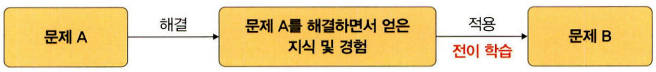
```
        if not (count % 500):
            print("Iteration: {}, Loss: {}, Accuracy: {}%".format(count, loss.data, accurac
y))
```

5.3 전이 학습

전이 학습 : 이미지넷처럼 아주 큰 데이터셋을 써서 훈련된 모델의 가중치를 가져와 우리가 해결하려는 과제에 맞게 보정해서 사용하는 것

-> 방법 : 1) 특성 추출 2) 미세 조정 기법



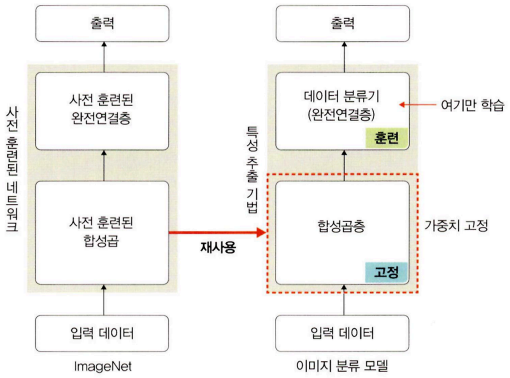
5.3.1 특성 추출 기법

특성 추출(Feature Extractor) : ImageNet 데이터셋으로 사전 훈련된 모델을 가져온 후 마지막에 완전 연결층 부분만 새로 생성

-> 마지막 완전 연결층만 학습하고 나머지 계층들은 학습하지 않음

- 구성 : 1) **합성곱층** : 합성곱층 + 풀링층 2) **데이터 분류기(완전연결층)** : 최종적으로 이미지에 대한 클래스를 분류하는 부분

- 사용가능한 이미지 분류 모델 : Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet



 # 5-12 라이브러리 호출

```
import os
import time
import copy
import glob
import cv2  //OpenCV 라이브러리
import shutil

import torch
import torchvision  //computer vision 용도의 패키지
import torchvision.transforms as transforms  //데이터 전처리를 위한 패키지
import torchvision.models as models  //다양한 파이토치 네트워크를 위한 패키지
import torchvision.datasets as datasets
import torch.nn as nn
import torch.optim as optim

import matplotlib.pyplot as plt
```

 # 5-13 이미지 데이터 전처리 방법 정의

```
data_path = './catanddog/train/'
transform = transforms.Compose(
                [
                    transforms.Resize([256, 256]),
                    transforms.RandomResizedCrop(224),
                    transforms.RandomHorizontalFlip(),
                    transforms.ToTensor(),
                ])  //torchvision/transform : 이미지 데이터를 모델 입력으로 사용할 수 있도록 변환
train_dataset = torchvision.datasets.ImageFolder(
```

```
    data_path,
    transform=transform
)  //datasets.ImageFolder : 데이터로더가 데이터를 불러올 대상과 방법 정의
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=32,
    num_workers=8,
    shuffle=True
)  //ImageFolder를 데이터로더에 할당하고 batch_size wlwjd

print(len(train_dataset))
```

5-14 학습에 사용될 이미지 출력

```
import numpy as np
samples, labels = next(iter(train_loader))  //iter : 전달된 데이터의 반복자 반환, next : 다음 출력할 요소 반환
classes = {0:'cat', 1:'dog'}  //개, 고양이에 대한 클래스로 구성
fig = plt.figure(figsize=(16,24))
for i in range(24):  //24개 이미지 데이터 출력
    a = fig.add_subplot(4,6,i+1)
    a.set_title(classes[labels[i].item()])  //레이블 정보를 함께 출력
    a.axis('off')
    a.imshow(np.transpose(samples[i].numpy(), (1,2,0)))  //np.transpose : 행&열을 바꿔 행렬의 차원을 바꿈
plt.subplots_adjust(bottom=0.2, top=0.6, hspace=0)
```

5-15 사전 훈련된 모델 내려받기

```
resnet18 = models.resnet18(pretrained=True)
```

ResNet18 : 50개 계층으로 구성된 합성곱 신경망
- 입력 제약이 크고 충분한 RAM이 없으면 학습 속도가 느릴 수 있음

5-16 사전 훈련된 모델의 파라미터 학습 유무 지정

```
def set_parameter_requires_grad(model, feature_extracting=True):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False  //역전파 중 파라미터들에 대한 변화를 계산할 필요가 없다는 뜻

set_parameter_requires_grad(resnet18)
```

ResNet18의 합성곱층을 사용하되 파라미터에 대해서는 학습을 하지 않도록 고정

이때 모델의 일부는 합성곱층 + 풀링층
이후 마지막 부분에 완전연결층 추가

5-17 ResNet18에 완전연결층 추가

```
resnet18.fc = nn.Linear(512, 2)  //2는 클래스가 두개라는 뜻
```

추가된 완전연결층은 개와 고양이 클래스를 분류하는 용도

5-18 모델의 파라미터 값 확인

```
for name, param in resnet18.named_parameters():  //model.name_parameters() : 모델에 접근해 파라미터값 가져올 때 사용
    if param.requires_grad:
        print(name, param.data)
```

5-19 모델 객체 생성 및 손실 함수 정의

```
model = models.resnet18(pretrained = True)  //모델 객체 생성

for param in model.parameters():  //모델의 합성곱층 가중치 고정
    param.requires_grad = False

model.fc = torch.nn.Linear(512, 2)
for param in model.fc.parameters():  //완전연결층은 학습
```

```
    param.requires_grad = True

optimizer = torch.optim.Adam(model.fc.parameters())
cost = torch.nn.CrossEntropyLoss()  //손실함수 정의
print(model)
```

5-20 모델 학습을 위한 함수 생성

```
def train_model(model, dataloaders, criterion, optimizer, device, num_epochs=13, is_train=Tr
ue):
    since = time.time()  //컴퓨터 현재 시각을 구하는 함수
    acc_history = []
    loss_history = []
    best_acc = 0.0

    for epoch in range(num_epochs):  //epoch(13)만큼 반복
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders:  //데이터로더에 전달된 데이터만큼 반복
            inputs = inputs.to(device)
            labels = labels.to(device)

            model.to(device)
            optimizer.zero_grad()  //기울기를 0으로 설정
            outputs = model(inputs)  //순전파 학습
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)
            loss.backward()  //역전파 학습
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            //출력결과와 레이블의 오차를 계산한 결과를 누적해 저장
            running_corrects += torch.sum(preds == labels.data)
            //출력결과와 레이블이 동일한지 확인한 결과 누적해 저장

        epoch_loss = running_loss / len(dataloaders.dataset)  //평균 오차 계산
        epoch_acc = running_corrects.double() / len(dataloaders.dataset)  //평균 정확도 계산

        print('Loss: {:.4f} Acc: {:.4f}'.format(epoch_loss, epoch_acc))

        if epoch_acc > best_acc:
            best_acc = epoch_acc

        acc_history.append(epoch_acc.item())
        loss_history.append(epoch_loss)
        torch.save(model.state_dict(), os.path.join('catanddog/', '{0:0=2d}.pth'.format(epoc
h)))
        print()

    time_elapsed = time.time() - since  //실행시간 계산
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 6
0))
    print('Best Acc: {:4f}'.format(best_acc))
    return acc_history, loss_history  //모델의 정확도와 오차 반환
```

5-21 파라미터 학습 결과를 옵티마이저에 전달

```
params_to_update = []
for name,param in resnet18.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)  //파라미터 학습 결과를 저장
        print("\t",name)

optimizer = optim.Adam(params_to_update)  //학습 결과를 옵티마이저에 전달
```

5-22 모델 학습

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
criterion = nn.CrossEntropyLoss()
```

```
train_acc_hist, train_loss_hist = train_model(resnet18, train_loader, criterion, optimizer,
device)
```

모델 훈련을 위해 전달되는 파라미터 : 모델, 학습데이터, 손실함수, 옵티마이저, 장치(CPU/GPU)

5-23 테스트 데이터 호출 및 전처리

```
test_path = './catanddog/test'

transform = transforms.Compose(
                [
                    transforms.Resize(224),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                ])
test_dataset = torchvision.datasets.ImageFolder(
    root=test_path,
    transform=transform
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=32,
    num_workers=1,
    shuffle=True
)

print(len(test_dataset))
```

5-24 테스트 데이터 평가 함수 생성

```
def eval_model(model, dataloaders, device):
    since = time.time()
    acc_history = []
    best_acc = 0.0

    saved_models = glob.glob('catanddog/' + '*.pth')  //glob : 현재 dir에서 원하는 파일만 추출할 때
사용
    saved_models.sort()
    print('saved_model', saved_models)

    for model_path in saved_models:
        print('Loading model', model_path)

        model.load_state_dict(torch.load(model_path))
        model.eval()
        model.to(device)
        running_corrects = 0

        for inputs, labels in dataloaders:  //테스트 반복
            inputs = inputs.to(device)
            labels = labels.to(device)

            with torch.no_grad():  //Autograd를 사용하지 않겠다는 것
                outputs = model(inputs)  //데이터를 모델에 적용한 결과를 outputs에 저장

            _, preds = torch.max(outputs.data, 1)  //torch.max : 주어진 텐서배열의 최대값이 들어있는
index 반환
            preds[preds >= 0.5] = 1  //torch.max의 출력값이 >0.5이면 올바르게 예측
            preds[preds < 0.5] = 0  //<0.5이면 틀리게 예측
            running_corrects += preds.eq(labels).int().sum()  //preds.eq(labes) : preds배열과
labels가 일치하는지 검사

        epoch_acc = running_corrects.double() / len(dataloaders.dataset)  //테스트 데이터의 정확
도 계산
        print('Acc: {:.4f}'.format(epoch_acc))

        if epoch_acc > best_acc:
            best_acc = epoch_acc

        acc_history.append(epoch_acc.item())
        print()

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

```
    return acc_history  //계산된 정확도 반환
```

 # 5-25 테스트 데이터를 평가 함수에 적용

```
val_acc_hist = eval_model(resnet18, test_loader, device)
```

 # 5-26 훈련과 테스트 데이터의 정확도를 그래프로 확인

```
plt.plot(train_acc_hist)
plt.plot(val_acc_hist)
plt.show()
```

 # 5-27 훈련 데이터의 오차에 대한 그래프 확인

```
plt.plot(train_loss_hist)
plt.show()
```

 # 5-28 예측 이미지 출력을 위한 전처리 함수

```
def im_convert(tensor):
    image=tensor.clone().detach().numpy()  //tensor.clone() : 기존 텐서 내용을 복사한 텐서를 생성
    //detach() : 기존 텐서에서 기울기가 전파되지 않는 텐서
    //tensor.clone().detach() : 기존 텐서를 복사한 새 텐서를 생성하지만 기울기에 영향을 주지는 않는다는 것
    image=image.transpose(1,2,0)
    image=image*(np.array((0.5,0.5,0.5))+np.array((0.5,0.5,0.5)))
    image=image.clip(0,1)  //clip() : 입력값을 특정 범위로 제한시키기 위해 사용
    return image
```

구분	메모리	계산 그래프 상주 유무
tensor.clone()	새롭게 할당	계산 그래프에 계속 상주
tensor.detach()	공유해서 사용	계산 그래프에 상주하지 않음
tensor.clone().detach()	새롭게 할당	계산 그래프에 상주하지 않음

 # 5-29 개와 고양이 예측 결과 출력

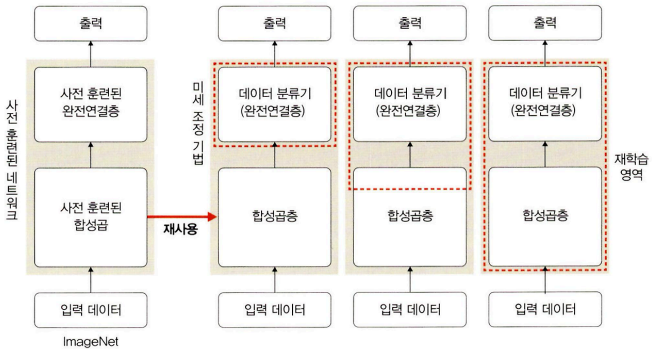
```
classes = {0:'cat', 1:'dog'}  //개&고양이 2개에 대한 레이블

dataiter=iter(test_loader)  //테스트 데이터셋 가져옴
images,labels=dataiter.next()  //테스트 데이터셋에서 이미지와 레이블 분리해 가져옴
output=model(images)
_,preds=torch.max(output,1)

fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
    ax=fig.add_subplot(2,10,idx+1,xticks=[],yticks=[])  //add_subplot() : 한 화면에 여러개 이미지 담을 때 사용
    plt.imshow(im_convert(images[idx]))  //이미지 출력을 위해 im_convert 함수 적용
    a.set_title(classes[labels[i].item()])
    ax.set_title("{}({})".format(str(classes[preds[idx].item()]),str(classes[labels[idx].item()])),color=("green" if preds[idx]==labels[idx] else "red"))
    //classes[pres[idx].item()] : preds[ids].item()값이 0과 1 중 어떤 값을 갖는지 판별하겠다는 것
plt.show()
plt.subplots_adjust(bottom=0.2, top=0.6, hspace=0)  //figure 안에서 subplot의 위치를 조정할 때 사용
```

5.3.2 미세 조정 기법

미세 조정(fine-tuning) 기법 : 사전 훈련된 모델과 합성곱층, 데이터 분류기의 가중치를 업데이트해 훈련시키는 방식

- **전략** :
  1) 데이터셋이 크고 사전 훈련된 모델과 유사성이 작을 경우 : 모델 전체를 재학습
  2) 데이터셋이 크고 사전 훈련된 모델과 유사성이 클 경우 : 합성곱층의 뒷부분과 데이터 분류기를 학습
  3) 데이터셋이 작고 사전 훈련된 모델과 유사성이 작을 경우 : 합성곱층의 일부분과 데이터 분류기를 학습
  4) 데이터셋이 작고 사전 훈련된 모델과 유사성이 클 경우 : 데이터 분류기만 학습

- **과적합 문제를 주의**해야 함



5.4 설명 가능한 CNN

설명 가능한 CNN(Explainable CNN) : 딥러닝 처리 결과를 사람이 이해할 수 있는 방식으로 제시하는 기술
- 내부에서 설명하기 어렵고, 얻은 결과를 신뢰하기 어렵기 때문에 **시각화할 필요성**이 있음

5.4.1 특성 맵 시각화

특성 맵(feature map)/활성화 맵 : 필터를 입력에 적용한 결과

예시)
**PIL(Python Image Library) 사전에 설치 필요

 # 5-30 필요한 라이브러리 호출

```
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import torch
import torch.nn.functional as F
import torch.nn as nn
from torchvision.transforms import ToTensor
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

 # 5-31 설명 가능한 네트워크 생성

```
class XAI(torch.nn.Module):
    def __init__(self, num_classes=2):
        super(XAI, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Conv2d(64, 64, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),  //inplace=True : 기존 데이터를 연산의 결괏값으로 대체
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(128, 128, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(128, 256, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(256),
```

```
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(256, 256, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(256, 256, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(256, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(512, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(512, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(512, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(512, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(512, 512, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(512, 512, bias=False),
            nn.Dropout(0.5),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(-1, 512)
        x = self.classifier(x)
        return F.log_softmax(x)
        //log_softmax() : 신경망 말단의 결괏값들을 확률 개념으로 해석하기 위해 소프트맥스 함수의 결과에 log값 취
한 연산
```

-> 13개의 합성곱층과 2개의 완전연결층으로 구성된 네트워크 생성

-> ReLU 활성화 함수 사용

5-32 모델 객체화

```
model=XAI()  //model이라는 이름의 객체 생성
model.cpu()  //model을 CPU에 할당
model.eval()  //테스트 데이터에 대한 모델 평가 용도로 사용
```

5-33 특성 맵을 확인하기 위한 클래스 정의

```
class LayerActivations:
    features=[]
    def __init__(self, model, layer_num):
        self.hook = model[layer_num].register_forward_hook(self.hook_fn)
        //hook : 각 계층의 활성화 함수 및 기울기 값 확인 가능

    def hook_fn(self, module, input, output):
        output = output
        #self.features = output.to(device).detach().numpy()
        self.features = output.detach().numpy()
```

```
    def remove(self):
        self.hook.remove()
```

5-34 이미지 호출

```
img=cv2.imread("./cat.jpg")
plt.imshow(img)
img = cv2.resize(img, (100, 100), interpolation=cv2.INTER_LINEAR)  //cv2.resize : 이미지 크기 변경
img = ToTensor()(img).unsqueeze(0)
//ToTensor()(img).unsqueeze(0) : 이미지를 텐서로 변환하고 1차원 데이터로 변경하겠다는 것

print(img.shape)
```

5-35 Conv2d 특성 맵 확인

```
result = LayerActivations(model.features, 0)  //0번째 Conv2d 특성 맵 확인
model(img)
activations = result.features
```

5-36 특성 맵 확인

```
fig, axes = plt.subplots(4,4)
fig = plt.figure(figsize=(12, 8))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for row in range(4):
    for column in range(4):
        axis = axes[row][column]
        axis.get_xaxis().set_ticks([])
        axis.get_yaxis().set_ticks([])
        axis.imshow(activations[0][row*10+column])
plt.show()
```

5-37 20번째 계층에 대한 특성 맵

```
result = LayerActivations(model.features, 20)  //20번째 Conv2d 특성 맵 확인

model(img)
activations = result.features
```

5-38 특성 맵 확인

```
fig, axes = plt.subplots(4,4)
fig = plt.figure(figsize=(12, 8))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for row in range(4):
    for column in range(4):
        axis = axes[row][column]
        axis.get_xaxis().set_ticks([])
        axis.get_yaxis().set_ticks([])
        axis.imshow(activations[0][row*10+column])
plt.show()
```

5-39 40번째 계층에 대한 특성 맵

```
result = LayerActivations(model.features, 40)  //40번째 Conv2d 특성 맵 확인

model(img)
activations = result.features
```

5-40 특성 맵 확인

```
fig, axes = plt.subplots(4,4)
fig = plt.figure(figsize=(12, 8))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for row in range(4):
```

```
    for column in range(4):
        axis = axes[row][column]
        axis.get_xaxis().set_ticks([])
        axis.get_yaxis().set_ticks([])
        axis.imshow(activations[0][row*10+column])
plt.show()
```

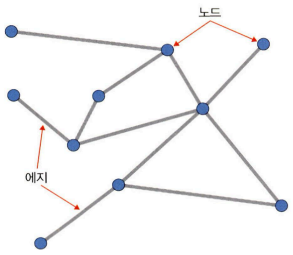
5.5 그래프 합성곱 네트워크

그래프 합성곱 네트워크(Graph Convolutional Network) : 그래프 데이터를 위한 신경망

5.5.1 그래프란

그래프 : 방향성이 있거나 없는 edge로 연결된 node의 집합

- 구성 요소 : **node(vertex)**, **edge**
  - 노드 : 원소를 의미
    에지 : 결합 방법(single, double, triple, aromatic 등)을 의미



5.5.2 그래프 신경망

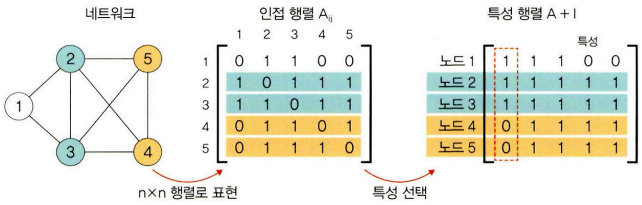
그래프 신경망(Graph Neural Network, GNN) : 그래프 구조에서 사용하는 신경망

1단계 : 인접 행렬(adjacency matrix)

- 노드 n개를 nxn 행렬로 표현
- 컴퓨터가 이해하기 쉽게 그래프로 표현하는 과정

2단계 : 특성 행렬(feature matrix)

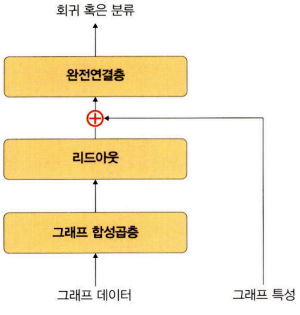
- 단위 행렬을 적용해 각 입력 데이터에서 이용할 특성 선택



5.5.3 그래프 합성곱 네트워크

그래프 합성곱 네트워크(Graph Convolutional Network, GCN) : 이미지에 대한 합성곱을 그래프 데이터로 확장한 알고리즘

- 구조 :



- **리드아웃(Readout)** : 특성 행렬을 하나의 벡터로 변환하는 함수

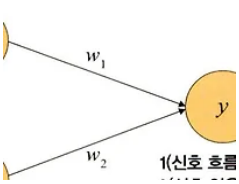
- **그래프 합성곱층(Graph Convolutoinal Layer)** : 가장 중요 !!
  - 데이터가 행렬 형태의 데이터로 변환되어 딥러닝 알고리즘을 적용할 수 있기 때문

공감

'EURON' 카테고리의 다른 글

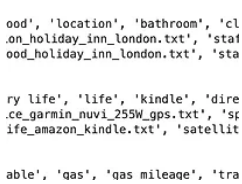
[4장] 딥러닝 시작 (0)	2024.09.30
9장 추천 시스템 (0)	2024.06.23
8장 텍스트 분석 Part.2 (0)	2024.06.17
8.3 Bag of Words - BOW (0)	2024.05.27
7장 군집화 (0)	2024.05.20

'EURON' Related Articles

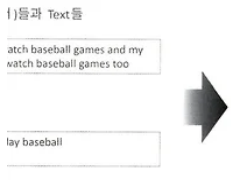


[4장] 딥러닝 시작

9장 추천 시스템



8장 텍스트 분석 Part.2



8.3 Bag of Words - BOW

HJP2000

Hodge 님의 블로그입니다.

댓글 0

Hodge

내용을 입력하세요.

등록

공지사항

최근 포스트

5장 합성곱 신경망 I

[4장] 딥러닝 시작

[프로그래머스] 배열 자르기

stream 공부 - continue

[Diet] Aug 14th, 2024 - 52.…

태그

더보기

검색

검색내용을 입력하세요.

전체 방문자

62

오늘	0
어제	0