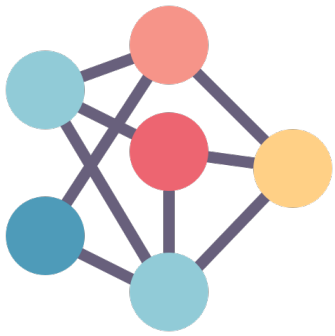

Hanbat National University

Course: Machine Learning

January 8, 2023

Author	*Student ID*
Jonghyeok Kim	20201967

Machine Learning (1/8)

Jonghyeok Kim

January 8, 2023

Contents

1 Deep Learning

1.1 Pretrained networks

1.1.1 Obtaining a pretrained network for image recognition

우선은 AlexNet(Image recognition을 위한 초창기 획기적인 네트워크)을 먼저 Load하고, Residual Network인 ResNet(2015년에 다른 Networks들 중에 ImageNet 분류(Classification), 검출(Detection), Localization 대회에서 우승했음.)을 Load해보자.

사전에 이미 훈련된 모델들은 **torchvision.models**에서 발견될 수 있다.

```
from torchvision import models

dir(models)
```

위 구문의 출력결과는 다음과 같다.

```
Output exceeds the size limit. Open the full output data in a text editor
['AlexNet',
 'AlexNet_Weights',
 'ConvNeXt',
 'ConvNeXt_Base_Weights',
 'ConvNeXt_Large_Weights',
 'ConvNeXt_Small_Weights',
 'ConvNeXt_Tiny_Weights',
 'DenseNet',
 'DenseNet121_Weights',
 'DenseNet161_Weights',
 'DenseNet169_Weights',
 'DenseNet201_Weights',
 'EfficientNet',
 'EfficientNet_B0_Weights',
 'EfficientNet_B1_Weights',
 'EfficientNet_B2_Weights',
 'EfficientNet_B3_Weights',
 'EfficientNet_B4_Weights',
 'EfficientNet_B5_Weights',
```

```
 'EfficientNet_B6_Weights',
 'EfficientNet_B7_Weights',
 'EfficientNet_V2_L_Weights',
 'EfficientNet_V2_M_Weights',
 'EfficientNet_V2_S_Weights',
 'GoogLeNet',
...
 'vit_h_14',
 'vit_l_16',
 'vit_l_32',
 'wide_resnet101_2',
 'wide_resnet50_2']
```

위의 출력결과에서 대문자로 표시되는 것들은 많은 인기있는 모델들을 구현하고 있는 파이썬의 클래스(Classes)들을 나타낸다.

위에서 출력된 모델들은 서로 Architecture가 다르다. 다시 말해서, 입력(Input)과 출력(Output) 사이의 연산(Operations)들의 순서가 서로 다르다는 의미이다.

반면에, 소문자로 표시되는 것들은 편의(Convenience)를 위한 함수(Function)인데, 이러한 함수는 그들의 클래스에서 인스턴스화되어진(instantiated) 모델들을 반환해준다. (때때로 서로 다른 파라미터들을 가진다.)

예를 들어, **resnet101**은 101개의 레이어(Layers)를 가진 **ResNet**의 인스턴스를 반환해준다.

마찬가지의 원리에 의하여, **resnet18**은 18개의 레이어를 가진 **ResNet**을 반환해주는 식이다.

이제 AlexNet으로 관심을 돌려보자.

1.1.2 AlexNet

AlexNet은 2012 ILSVRC에서 15.4%로 상위 5개의 테스트 에러율(Test error rate)(즉, 올바른 레이블(Label)이 상위 5개 예측에 있어야 함.)의 큰 격차로 우승했다.

2위는 Deep network에 기반하지 않았으며, 26.2%의 테스트 에러율을 보여주었다.

이는 컴퓨터 비전(Computer vision)의 역사에 기록되는 순간이었다.

이로부터 컴퓨터 비전 커뮤니티(Community)는 비전 업무(Vision tasks)에 있어서 딥러닝(Deep learning)의 진가를 깨닫게 되었다.

다음은 AlexNet의 구조를 보인 그림이다.

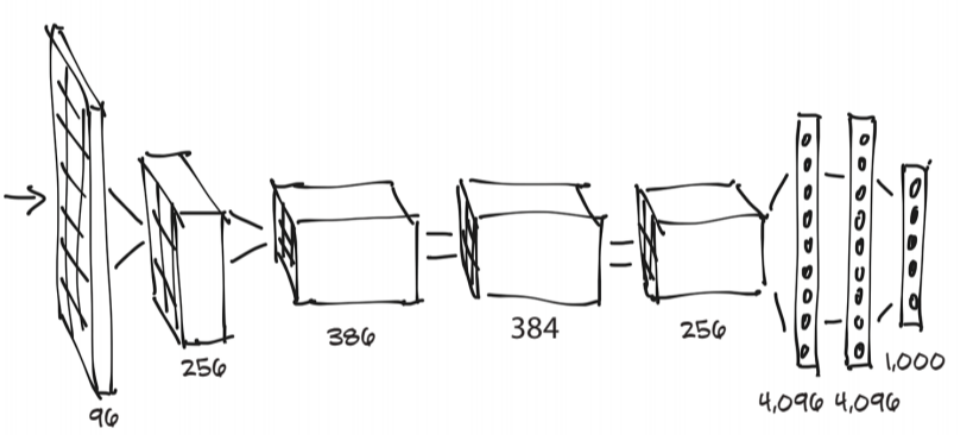

Figure 1: The AlexNet Architecture

왼쪽에서 입력 이미지(Input images)들이 들어오고, 5개의 필터들의 스택(Five stacks of filters)들을 통과하며, 각각의 필터들은 많은 출력 이미지(Output images)들을 만들어낸다.

각 필터를 통과한 이후에 그 이미지들은 그림 상에 표시된 것과 같이 크기가 줄어들게 된다.

마지막 필터들의 스택에 의해 만들어진 이미지들은 4,096개의 요소(Element)를 가지는 1D 벡터(Vector)로 배열되고, 1,000개의 출력확률(Output probabilities)을 만들도록 분류된다. (각 출력 클래스(Output class)에 하나의 출력확률이 대응된다.)

입력 이미지에 대해 AlexNet 아키텍처(Architecture)를 실행(Run)하기 위해, AlexNet 클래스의 인스턴스를 다음의 코드로 생성할 수 있다.

```
alexnet = models.AlexNet()
```

여기서, **alexnet**은 **AlexNet** 아키텍처를 실행할 수 있는 객체(Object)이다.

당분간은 **AlexNet**이 단지 함수(Function)처럼 호출될 수 있는 불투명한 객체라고 이해하자.

정확한 크기의 입력 데이터를 가진 **alexnet**을 제공함으로써, 그 네트워크(Network)를 통해 Forward pass를 실행할 것이다.

즉, 입력 데이터는 뉴런(Neurons)들의 첫 번째 집합(Set)을 통해 실행될 것이고, 그 출력 데이터들은 다음 뉴런들의 집합으로 들어갈 것이다. 이러한 방식은 마지막 출력때까지 이어질 것이다.

실질적으로 말하자면, 올바른 자료형(type)의 **input** 객체(Object)를 가진다고 가정하면, 다음과 같이 Forward pass를 실행할 수 있다.

```
output = alexnet(input)
```

그러나, 만약 위와 같은 코드를 작성하였다면, 데이터를 전체 네트워크에 통과시키면, 쓰레기(Garbage) 값이 도출될 것이다.

이는 지금 현재 네트워크(Network)가 초기화되지 않았기(Uninitialized) 때문이다. 즉, 그것의 가중치(Weights)(Network에 의해 입력 데이터들이 더해지고 곱해지는 숫자(Numbers)가 어떠한 것에 대해서도 훈련되지 않았다. 그러므로 그 네트워크는 빈(Blank)(또는 오히려 무작위한(Random)) Slate이다.

그러므로 이것을 처음부터 훈련시키거나 사전에 훈련되어진 가중치를 불러올 필요가 있다.

1.1.3 ResNet

다음으로 살펴볼 아키텍처는 ResNet이다.

이제 **resnet101** 함수를 사용해서 101개 레이어(Layer)인 Convolutional Neural Network를 인스턴스화할 것이다.

사물을 원근감 있게 보기 위해(Just to put things in perspective) 즉, 전체 나무를 보기 위해, 2015년 Residual networks의 출현 이전에 101개의 층에서 안정적인 학습을 진행하는 것이 극도로 어렵다고 여겨져왔었다.

Residual networks는 그것을 가능하게 하는 트릭(Trick)을 사용했고, 그렇게 함으로써 그 해 한번에 여러 벤치마크(Benchmarks)를 이겼다.

다음의 코드를 통해 ResNet의 인스턴스(Instance)를 생성하자.

```
resnet = models.resnet101(pretrained=True)
```

위에서 확인할 수 있듯이, **resnet101**의 인자로 어떤 Argument를 전달하였는데, 이는 그 함수(resnet101)가 120만개의 이미지들과 1,000개의 범주(Categories)의 ImageNet 데이터셋(Dataset)으로 학습한 **resnet101**의 가중치들을 다운로드하도록 지시할 것이다.

1.1.4 Ready, set, almost run

resnet101이 어떻게 생겼는지 확인해보자.

이는 반환된 모델의 값을 출력함으로써 확인할 수 있는데, 위의 Figure 1처럼 그림이 아닌, 글의 형태로 제시되며, 네트워크의 구조(Structure)에 대한 세부사항(Details)들을 제시한다.

```
resnet
```

위 구문의 출력결과는 다음과 같다.

```
Output exceeds the size limit. Open the full output data in a text editor
ResNet(
```

```
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
...
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

위의 출력결과에서 확인할 수 있는 것은 매 줄마다 표시된 **Modules**인데, 이는 파이썬의 모듈(Modules)과 공통점이 존재하지 않는다. 즉, 서로 다른 연산이다. 단지, 여기서의 **Modules**는 Neural 네트워크(Network)를

구성하고 있는 블럭(Blocks)일 뿐이다. 이러한 **Modules**들은 다른 딥러닝 프레임워크(Deep learning frameworks)에서는 **layers**라고도 불린다.

또한, 많은 **Bottleneck** 모듈들도 확인할 수 있는데, 지금 현재 출력결과에 표시되진 않았으나, 총 101개의 **Bottleneck** 모듈들이 존재한다. 이것은 Convolutions과 다른 모듈들을 포함하고 있다.

이것은 컴퓨터 비전(Computer Vision)에 대한 전형적인 Deep neural network의 구조를 띤다. 즉, 1,000개의 출력 클래스(Classes)들(out_features)의 각각에 대한 점수(Scores)들을 제공하는 어떤 레이어 (fc)로 끝나면서 동시에 다소 순차적인 필터(Filters)들과 비선형 함수들(Non-linear functions)의 Cascade로 구성되어 있다.

resnet 변수(Variable)는 함수(Function)와 같이 호출될 수 있고, 입력으로 하나 이상의 이미지들을 취하고, 1,000개의 ImageNet 클래스들의 각각에 대한 동일한 수의 점수들을 제공한다.

그러나, 이것을 하기 전에 먼저 입력 이미지들을 전처리해야하는데, 이는 올바른 크기를 가지고, 그들의 값들(색깔들)이 대략 같은 범위 안에 놓이도록 하기 위함이다.

이러한 전처리를 하기 위해서, **torchvision** 모듈은 **transforms**를 제공하는데, 이것은 빠르게 기본적인 전처리 함수들의 Pipelines을 구축하도록 해준다.

```
from torchvision import transforms


preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
)])
```

위의 코드에서 **preprocess** 함수를 다음과 같이 정의했다.

입력 이미지를 256×256으로 그 크기를 조정하고, 중앙(Center) 주변으로 224×224만큼 이미지를 자르며(Crop), 이를 Tensor(PyTorch의 다차원 배열(Multidimensional array)로, 이 경우 3D 배열인데, 색상(Color), 높이(Height), 너비(Width) 정보를 지닌다.)로 변환한 다음, 정의된 평균(Means)과 표준편차(Standard deviations)를 지니도록 그것의 RGB(Red, Green, Blue) 구성요소(Components)를 정규화(Normalize)하라.

만약 네트워크가 의미있는 답(Meaningful answer)을 도출하게끔 하고싶다면, 이러한 것들이 훈련동안 네트워크에 제시했던 것과 일치할 필요가 있다.

이제 다음과 같은 리트리버(Retriever) 사진에 대해,

Figure 2: Bobby, our very special input image

전처리한 다음, ResNet이 이 이미지에 대해 무엇이라고 생각하는지 확인해보자.

먼저 **Pillow**(파이썬에서 이미지 조작을 위한 모듈)를 사용해서 로컬 파일시스템(Local filesystem)에서 해당 이미지를 불러오는 것부터 시작하자.

```
from PIL import Image


img = Image.open("../data/p1ch2/bobby.jpg")
```

불러들인 이미지가 제대로 불러들여졌는지를 확인하기 위해 Jupyter Notebook에서 다음의 구문을 입력하여 Inline으로 해당 그림을 확인할 수 있다.

```
img
```

Jupyter Notebook을 사용하고 있지 않다면, 해당 이미지를 표시하기 위해 어떤 뷰어(Viewer)를 가진 창(Windows)을 띄워주는 **show** 메소드(Method)를 호출할 수도 있다.

```
img.show()
```

다음으로, 불러들인 이미지를 위에서 정의한 전처리 Pipeline을 통과시킬 수 있다.

```
img_t = preprocess(img)
```

통과하는 동안, 해당 이미지는 모양이 바뀌고(Reshape), 잘라지며(Crop), Tensor로 변환(Transform) 후, 마지막으로 해당 Tensor를 정규화(Normalize)하게 된다.

```
import torch


batch_t = torch.unsqueeze(img_t, 0)
```

1.1.5 Run!

새로운 데이터에 대해 정의된 모델을 실행하는 과정을 Deep learning Circle에서는 **inference**라 한다.

inference를 하기 위해서는 그 네트워크를 **eval** 모드(Mode)로 설정할 필요가 있다.

```
resnet.eval()
```

위의 출력결과는 다음과 같다.

```
Output exceeds the size limit. Open the full output data in a text editor
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
```

```
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
...
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

일부 사전에 훈련된 모델들에서 **batch normalization**과 **dropout**을 하지 않게 되면, 내부적으로 돌아가는 방식에 의해 의미있는 답을 얻지 못할 것이다.

위에서 **eval**을 설정하였기 때문에 **inference**할 준비가 되었다.

```
out = resnet(batch_t)
```

```
out
```

위 구문들의 출력결과는 다음과 같다.

```
Output exceeds the size limit. Open the full output data in a text editor
tensor([[-3.4803e+00, -1.6618e+00, -2.4515e+00, -3.2662e+00, -3.2466e+00,
         -1.3611e+00, -2.0465e+00, -2.5112e+00, -1.3043e+00, -2.8900e+00,
         -1.6862e+00, -1.3055e+00, -2.6129e+00, -2.9645e+00, -2.4300e+00,
         -2.8143e+00, -3.3019e+00, -7.9404e-01, -6.5183e-01, -1.2308e+00,
         -3.0193e+00, -3.9457e+00, -2.2675e+00, -1.0811e+00, -1.0232e+00,
         -1.0442e+00, -3.0918e+00, -2.4613e+00, -2.1964e+00, -3.2354e+00,
         -3.3013e+00, -1.8553e+00, -2.0921e+00, -2.1327e+00, -1.9102e+00,
         -3.2403e+00, -1.1396e+00, -1.0925e+00, -1.2186e+00, -9.3332e-01,
         -4.5093e-01, -1.5489e+00,  1.4161e+00,  1.0871e-01, -1.8442e+00,
         -1.4806e+00,  9.6227e-01, -9.9456e-01, -3.0060e+00, -2.7384e+00,
         -2.5798e+00, -2.0666e+00, -1.8022e+00, -1.9328e+00, -1.7726e+00,
```

```
        -1.3041e+00, -4.5848e-01, -2.0537e+00, -3.2804e+00, -5.0451e-01,
        -3.8174e-01, -1.1147e+00, -7.3998e-01, -1.4299e+00, -1.4883e+00,
        -2.1073e+00, -1.7373e+00, -4.0412e-01, -1.9374e+00, -1.4862e+00,
        -1.2102e+00, -1.3223e+00, -1.0832e+00,  7.9208e-02, -4.1344e-01,
        -2.7477e-01, -8.5398e-01,  6.0364e-01, -8.9196e-01,  1.4761e+00,
        -2.6427e+00, -3.6478e+00, -2.7066e-01, -1.2360e-01, -2.2445e+00,
        -2.3425e+00, -1.4430e+00,  2.5264e-01, -1.0588e+00, -2.8812e+00,
        -2.5145e+00, -2.2579e+00,  4.1647e-01, -1.3463e+00, -1.6450e-02,
        -2.8798e+00, -5.5658e-01, -1.3859e+00, -2.9352e+00, -1.8880e+00,
        -4.2244e+00, -2.9742e+00, -2.0298e+00, -2.3869e+00, -2.7324e+00,
        -3.9905e+00, -3.6113e+00, -5.4423e-01, -1.0291e+00, -1.8998e+00,
        -3.5611e+00, -1.5031e+00,  1.0660e+00, -7.1587e-01, -7.2612e-01,
        -2.2173e+00, -2.2616e+00, -5.9990e-01, -1.4349e+00, -2.5965e+00,
        -3.9844e+00, -9.4164e-01, -5.3675e-01, -8.4138e-01, -1.1660e+00,
...
        -4.4594e+00,  5.5604e-01, -1.3140e+00, -3.8407e+00, -7.5988e-01,
        -5.7457e-01, -2.5448e+00,  2.3831e+00,  6.1368e-01,  4.8295e-01,
         2.8674e+00, -3.7442e+00,  1.5085e+00, -3.2500e+00, -2.4894e+00,
        -3.3541e-01,  1.2856e-01, -1.1355e+00,  3.3969e+00,  4.4584e+00]],
       grad_fn=<AddmmBackward0>)
```

위 출력결과를 도출하는 데 있어, 총 4450만 개의 파라미터(Parameters)들이 포함된 연산(Operations)들의 집합(Set)이 ImageNet 클래스마다 하나씩 하여 총 1,000개의 점수에 대한 벡터(Vector)가 만들어지며 수행되었다.

그런데 이러한 작업은 오랜 시간이 소요되지 않았다.

이제 가장 높은 점수를 받은 클래스의 레이블(Label)을 찾을 필요가 있는데, 이것은 본 모델(Model)이 해당 이미지를 어떻게 보았는지를 시사한다. 만약 그 레이블이 인간이 그 이미지를 설명하는 것과 일치한다면, 좋은 결과인 것이다. 그런데 만약 그러한 결과가 도출되지 않았다면, 훈련동안 무엇인가가 잘못되었거나 입력 이미지가 모델이 그것을 적절히 처리할 수 있도록 모델이 기대하고 있는 것과 자못 다른 것과 같은 비슷한 문제(Issue)로 인해 발생했을 것이다.

예측된 레이블의 리스트(Lists)를 확인하기 위해 훈련동안 네트워크에 제시된 동일한 순서로 그 레이블을 목록화한 텍스트 파일(Text file)을 불러올 것이고, 그러고나서 그 네트워크에서 가장 높은 점수를 받은 인덱스(index)에 있는 레이블을 추출할 것이다.

이미지 인식(Image recognition)에 대한 거의 대다수의 모델들은 뒤에서 보게될 출력의 형태와 상당한 유사성(Similarity)을 지니고 있다.

ImageNet 데이터셋 클래스들에 대한 1,000개의 레이블을 포함하고 있는 파일을 불러들이자.

```
with open('../data/p1ch2/imagenet_classes.txt') as f:
    labels = [line.strip() for line in f.readlines()]
```

out Tensor에서 최대 점수에 상응하는 인덱스를 결정할 필요가 있다. 이는 PyTorch의 **max** 함수를 사용해서 할 수 있고, 그 결과, Tensor에서의 최대 값과 그 최대 값이 어디서 발생하고 있는지를 나타내는 인덱스도 함께 출력해준다.

```
_, index = torch.max(out, 1)
```

이제 인덱스를 사용하여 그 레이블에 접근할 수 있다.

여기서, 인덱스(Index)는 plain Python의 숫자(Number)가 아니지만, 하나의 요소(One-element)이고, 하나의 차원(One-dimension)만을 지닌 Tensor이다. (구체적으로, **tensor([207])**) 그러므로, **index[0]**과 같이 인덱스를 사용하여 레이블에서 실제의 수치적 값(Numerical value)을 얻을 필요가 있다.

또한, 얻은 출력결과를 그것들의 합계로 나누어 [0, 1]의 범위로 정규화(Normalize)해주는 **torch.nn.functional.softmax**를 사용할 수 있다. 이로한 처리과정을 통해 도출된 값은 본 모델이 그것의 예측에 대해 얼마나 자신(Confidence)이 있는지를 나타내는 것과 대략적으로 유사하다.

```
percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
labels[index[0]], percentage[index[0]].item()
```

위 구문들의 출력결과는 다음과 같다.

```
('golden retriever', 96.29335021972656)
```

위 출력결과가 시사하는 바는 본 모델이 해당 이미지를 골든 리트리버로 보고있음에 96%의 확신을 가진다는 의미이다.

이렇듯 모델이 클래스에 대한 점수를 제공하기 때문에, 2번째, 3번째 등에 해당하는 점수들도 역시 확인하는 것이 가능하다. 이를 위해 값들을 오름차순(Ascending order) 또는 내림차순(Descending order)로 정렬하고, 또한 원래 배열에서 정렬된 값들에 대한 인덱스값들도 제공해주는 **sort** 함수를 사용할 수 있다.

```
_, indices = torch.sort(out, descending=True)
[(labels[idx], percentage[idx].item()) for idx in indices[0][:5]]
```

위 구문들의 출력결과는 다음과 같다.

```
[('golden retriever', 96.29335021972656),
 ('Labrador retriever', 2.8081161975860596),
 ('cocker spaniel, English cocker spaniel, cocker', 0.2826734781265259),
 ('redbone', 0.20862983167171478),
 ('tennis ball', 0.11621581763029099)]
```

2 Conclusion

2.1 Conclusion

3 Acknowledgement

Appendix A

A References

[1] Luca Antiga Eli Stevens. *Code Files for Deep-learning with pytorch*. 2020. URL: `https://github.com/deep-learning-with-pytorch/dlwpt-code` (visited on 01/08/2023).

[2] Kuldeep Singh. *Linear Algebra: Step by Step*. Oxford University Press, 2013.