFAR-10 분류모델 – torchvision.models

```
model = models.resnet34(pretrained = False)
```

```
Train Epoch: 10 [0/50000 (0%)]  Train Loss: 0.679751
Train Epoch: 10 [6400/50000 (13%)]      Train Loss: 0.493025
Train Epoch: 10 [12800/50000 (26%)]     Train Loss: 0.722751
Train Epoch: 10 [19200/50000 (38%)]     Train Loss: 0.459644
Train Epoch: 10 [25600/50000 (51%)]     Train Loss: 0.621173
Train Epoch: 10 [32000/50000 (64%)]     Train Loss: 0.344826
Train Epoch: 10 [38400/50000 (77%)]     Train Loss: 0.316430
Train Epoch: 10 [44800/50000 (90%)]     Train Loss: 0.455933

[EPOCH: 10],    Test Loss: 0.6742,      Test Accuracy: 78.39 %
```

CIFAR-10 분류모델 – torchvision.models

```
model = models.resnet34(pretrained = True)
```

```
Train Epoch: 10 [0/50000 (0%)]  Train Loss: 0.187622
Train Epoch: 10 [6400/50000 (13%)]      Train Loss: 0.147061
Train Epoch: 10 [12800/50000 (26%)]     Train Loss: 0.217833
Train Epoch: 10 [19200/50000 (38%)]     Train Loss: 0.226637
Train Epoch: 10 [25600/50000 (51%)]     Train Loss: 0.361518
Train Epoch: 10 [32000/50000 (64%)]     Train Loss: 0.158119
Train Epoch: 10 [38400/50000 (77%)]     Train Loss: 0.386843
Train Epoch: 10 [44800/50000 (90%)]     Train Loss: 0.164800

[EPOCH: 10],    Test Loss: 0.6856,      Test Accuracy: 81.65 %
```

Thank you

kgpark88@gmail.com