

# 딥러닝 개념 이론

CNN & RNN 으로 알아보는  
딥러닝 구조

지도교수 : 김선만  
발표자 : AINC 202158031 소민섭



# Contents

## 01 딥러닝이란?

Neural NetWork 구조

## 02 CNN

CNN 구조 및 흐름

## 03 RNN

RNN 구조 및 흐름

---

## 04 ResNet CNN 실습

Resnet 활용한 CNN 이미지 실습



# 딥러닝이란?

## Neural NetWork 구조

### 퍼셉트론

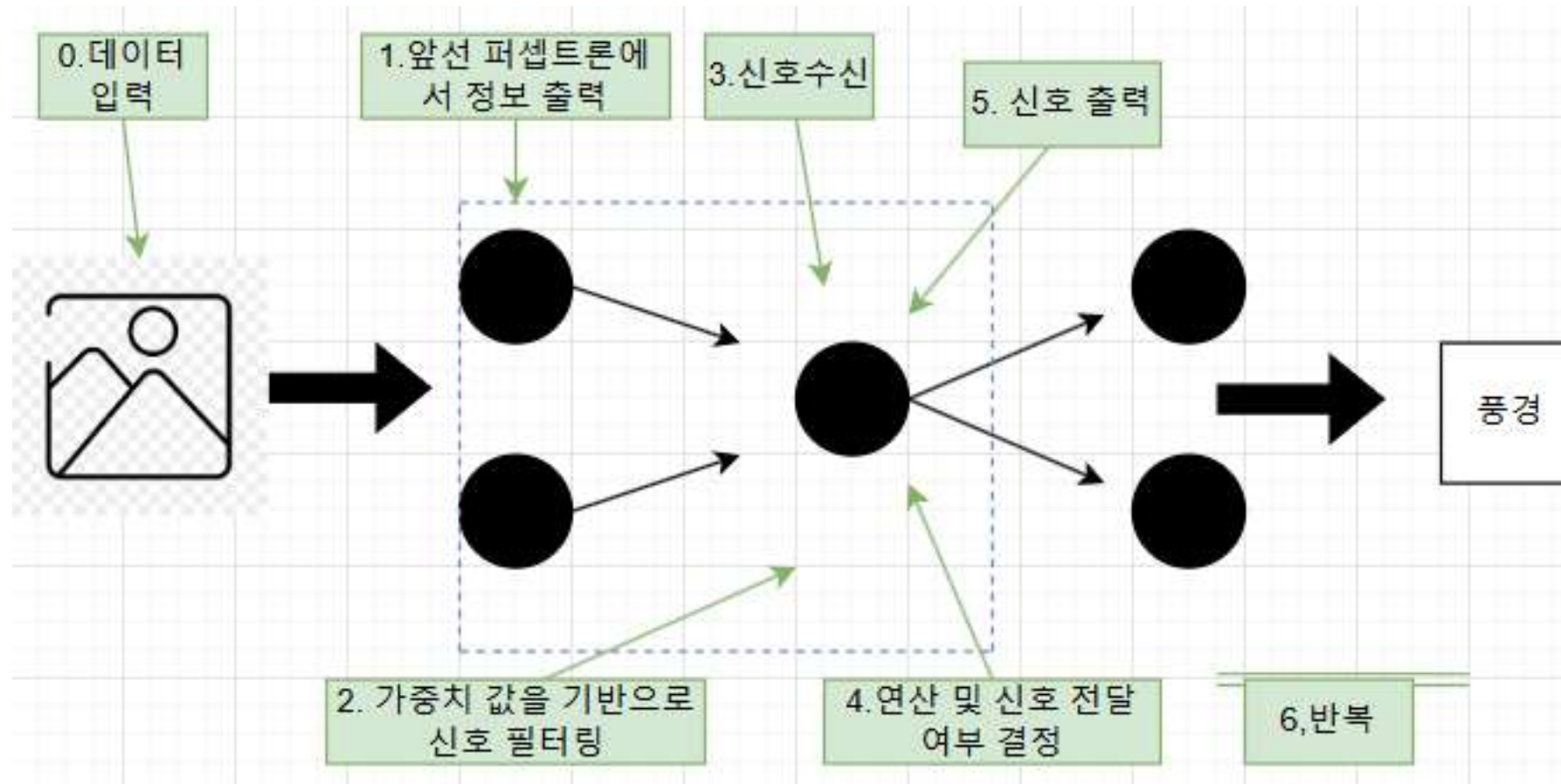
1957년 미국 신경 생물 학자 프랭크 로젠블랫이 제한  
-뉴런의 동작을 단순화해서 수학적으로 모델링

뉴런과 마찬가지로 다수의 입력을 바탕으로 하나의 결과를 출력

앞선 신호들로부터 정보를 얻고 정제해 후속 전파

### 딥러닝 모델의 다양한 문제 적용

1. 퍼셉트론을 다양하게 배치해 데이터 특성에 맞는 구조를 설계 가능
2. 복잡한 데이터의 패턴을 익힐 수 있도록 모델의 크기 조절 가능
3. 비정형 데이터인 이미지, 텍스트 오디오 등의 넓은 영역의 데이터를 다룰 수 있음

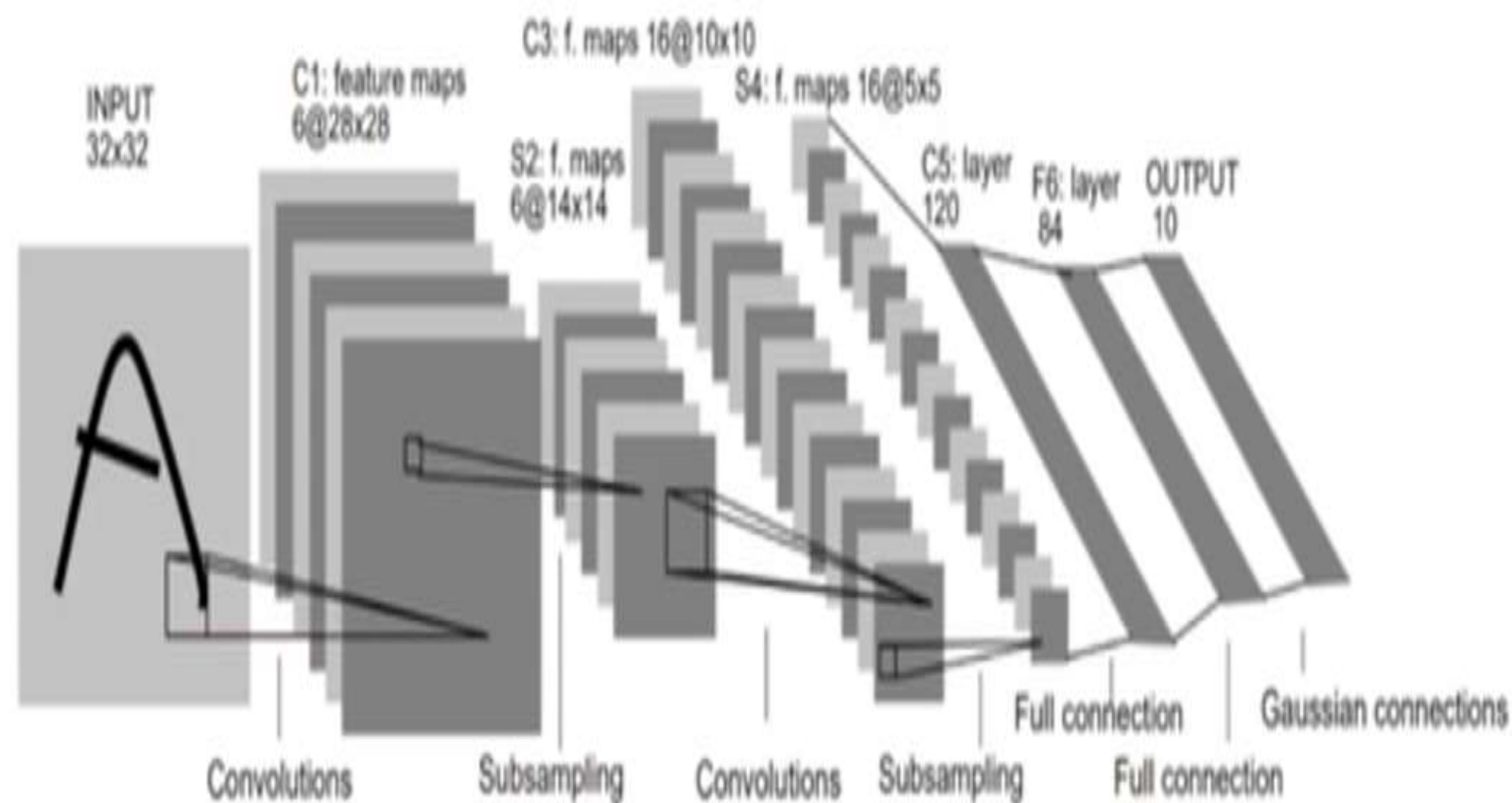


뉴런 -> 퍼셉트론(원)  
시냅스 -> 가중치(->)



# CNN

## CNN의 구조 및 흐름



### CNN이란?

사람의 시각 정보 처리 방법을 모방한 이미지를 처리하는 딥러닝 모델

사람

1. 분석 단위를 설정 후 **정보를 추출**
2. **주변 정보를 통합**해 차츰 **상위 개념을 구성**
3. 목적하는 상위 개념에 도달할 때까지 **반복** → 안될 시 다시 1.

CNN

#### 1. Convolutional Filter

(이미지 안에 있는 정보처리 -> 정보를 뽑아냄)

#### 2. Pooling

(만들어진 정보가 주변 정보와 모여서 상위 정보를 만듦)

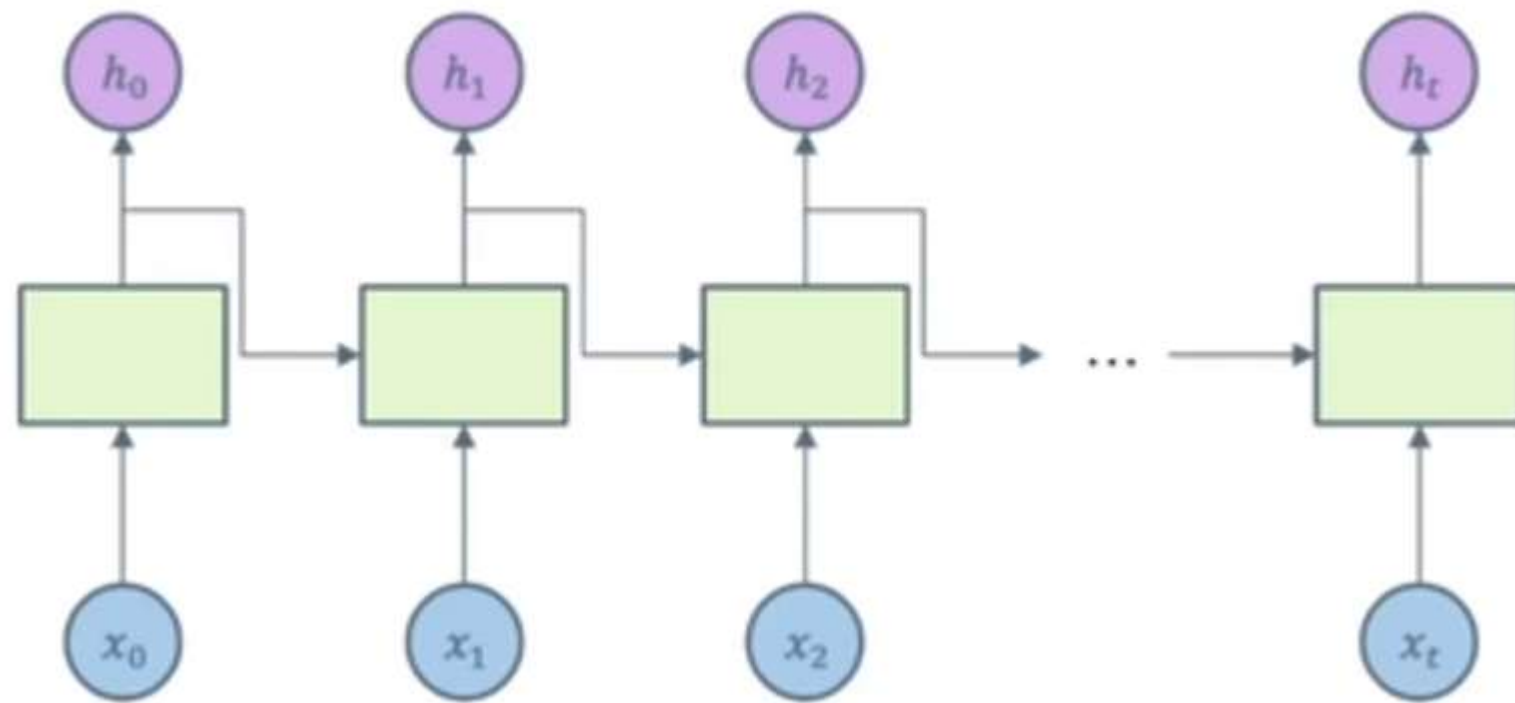
#### 3. 반복

(정보처리 -> 의미 추출 과정 반복) x? = 최종적 결과물



# RNN

## RNN의 구조 및 흐름



$x$  = 정보  
 $h$  = hidden state = 기억

### RNN이란?

텍스트와 같은 순차 데이터를 처리하기 위해 고안된 딥러닝 모델

사람

1. 문장의 앞에서부터 **한 단어 씩 읽음**
2. 새로운 단어가 들어오면 이를 **읽고 이해**
3. **앞서 해석해 만들어낸 기억 정보**에 정보를 추가해 **기억 업데이트**
4. 문장이 끝나는 시점까지 위 과정 **되풀이**

RNN

1. 문장의 앞에서 부터 **한 단어씩 입력** 받음
2. 새로운 단어가 들어오면 이를 **처리해서 정보 추출**
3. **앞서 해석해 만들어낸 정보 덩어리**에 추출한 정보를 추가하여 **정보를 업데이트**
4. 문장이 끝나는 시점 까지 위 과정을 **되풀이(Recurrent)**



# ResNet을 활용한 CNN 실습

## ResNet을 활용한 CNN 이미지 실습

### Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun  
Microsoft Research  
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

#### Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions<sup>1</sup>, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

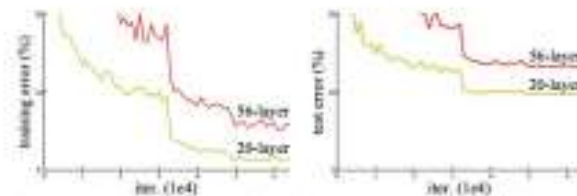


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [1, 9], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a degradation problem has been exposed: with the network

### ResNet이란?

2015년에 제한된 딥러닝 모델

사람의 이미지 인식 능력을 넘어선 최초의 이미지 인식 모델

모델의 깊이인 Layer에 따라 18, 34, 50, 101, 152 이라는 5가지 종류가 존재

- 깊이가 얇을수록 성능이 낮지만 빠르게 결과를 얻을 수 있음

- 깊이가 깊을수록 성능이 높아지지만 결과를 도출하기 위한 시간이 오래 걸림

입력으로 하나의 이미지를 제공하고 해당 이미지의 종류(클래스)를 예측

일반적으로 선택 가능한 모든 클래스에 대한 정답 점수값을 예측

최종 결과는 그 점수 중 가장 큰 값을 선택하면서 얻을 수 있음

이 점수들을 활용해 확률 값을 취할 수 있음





# ResNet을 활용한 CNN 실습

## ResNet을 활용한 CNN 이미지 실습

```
import re
import torchvision.models as models

# 모델 선택 및 로드
def get_model_and_trans(model_name):
    if model_name not in ['resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152']:
        raise ValueError("model name을 확인해주세요. 옵션 : 'resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152'")

    weight_name = 'ResNet' + re.findall(r'\d+', model_name)[0] + '_Weights'
    weights = getattr(models, weight_name).DEFAULT

    transforms = weights.transforms()

    model = getattr(models, model_name)(weights=weights)
    model.eval()

    meta_data = weights.meta
    return model, transforms, meta_data

# 모델 선택
model_name='resnet18'

# 모델과 그에 맞는 전처리 과정을 불러오기
myModel, transform, meta_data = get_model_and_trans(model_name)

myModel
```



<-대상 이미지

1. 웹 상 이미지 링크 복사 후 url 변수에 할당
2. 모델 선택 및 로드
3. 모델 선택: resnet18
4. 모델에 맞는 전처리 과정 불러오기



# ResNet을 활용한 CNN 실습

## ResNet을 활용한 CNN 이미지 실습

```
import torch

# 이미지 전처리 진행
trans_image = transform(image)
trans_image = torch.unsqueeze(trans_image, 0)

def imshow_from_tensor(tensor):
    image = tensor.cpu().clone() # 텐서 복제
    image = image.squeeze(0) # 배치 차원 제거
    image = image.permute(1, 2, 0) # 차원 재배열
    image = image.numpy()

    plt.imshow(image)
    plt.axis('off')
    plt.show()

# 전처리 후 원본 이미지 확인
imshow_from_tensor(trans_image)

# 모델에 입력하고 결과를 출력
output = myModel(trans_image)

# 1000개의 결과값을 출력
# 최고 값 = 결과 값
print(output)

import torch.nn.functional as F

# 출력 결과로 출력 결과 도출
conf, predicted = F.softmax(output, dim=1).max(1)
total_class = meta_data["categories"]
cls = total_class[predicted.item()]

print(f'입력 이미지는 {conf.item()*100:.2f} % 의 확률로 {cls}.')
```

```
0.3998e+00, -7.0948e-01, 2.1679e+00, -2.5169e+00, -3.6810e-01,
2.2207e+00, 1.1386e+00, 3.0967e+00, 2.4795e+00, -1.0159e+00,
-2.1657e+00, 1.8209e-01, 1.5645e+00, -1.3423e+00, 1.4347e-01,
2.4364e-02, -1.4554e+00, -3.3700e+00, -4.2014e+00, 5.0105e-01,
7.3332e+00, 4.3855e-01, 2.1449e+00, 2.3168e+00, -8.9934e-02,
-1.5274e+00, 9.4997e-01, -9.1652e-01, 1.9327e+00, -6.2025e-01,
-2.3223e+00, -5.1564e-01, -1.1356e-02, -5.9758e-01, -1.2118e+00,
-2.9022e+00, 4.2615e-01, -1.0516e+00, 3.9657e-02, 3.0237e+00,
1.5071e-01, 8.1131e-01, 2.0054e-01, 2.9405e+00, -1.9742e+00,
2.1375e-01, 2.0401e+00, -3.0035e+00, -1.8325e+00, 1.0011e+00,
-7.6934e-01, -9.5365e-01, -5.6239e-01, -6.5971e-01, -4.4625e-01,
1.4457e+00, 1.7336e-01, 4.8633e-01, 1.2714e+00, -6.7656e-01,
-1.2574e+00, 1.7139e+00, -1.9945e+00, -1.4204e-01, -1.8310e+00,
2.6089e+00, 7.5221e-01, 7.6382e+00, -1.0435e+00, -3.2599e+00,
1.9073e+00, -5.8040e-01, -2.3596e+00, -1.8832e+00, -2.1613e+00,
-1.4205e+00, 2.6575e+00, 1.6115e+00, -2.0900e+00, -3.7304e-01,
1.6754e+00, -1.0707e+00, 8.8524e-01, 1.1772e+00, 6.8744e-01,
2.9972e+00, -5.3199e-02, 3.2496e+00, -1.6345e+00, -4.7160e+00,
-1.2256e+00, 2.3969e+00, -1.0332e+00, -3.7425e+00, 2.8315e+00,
2.6440e+00, -1.0607e+00, 3.5118e+00, 5.3295e-01, -3.5170e-01,
1.1479e+00, -1.8540e+00, -2.3250e+00, -2.4952e+00, -2.6732e+00,
-1.9617e-01, 1.7392e+00, 1.4150e-01, 4.2144e-01, -3.9027e+00,
-5.9125e-01, -9.3241e-01, 4.1665e-01, 2.6299e+00, 1.8835e+00,
-9.2205e-01, -2.7109e-01, -4.3700e-01, 7.2158e-01, 1.8575e+00,
-7.2230e-01, -5.4526e-01, 8.8742e-01, 1.4544e+00, -5.1805e-01,
5.0791e-01, 2.6642e+00, 1.8234e+00, -2.0660e+00, -1.2696e+00,
8.8210e-01, -3.6660e-01, -2.0249e+00, -1.2021e+00, -1.3006e+00,
-3.1467e+00, -1.0078e+00, -4.1447e+00, -2.2853e-01, -1.3739e+00,
-0.0621e-01, -4.4076e-01, -4.9994e-01, -2.2914e+00, 6.9939e-01,
8.0377e-01, 4.3915e-01, -1.7146e+00, -1.9461e+00, -5.2120e-01,
-8.0130e-01, 7.8606e-01, 1.2123e+00, 8.2252e-01, 2.7527e+00,
-1.0839e+00, -3.9002e+00, -1.9220e+00, -8.5891e-02, -1.2098e+00,
-1.3113e+00, -8.3505e-01, 2.1806e+00, 7.0621e-01, 1.0374e+00,
-2.2586e+00, -3.0030e-01, -3.2046e+00, -1.7094e+00, -2.6100e+00,
1.0950e+00, 3.9315e-01, -4.6189e+00, 1.1931e-01, -1.0136e+00,
-2.5084e+00, 1.5447e-01, 1.0193e-01, -1.5166e+00, -3.8398e-01,
-2.7674e+00, -4.2367e-01, -2.2637e+00, -1.7593e-01, 5.9246e-01,
-9.8523e-01, 8.5582e-01, 4.9665e-01, -5.0251e-01, -3.5420e+00,
2.9178e+00, 4.6066e-01, 4.5284e-01, 1.2490e+00, 3.2210e-01,
-2.4059e+00, 1.2030e+00, -2.1466e+00, -2.4265e+00, -2.3045e+00,
-1.9125e+00, -4.8824e+00, 1.6240e+00, -2.1554e+00, -2.7561e+00,
-3.6475e+00, -1.3360e+00, 2.0436e+00, -4.1196e-01, -1.0434e-01,
2.0536e+00, 2.3462e+00, -9.7570e-01, 1.9940e+00, 1.6500e+00]
```

최종 결과 입력 이미지는  
94.47 % 의 확률로  
golden retriever.

### 이미지 전처리 후 출력 결과 후처리

ResNet 모델은 입력된 이미지가 1,000개의 이미지 종류  
중 어떤 부류에 속하는지를 판단하는 모델

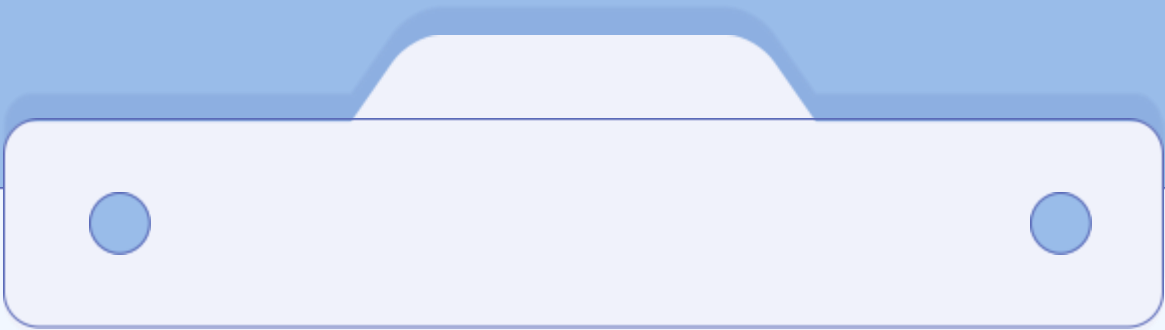
모델의 출력은 1,000의 크기를 갖는 vector • 이 vector의  
값 중, 제일 큰 값을 갖는 위치의 class 가 실제 정답

- 어떤 위치가 어떤 class 인지 meta data의 형태로  
weights 객체에 들어있음

- 출력으로 나온 결과 값을 이용해 딥러닝 모델이  
얼마나 큰 정확도로 입력 이미지의 종류에 확신을 갖고  
있는지 계산할 수 있음







# 감사합니다

Q & A

지도교수 : 김선만  
발표자 : AINC 202158031 소민섭