7주차 예습과제

공부할 범위 - 6.1.1~6.1.2

6. 합성곱 신경망2

6.1 이미지 분류를 위한 신경망

6.2 객체 인식을 위한 신경망

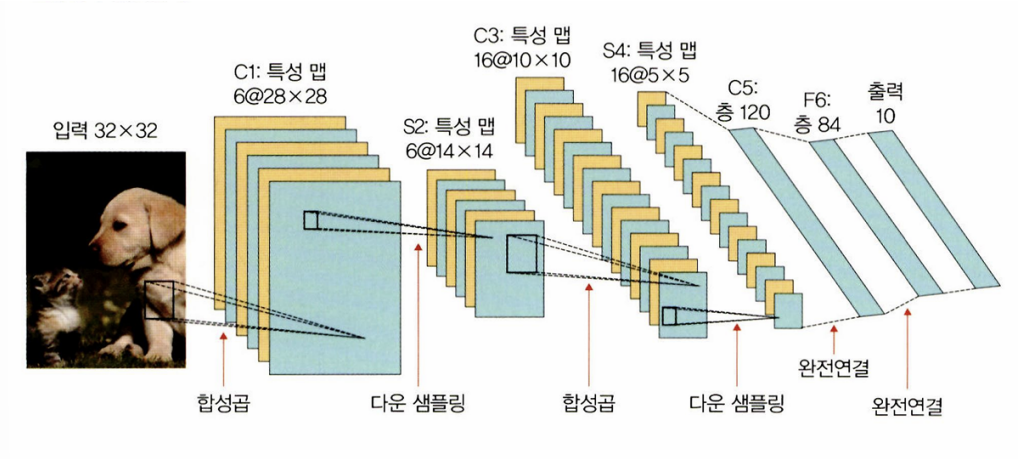
6.3 이미지 분할을 위한 신경망

6.1 이미지 분류를 위한 신경망

입력 데이터가 이미지인 classification은 특정 대상이 영상 내에 존재하는지 여부를 판단하는 문제.

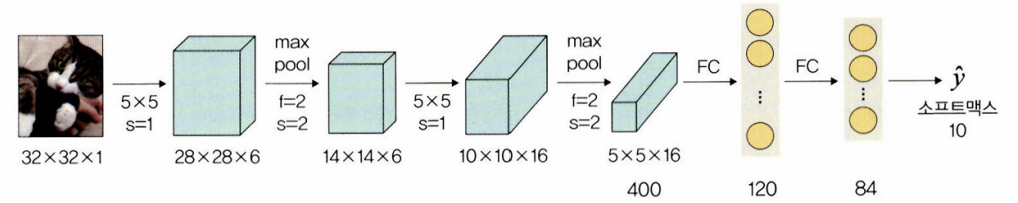
6.1.1 LeNet-5

CNN의 초석. 합성곱과 다운 샘플링을 반복적으로 거치면서 마지막에 완전연결층에서 classification을 수행.



이 그림은 C1에서 5X5 합성곱 연산을 수행한 후 28X28 크기의 특성맵을 여섯 개 생성. S2에서 다운 샘플링을 진행해 14X14로 크기를 줄임. C3에서 다시 5X5 합성곱 연산을 해 10X10 크기의 특성 맵 16개를 생성. S4에서 다운 샘플링으로 5X5로 줄임. C5에서 5X5 합

성곱 연산으로 1X1 크기의 특성맵 120개 생성. F6에서 완전연결층으로 C5의 결과를 84개 유닛에 연결. 앞에서 다룬 예제를 LeNet-5를 사용하면 아래와 같은 과정을 거침.



실습을 위한 코드는 다음과 같다.

- 필요한 라이브러리

```
import torch
import torchvision
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from torch.autograd import Variable
from torch import optim
import torch.nn as nn
import torch.nn.functional as F
import os
import cv2
from PIL import Image
from tqdm import tqdm_notebook as tqdm
import random
from matplotlib import pyplot as plt

device = torch.device("cuda:0" if torch.cuda.is_available() e
```

- 데이터 전처리 코드

```
class ImageTransform():
    def __init__(self, resize, mean, std):
        self.data_transform = {
            'train': transforms.Compose([
                transforms.RandomResizedCrop(resize, scale=(0
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize(mean, std)
```

```
            ]),
            'val': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(resize),
                transforms.ToTensor(),
                transforms.Normalize(mean, std)
            ])
        }

    def __call__(self, img, phase):
        return self.data_transform[phase](img)
```

5장에서 다룬 것과 유사.

- 데이터를 가져와 훈련, 검증, 테스트 데이터로 분리

```
cat_directory = 'dogs-vs-cats/Cat/'
dog_directory = 'dogs-vs-cats/Dog/'

cat_images_filepaths = sorted([os.path.join(cat_directory, f)
dog_images_filepaths = sorted([os.path.join(dog_directory, f)
images_filepaths = [*cat_images_filepaths, *dog_images_filepa
correct_images_filepaths = [i for i in images_filepaths if cv

random.seed(42)
random.shuffle(correct_images_filepaths)
train_images_filepaths = correct_images_filepaths[:400]
val_images_filepaths = correct_images_filepaths[400:-10]
test_images_filepaths = correct_images_filepaths[-10:]
print(len(train_images_filepaths), len(val_images_filepaths),
```

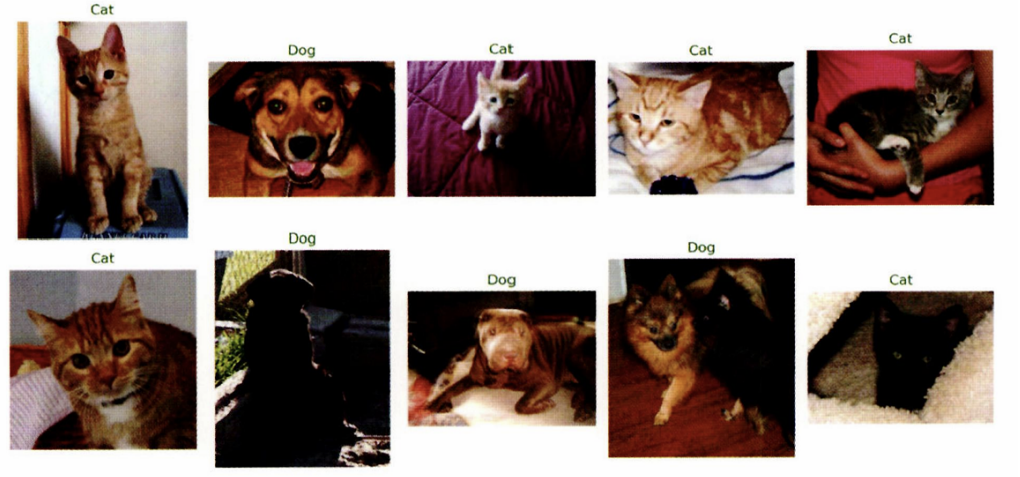
sorted를 사용하면 데이터가 정렬된 리스트로 반환됨.
images_filepaths에서 이미지 파일들을 불러오고 for문을 이용해 가져온 데이터에 대해 i를 이용해 리스트로 만듦.
random.see(42)로 난수를 생성.

- 테스트 데이터셋의 이미지 확인

```
def display_image_grid(images_filepaths, predicted_labels=(),
    rows = len(images_filepaths) // cols
    figure, ax = plt.subplots(nrows=rows, ncols=cols, figsize
    for i, image_filepath in enumerate(images_filepaths):
        image = cv2.imread(image_filepath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        true_label = os.path.normpath(image_filepath).split(o
        predicted_label = predicted_labels[i] if predicted_la
        color = "green" if true_label == predicted_label else
        ax.ravel()[i].imshow(image)
        ax.ravel()[i].set_title(predicted_label, color=color)
        ax.ravel()[i].set_axis_off()
    plt.tight_layout()
    plt.show()
```

cv2.cvtColor로 이미지의 색상 변경.
os.path.normpath(image_filepath).split(os.sep)[-2] : 이미지 전체 경로를 정규화하고 분할.
predicted_lable에 대한 값 정의.

- 테스트 데이터셋 이미지 출력



- 이미지 데이터셋 클래스 정의

```
class DogvsCatDataset(Dataset):
    def __init__(self, file_list, transform=None, phase='trai
        self.file_list = file_list
```

```
        self.transform = transform
        self.phase = phase

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        img_path = self.file_list[idx]
        img = Image.open(img_path)
        img_transformed = self.transform(img, self.phase)

        label = img_path.split('/')[-1].split('.')[0]
        if label == 'dog':
            label = 1
        elif label == 'cat':
            label = 0
        return img_transformed, label
```

데이터를 불러오는 방법을 정의하는 클래스. 고양이가 있는 이미지의 레이블은 0, 개가 있는 이미지의 레이블은 1.

- 변수 값 정의

```
size = 224
mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)
batch_size = 32
```

- 이미지 데이터셋 정의

```
train_dataset = DogvsCatDataset(train_images_filepaths, trans
val_dataset = DogvsCatDataset(val_images_filepaths, transform

index = 0
print(train_dataset.__getitem__(index)[0].size())
print(train_dataset.__getitem__(index)[1])
```

훈련과 검증 데이터셋을 정의. 전처리도 진행.

- 데이터로더 정의

```
train_dataloader = DataLoader(train_dataset, batch_size=batch
val_dataloader = DataLoader(val_dataset, batch_size=batch_siz
dataloader_dict = {'train': train_dataloader, 'val': val_data

batch_iterator = iter(train_dataloader)
inputs, label = next(batch_iterator)
print(inputs.size())
print(label)
```

데이터를 한 번에 로드하지 않고 여러 batch로 나눠서 불러옴.

- 모델 네트워크 클래스

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=3, out_channels=16,
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        self.cnn2 = nn.Conv2d(in_channels=16, out_channels=32
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Linear(32*53*53, 512)
        self.relu5 = nn.ReLU()
        self.fc2 = nn.Linear(512, 2)
        self.output = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.cnn1(x)
        out = self.relu1(out)
        out = self.maxpool1(out)
        out = self.cnn2(out)
        out = self.relu2(out)
        out = self.maxpool2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
```

```
        out = self.output(out)
        return out
```

- 모델 객체 생성

```
model = LeNet().to(device)
print(model)
```

- 모델 네트워크 구조 확인

```
from torchsummary import summary
summary(model, input_size=(3, 224, 224))
```

torchsummary를 이용해 모델의 네트워크 관련 정보를 확인할 수 있음.

- 학습 가능한 파라미터 수

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.req

print(f'The model has {count_parameters(model):,} trainable p
```

- 옵티마이저, 손실 함수 정의

```
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=
criterion = nn.CrossEntropyLoss()
```

모멘텀 SGD 사용. 모멘텀 SGD는 SGD에 관성이 추가된 것인데, 매번 기울기를 구하되 가중치를 수정하기 전에 이전 수정 방향을 참고해 같은 방향으로 일정한 비율만큼 수정되게 하는 방법.

- 모델 파라미터와 손실함수 CPU에 할당

```
model = model.to(device)
criterion = criterion.to(device)
```

- 모델 학습 함수 정의

```
def train_model(model, dataloader_dict, criterion, optimizer,
    since = time.time()
    best_acc = 0.0

    for epoch in range(num_epoch):
        print('Epoch {}/{}'.format(epoch + 1, num_epoch))
        print('-'*20)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            epoch_loss = 0.0
            epoch_corrects = 0

            for inputs, labels in tqdm(dataloader_dict[phase]
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train')
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                    epoch_loss += loss.item() * inputs.size(0
                    epoch_corrects += torch.sum(preds == labe

            epoch_loss = epoch_loss / len(dataloader_dict[pha
            epoch_acc = epoch_corrects.double() / len(dataloa

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase,
```

```
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = model.state_dict()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))
    return model
```

- 모델 학습

```
import time

num_epoch = 10
model = train_model(model, dataloader_dict, criterion, optimi
```

- 모델 테스트 함수 정의

```
import pandas as pd

id_list = []
pred_list = []
_id=0
with torch.no_grad():
    for test_path in tqdm(test_images_filepaths):
        img = Image.open(test_path)
        _id =test_path.split('/')[-1].split('.')[1]
        transform = ImageTransform(size, mean, std)
        img = transform(img, phase='val')
        img = img.unsqueeze(0)
        img = img.to(device)

        model.eval()
        outputs = model(img)
        preds = F.softmax(outputs, dim=1)[:, 1].tolist()
        id_list.append(_id)
```

```
        pred_list.append(preds[0])

res = pd.DataFrame({
    'id': id_list,
    'label': pred_list
})

res.sort_values(by='id', inplace=True)
res.reset_index(drop=True, inplace=True)

res.to_csv('LesNet.csv', index=False)
```

torch.unsqueeze: 텐서에 차원을 추가할 때 사용. (0)은 차원이 추가될 위치를 의미.

softmax: 지정된 차원을 따라 텐서의 요소가 범위에 있고 합계가 1이 되도록 크기 조정.

- 테스트 데이터셋의 예측 결과

```
res.head(10)
```

- 테스트 데이터셋 이미지 출력

```
class_ = classes = {0:'cat', 1:'dog'}
def display_image_grid(images_filepaths, predicted_labels=(),
    rows = len(images_filepaths) // cols
    figure, ax = plt.subplots(nrows=rows, ncols=cols, figsize
    for i, image_filepath in enumerate(images_filepaths):
        image = cv2.imread(image_filepath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        a = random.choice(res['id'].values)
        label = res.loc[res['id'] == a, 'label'].values[0]
        if label > 0.5:
            label = 1
        else:
            label = 0
        ax.ravel()[i].imshow(image)
        ax.ravel()[i].set_title(class_[label])
        ax.ravel()[i].set_axis_off()
```

```
    plt.tight_layout()
    plt.show()
```

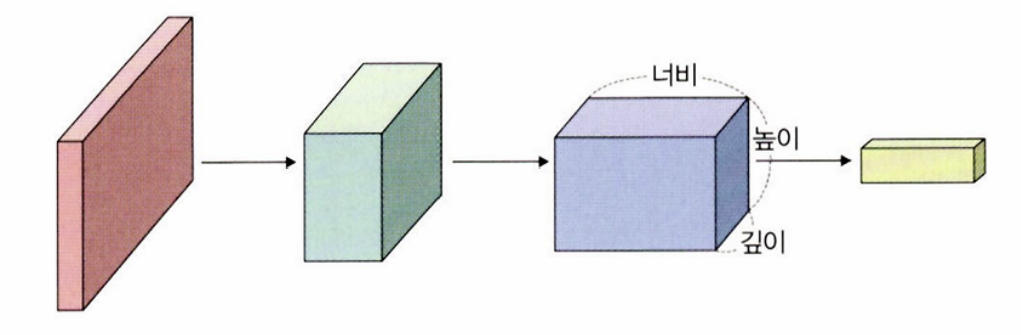
threshold가 0.5, 레이블이 0.5보다 크면 개고 0.5보다 작으면 고양이.

- 테스트 데이터셋 예측 결과 이미지 출력

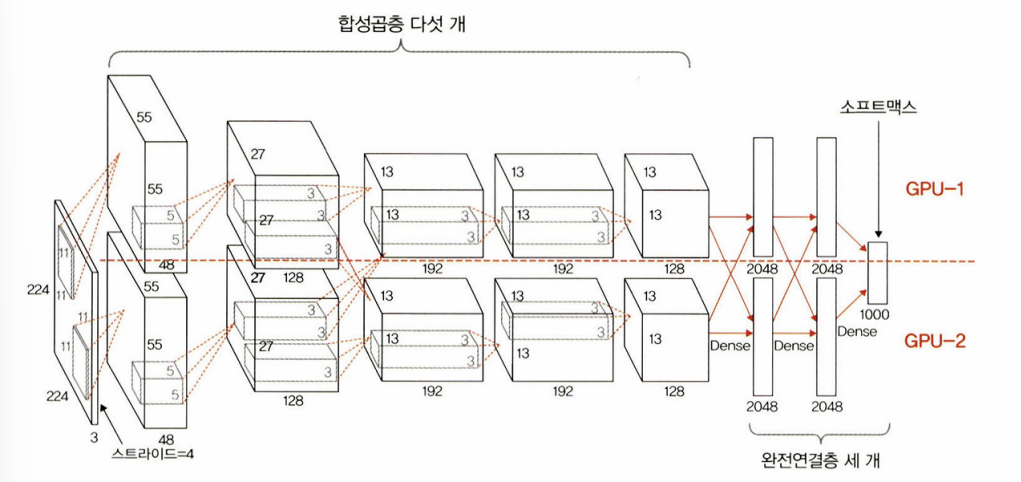
```
display_image_grid(test_images_filepaths)
```

6.1.2 AlexNet

AlexNet도 CNN을 기본으로 함. CNN 구조를 한 번 더 살펴보자면, CNN은 이미지를 다루기 때문에 3차원 구조를 가짐.



이를 기반으로 하는 AlexNet 구조는 합성곱층 다섯 개와 완전연결층 세 개로 구성되어 있음. 마지막 완전연결층은 소프트맥스 활성화 함수를 사용.



AlexNet의 첫 번째 합성곱층 커널의 크기는 11X11X3이고 스트라이드를 4로 적용하여 특성맵을 96개 생성하기 때문에 55X55X96의 출력을 가짐. 첫 번째 계층을 거치며 GPU-1에서

는 주로 컬러와는 상관없는 정보를 추출하기 위한 커널이 학습되고, GPU-2에서 주로 컬러와 관련된 정보를 추출하기 위한 커널이 학습됨. 마찬가지로 코드를 살펴봄. 대체로 LeNet의 코드와 유사하기 때문에 차이가 나는 부분 위주로 정리함.

- AlexNet 모델 네트워크 정의

```
class AlexNet(nn.Module):
    def __init__(self) -> None:
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, paddin
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 2),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
```

```
        x = self.classifier(x)
        return x
```