HM1

January 25, 2024

[ ]:
```
import torch
print(torch.__version__)
print(torch.cuda.is_available())
```

```
2.1.2
True
```

1 Exercise 1

[ ]:
```
import torch
from torchvision import transforms, models
import urllib.request
from torchvision import models
import matplotlib.pyplot as plt
from PIL import Image

# download image
url = 'https://cdn11.bigcommerce.com/s-ig5sr43nuo/images/stencil/1280x1280/
 ↪products/1375/5999/P4070245__50653.1683325700.jpg?c=2'
fpath = 'cup.jpg'
urllib.request.urlretrieve(url, fpath)

# open and plot image
img = Image.open('cup.jpg')
plt.imshow(img)

# download class names
url = "https://raw.githubusercontent.com/joe-papa/pytorch-book/main/files/
 ↪imagenet_class_labels.txt"
fpath = 'imagenet_class_labels.txt'
urllib.request.urlretrieve(url, fpath)

with open('imagenet_class_labels.txt') as f:
  classes = [line.strip() for line in f.readlines()]

# load model
model = models.vgg16(pretrained=True)
```

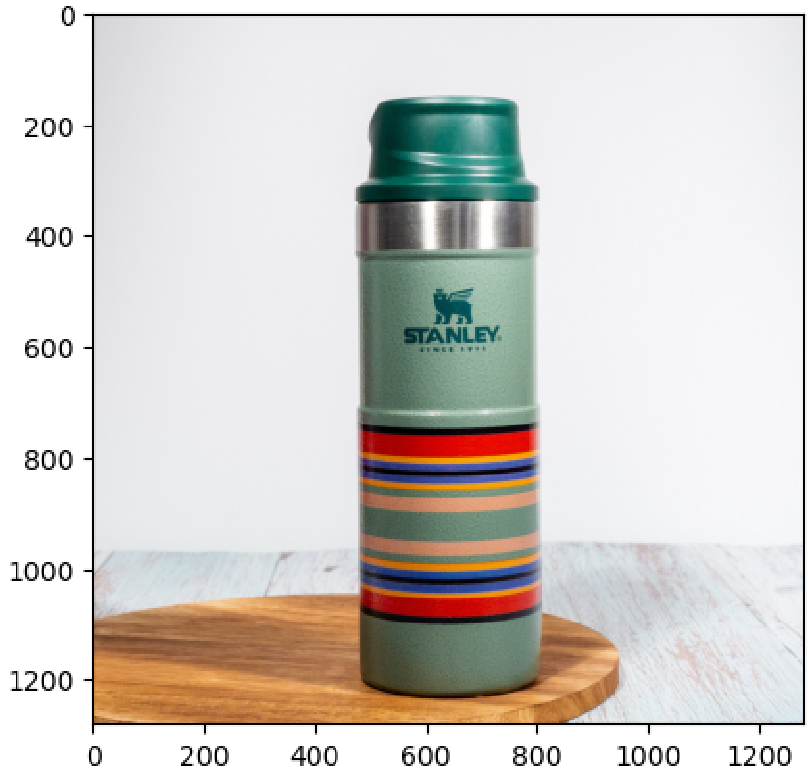
```python
# model = models.alexnet(pretrained=True)
# predict
transform = transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
  transforms.Normalize(
      mean=[0.485, 0.456, 0.406],
      std=[0.229, 0.224, 0.225])])

img_tensor = transform(img)
batch = torch.unsqueeze(img_tensor, 0)

device = "cuda" if torch.cuda.is_available() else "cpu"
model.eval()
model.to(device)
y = model(batch.to(device))

prob = torch.nn.functional.softmax(y, dim=1)[0] * 100
_, indices = torch.sort(y, descending=True)
for idx in indices[0][:5]:
  print(classes[idx], prob[idx].item())
```

```
898: 'water bottle', 33.40708923339844
585: 'hair spray', 16.637653350830078
631: 'lotion', 10.778732299804688
838: 'sunscreen, sunblock, sun blocker', 6.996231555938721
720: 'pill bottle', 4.400457382202148
```



For the alexnet:

- What is the classification result (label)?

  Lotion

- What are the similar labels in ImageNet?

  'lotion', 'cocktail shaker','pill bottle', 'water bottle', 'hair spray'

- What is the confidence of the classification result?

  25.575265884399414, 20.700424194335938, 15.21296215057373, 14.074629783630371, 7.729822158813477

For vgg16:

- What is the classification result (label)?

  water bottle

- What are the similar labels in ImageNet?

  'water bottle', 'hair spray', 'lotion', 'sunscreen, sunblock, sun blocker', 'pill bottle',

- What is the confidence of the classification result?

  33.40708923339844, 16.637653350830078, 10.778732299804688, 6.996231555938721, 4.400457382202148

Which model is more confident in its decision about your image? Do decisions differ?

VGG16 is more confident there is more confidence in the first prediction and has lower confidence in the other predictions.

What is the number of parameters you obtain for alexnet and vgg16, respectively?

[ ]:
```
from torchinfo import summary
summary(model, input_size=(16, 3, 224, 224), row_settings=("depth",␣
 ↪"ascii_only"))
```

[ ]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
VGG                                      [16, 1000]                --
+ Sequential: 1-1                        [16, 512, 7, 7]           --
|    + Conv2d: 2-1                       [16, 64, 224, 224]        1,792
|    + ReLU: 2-2                         [16, 64, 224, 224]        --
|    + Conv2d: 2-3                       [16, 64, 224, 224]        36,928
|    + ReLU: 2-4                         [16, 64, 224, 224]        --
|    + MaxPool2d: 2-5                    [16, 64, 112, 112]        --
|    + Conv2d: 2-6                       [16, 128, 112, 112]       73,856
|    + ReLU: 2-7                         [16, 128, 112, 112]       --
|    + Conv2d: 2-8                       [16, 128, 112, 112]       147,584
|    + ReLU: 2-9                         [16, 128, 112, 112]       --
|    + MaxPool2d: 2-10                   [16, 128, 56, 56]         --
|    + Conv2d: 2-11                      [16, 256, 56, 56]         295,168
|    + ReLU: 2-12                        [16, 256, 56, 56]         --
|    + Conv2d: 2-13                      [16, 256, 56, 56]         590,080
|    + ReLU: 2-14                        [16, 256, 56, 56]         --
|    + Conv2d: 2-15                      [16, 256, 56, 56]         590,080
|    + ReLU: 2-16                        [16, 256, 56, 56]         --
|    + MaxPool2d: 2-17                   [16, 256, 28, 28]         --
|    + Conv2d: 2-18                      [16, 512, 28, 28]         1,180,160
|    + ReLU: 2-19                        [16, 512, 28, 28]         --
|    + Conv2d: 2-20                      [16, 512, 28, 28]         2,359,808
|    + ReLU: 2-21                        [16, 512, 28, 28]         --
|    + Conv2d: 2-22                      [16, 512, 28, 28]         2,359,808
|    + ReLU: 2-23                        [16, 512, 28, 28]         --
|    + MaxPool2d: 2-24                   [16, 512, 14, 14]         --
|    + Conv2d: 2-25                      [16, 512, 14, 14]         2,359,808
|    + ReLU: 2-26                        [16, 512, 14, 14]         --
|    + Conv2d: 2-27                      [16, 512, 14, 14]         2,359,808
|    + ReLU: 2-28                        [16, 512, 14, 14]         --
|    + Conv2d: 2-29                      [16, 512, 14, 14]         2,359,808
|    + ReLU: 2-30                        [16, 512, 14, 14]         --
```

```
|    + MaxPool2d: 2-31                   [16, 512, 7, 7]          --
+ AdaptiveAvgPool2d: 1-2                 [16, 512, 7, 7]          --
+ Sequential: 1-3                        [16, 1000]               --
|    + Linear: 2-32                      [16, 4096]               102,764,544
|    + ReLU: 2-33                        [16, 4096]               --
|    + Dropout: 2-34                     [16, 4096]               --
|    + Linear: 2-35                      [16, 4096]               16,781,312
|    + ReLU: 2-36                        [16, 4096]               --
|    + Dropout: 2-37                     [16, 4096]               --
|    + Linear: 2-38                      [16, 1000]               4,097,000
================================================================================
==========
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
Total mult-adds (G): 247.74
================================================================================
==========
Input size (MB): 9.63
Forward/backward pass size (MB): 1735.26
Params size (MB): 553.43
Estimated Total Size (MB): 2298.32
================================================================================
==========
```

2 Exercise 2

Hi, I'm Mohammad Rostami. I'm a Ph.D. candidate in ECE, I specialize in few-shot learning. I studied math for myy bachlor's and master's so I'm familiar to most subjects of math. I started machine learning at 2018 and was a RA and Machine Learning Engineering for a couple of years.