Deep Learning with Pytorch -- Eli Stevens, Luca Antiga, and Thomas Viehmann. Learning Chapters 1-4

In [ ]:
```
from torchvision import models
```

In [ ]:
```
dir(models)
```

Out[ ]:
```
['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',
 'GoogLeNetOutputs',
 'GoogLeNet_Weights',
 'Inception3',
 'InceptionOutputs',
 'Inception_V3_Weights',
 'MNASNet',
 'MNASNet0_5_Weights',
 'MNASNet0_75_Weights',
 'MNASNet1_0_Weights',
 'MNASNet1_3_Weights',
 'MaxVit',
 'MaxVit_T_Weights',
 'MobileNetV2',
 'MobileNetV3',
 'MobileNet_V2_Weights',
 'MobileNet_V3_Large_Weights',
 'MobileNet_V3_Small_Weights',
 'RegNet',
 'RegNet_X_16GF_Weights',
 'RegNet_X_1_6GF_Weights',
 'RegNet_X_32GF_Weights',
 'RegNet_X_3_2GF_Weights',
 'RegNet_X_400MF_Weights',
 'RegNet_X_800MF_Weights',
 'RegNet_X_8GF_Weights',
 'RegNet_Y_128GF_Weights',
 'RegNet_Y_16GF_Weights',
 'RegNet_Y_1_6GF_Weights',
 'RegNet_Y_32GF_Weights',
 'RegNet_Y_3_2GF_Weights',
 'RegNet_Y_400MF_Weights',
```

```
'RegNet_Y_800MF_Weights',
'RegNet_Y_8GF_Weights',
'ResNeXt101_32X8D_Weights',
'ResNeXt101_64X4D_Weights',
'ResNeXt50_32X4D_Weights',
'ResNet',
'ResNet101_Weights',
'ResNet152_Weights',
'ResNet18_Weights',
'ResNet34_Weights',
'ResNet50_Weights',
'ShuffleNetV2',
'ShuffleNet_V2_X0_5_Weights',
'ShuffleNet_V2_X1_0_Weights',
'ShuffleNet_V2_X1_5_Weights',
'ShuffleNet_V2_X2_0_Weights',
'SqueezeNet',
'SqueezeNet1_0_Weights',
'SqueezeNet1_1_Weights',
'SwinTransformer',
'Swin_B_Weights',
'Swin_S_Weights',
'Swin_T_Weights',
'Swin_V2_B_Weights',
'Swin_V2_S_Weights',
'Swin_V2_T_Weights',
'VGG',
'VGG11_BN_Weights',
'VGG11_Weights',
'VGG13_BN_Weights',
'VGG13_Weights',
'VGG16_BN_Weights',
'VGG16_Weights',
'VGG19_BN_Weights',
'VGG19_Weights',
'ViT_B_16_Weights',
'ViT_B_32_Weights',
'ViT_H_14_Weights',
'ViT_L_16_Weights',
'ViT_L_32_Weights',
'VisionTransformer',
'Weights',
'WeightsEnum',
'Wide_ResNet101_2_Weights',
'Wide_ResNet50_2_Weights',
'_GoogLeNetOutputs',
'_InceptionOutputs',
'__builtins__',
'__cached__',
'__doc__',
'__file__',
'__loader__',
'__name__',
'__package__',
'__path__',
'__spec__',
```

```
'_api',
'_meta',
'_utils',
'alexnet',
'convnext',
'convnext_base',
'convnext_large',
'convnext_small',
'convnext_tiny',
'densenet',
'densenet121',
'densenet161',
'densenet169',
'densenet201',
'detection',
'efficientnet',
'efficientnet_b0',
'efficientnet_b1',
'efficientnet_b2',
'efficientnet_b3',
'efficientnet_b4',
'efficientnet_b5',
'efficientnet_b6',
'efficientnet_b7',
'efficientnet_v2_l',
'efficientnet_v2_m',
'efficientnet_v2_s',
'get_model',
'get_model_builder',
'get_model_weights',
'get_weight',
'googlenet',
'inception',
'inception_v3',
'list_models',
'maxvit',
'maxvit_t',
'mnasnet',
'mnasnet0_5',
'mnasnet0_75',
'mnasnet1_0',
'mnasnet1_3',
'mobilenet',
'mobilenet_v2',
'mobilenet_v3_large',
'mobilenet_v3_small',
'mobilenetv2',
'mobilenetv3',
'optical_flow',
'quantization',
'regnet',
'regnet_x_16gf',
'regnet_x_1_6gf',
'regnet_x_32gf',
'regnet_x_3_2gf',
'regnet_x_400mf',
```

```
 'regnet_x_800mf',
 'regnet_x_8gf',
 'regnet_y_128gf',
 'regnet_y_16gf',
 'regnet_y_1_6gf',
 'regnet_y_32gf',
 'regnet_y_3_2gf',
 'regnet_y_400mf',
 'regnet_y_800mf',
 'regnet_y_8gf',
 'resnet',
 'resnet101',
 'resnet152',
 'resnet18',
 'resnet34',
 'resnet50',
 'resnext101_32x8d',
 'resnext101_64x4d',
 'resnext50_32x4d',
 'segmentation',
 'shufflenet_v2_x0_5',
 'shufflenet_v2_x1_0',
 'shufflenet_v2_x1_5',
 'shufflenet_v2_x2_0',
 'shufflenetv2',
 'squeezenet',
 'squeezenet1_0',
 'squeezenet1_1',
 'swin_b',
 'swin_s',
 'swin_t',
 'swin_transformer',
 'swin_v2_b',
 'swin_v2_s',
 'swin_v2_t',
 'vgg',
 'vgg11',
 'vgg11_bn',
 'vgg13',
 'vgg13_bn',
 'vgg16',
 'vgg16_bn',
 'vgg19',
 'vgg19_bn',
 'video',
 'vision_transformer',
 'vit_b_16',
 'vit_b_32',
 'vit_h_14',
 'vit_l_16',
 'vit_l_32',
 'wide_resnet101_2',
 'wide_resnet50_2']
```

In [ ]:
```
resnet = models.resnet101(pretrained=True)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: Use
rWarning: The parameter 'pretrained' is deprecated since 0.13 and may be remo
ved in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: Use
rWarning: Arguments other than a weight enum or `None` for 'weights' are depr
ecated since 0.13 and may be removed in the future. The current behavior is e
quivalent to passing `weights=ResNet101_Weights.IMAGENET1K_V1`. You can also
use `weights=ResNet101_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet101-63fe2227.pth" to
/root/.cache/torch/hub/checkpoints/resnet101-63fe2227.pth
100%|██████████| 171M/171M [00:00<00:00, 193MB/s]
```

In [ ]:
```
resnet
```

Out[ ]:
```
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=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_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_run
ning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
```

```
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
```

```
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
```

```
    )
    (6): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (7): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (8): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (9): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
```

```
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (10): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (11): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (12): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (13): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
```

```
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (14): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (15): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (16): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (17): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
```

```
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (18): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (19): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (20): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
```

```
      (relu): ReLU(inplace=True)
    )
    (21): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (22): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): Bottleneck(
```

```
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

In [ ]:
```
# Before giving inputs to the model, we need to transform them.
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])
])

# we scale image to 256 x 256, crop it to 224 x 224 around the center, trans
# it to a tensor, normalize the RGB to defined mean and std to match the trai
from PIL import Image  # Import the Image object
img = Image.open('/content/pexels-pixabay-416179.jpg')
```

In [ ]:
```
display(img)
```

Output hidden; open in https://colab.research.google.com to view.

In [ ]:
```
# Get the dimensions
width, height = img.size
```

```
print(f"Image dimensions: Width = {width}, Height = {height}")

# You can also print the size tuple directly
print(f"Image size tuple: {img.size}")
```

```
Image dimensions: Width = 6026, Height = 4020
Image size tuple: (6026, 4020)
```

In [ ]:
```
img_t = preprocess(img)
```

In [ ]:
```
import torch
batch_t = torch.unsqueeze(img_t, 0)
```

In [ ]:
```
resnet.eval()
```

Out[ ]:
```
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=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_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_run
ning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_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_runn
ing_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
```

```
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
```

```
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
```

```
    )
    (6): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (7): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (8): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (9): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
```

```
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (10): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (11): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (12): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (13): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
```

```
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (14): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (15): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (16): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (17): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
```

```
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (18): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (19): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (20): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
```

```
      (relu): ReLU(inplace=True)
    )
    (21): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (22): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): Bottleneck(
```

```
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

In [ ]:
```
out = resnet(batch_t)
out
```

Out[ ]:
```
tensor([[-7.6672e-01,  2.1670e+00, -1.8659e+00, -3.5915e-01, -2.1249e+00,
         -1.7501e+00, -1.9125e+00,  5.0394e-03,  2.6134e+00, -3.4251e+00,
          1.0333e+01,  8.7033e+00,  1.1506e+01,  9.5496e+00,  1.0374e+01,
          5.8207e+00,  9.0328e+00,  6.9435e+00,  3.7255e+00,  6.7266e+00,
          7.1344e+00,  4.0496e+00, -1.6804e+00,  1.1439e+00,  4.0845e-01,
         -9.8793e-01,  6.0407e-01,  8.7832e-01, -2.4239e-01,  8.6290e-01,
          1.8759e-01,  9.3672e-01,  5.9699e-01, -1.9090e+00, -2.1212e+00,
          4.1397e-01,  1.0728e+00,  2.1460e+00,  5.4484e-01, -6.7102e-01,
          4.2157e+00,  2.1931e+00,  3.6355e+00,  1.7523e+00,  1.2219e+00,
         -4.5293e-01,  2.8808e+00,  2.5661e+00, -1.1280e+00, -2.1502e+00,
         -2.0490e+00, -2.7249e-01,  1.4543e+00,  4.3040e-01, -9.9054e-01,
          1.8244e+00, -6.3292e-01,  2.5539e-02,  1.2437e+00,  3.8578e+00,
          9.5514e-01, -6.0785e-01,  2.1912e-03, -1.3756e+00,  2.0734e+00,
         -1.3858e+00,  4.9343e-01, -1.4269e+00, -1.1190e+00, -1.0282e+00,
          1.3062e+00, -1.5478e+00,  4.3777e-01,  3.8362e-01, -1.6106e-01,
          1.6620e+00, -1.7068e+00,  1.4392e+00,  2.7804e+00,  2.2416e-01,
          2.6312e+00,  2.5561e+00,  5.1397e+00,  2.0191e+00, -1.6233e+00,
          3.4742e+00,  6.5189e+00,  2.6758e+00,  2.2609e+00,  1.4100e+00,
          4.6829e+00,  6.0551e+00,  8.9535e+00,  2.2284e+00,  6.8281e+00,
          7.4998e+00,  1.1998e+00,  8.7725e-01,  1.0560e+00,  7.2790e-03,
         -1.7843e+00, -2.8449e+00, -1.5508e+00,  1.8438e-02, -2.3738e+00,
         -1.5487e+00, -2.0442e+00, -9.3456e-01, -1.9813e+00, -1.8429e+00,
         -2.1880e+00, -9.2045e-01,  2.7308e-01,  3.5112e+00,  4.4405e-01,
         -1.4114e+00, -2.8543e+00,  2.3921e-01, -1.3653e+00, -9.5676e-01,
         -3.3451e-01, -1.6699e+00,  3.8907e-01, -4.0993e-01,  1.6630e+00,
          2.0371e-02, -1.1068e+00, -8.7809e-01, -8.9186e-01,  5.7463e-01,
         -1.6320e-01,  2.2757e+00,  4.3540e-01,  4.7704e+00,  1.3589e+00,
         -3.9482e-01,  6.5625e+00,  2.9475e+00, -1.2802e+00,  9.0008e-01,
          3.8381e+00,  3.4830e+00,  3.4882e+00,  2.2184e+00, -2.1914e+00,
         -1.5437e+00, -2.4163e+00, -3.9697e+00, -5.1137e+00, -2.8630e+00,
         -6.4653e-01, -5.1596e-01,  1.4724e-01, -1.2256e+00, -4.2297e-01,
          6.3140e-01, -7.4637e-01, -1.4161e+00, -2.8215e+00,  1.0528e+00,
         -2.0542e+00, -1.5237e+00, -2.4123e+00,  1.5339e-01, -1.8051e+00,
         -2.8335e+00, -1.9284e+00, -8.2754e-01, -5.3824e-02, -1.3356e+00,
         -1.0242e+00, -2.2090e+00, -3.5880e-01, -1.0829e+00, -2.1080e+00,
         -1.2889e-01, -1.0398e+00, -1.2876e+00, -7.8831e-01, -1.2384e+00,
          7.9410e-02, -1.2281e+00,  6.8301e-01, -5.1557e-01, -8.5871e-01,
         -7.7083e-02,  2.1792e-01,  6.1334e-01,  2.9855e-01, -1.0932e+00,
         -1.0269e+00,  2.6591e-01,  5.3052e-01,  2.2022e-01, -8.4932e-01,
         -3.5976e+00, -2.1355e+00, -1.9749e+00, -6.5611e-01, -2.1248e+00,
          1.5264e+00, -2.5819e+00,  1.1425e+00, -2.9429e+00,  1.5384e+00,
         -1.2667e+00, -1.2378e+00,  2.9312e-01, -2.5945e-02,  1.0142e+00,
         -1.5097e+00,  3.4358e-01, -2.4857e-01, -6.1833e-01,  2.9182e-01,
         -3.5288e-01, -5.3765e-01, -2.9564e+00, -6.9246e-01,  4.7337e-01,
          1.0712e+00, -1.5436e+00, -2.6755e+00, -1.5541e+00, -1.1867e+00,
          9.0673e-01,  6.9080e-01, -1.6591e+00, -2.4395e+00, -2.1489e+00,
         -1.6929e+00, -4.5576e-01, -2.8705e+00,  2.0961e-01, -1.5716e-01,
         -3.8530e-01, -5.7157e-01, -2.5162e+00, -9.9979e-01, -7.5575e-01,
         -9.9282e-02, -1.7464e+00, -1.1130e+00,  1.1408e-01, -1.8751e+00,
         -1.9440e+00, -1.5638e+00, -2.3073e+00, -1.6565e+00, -1.6954e+00,
         -2.3720e+00, -1.8856e+00, -7.0037e-01, -5.5150e-01, -1.3031e+00,
          1.9594e-03, -4.8780e-01, -2.7870e+00, -2.7624e+00, -8.9183e-02,
         -7.5253e-01, -1.2710e+00,  1.1543e+00, -9.0153e-01, -2.1500e+00,
         -7.2293e-01, -9.9373e-01, -1.2828e+00, -1.4111e+00, -2.0194e+00,
         -2.3722e+00, -1.0784e+00, -1.1851e+00, -7.2593e-01,  1.7324e-01,
         -1.3723e+00, -1.6816e+00,  5.8986e-02, -1.7770e+00, -2.6908e+00,
```

-1.0911e+00, -7.1650e-01, 5.7657e-01, 8.1083e-01, -1.9204e+00,
-1.1695e+00, -1.6191e+00, -2.0580e+00, -1.8132e+00, -3.9935e+00,
-2.0947e+00, -1.8597e+00, -9.2276e-01, -1.0842e+00, -2.2407e+00,
-1.7818e+00, -3.5773e+00, -2.9641e+00, 3.6775e-01, -6.1824e-01,
 7.2230e-01, 3.8337e+00, 1.6679e-01, 1.8410e+00, 2.2925e+00,
 6.3550e-01, 8.9117e-01, 1.4078e+00, 2.0791e+00, 1.1483e+00,
 8.9077e-01, 2.8422e+00, 1.6792e+00, 3.7574e+00, -6.1552e-01,
 3.2170e+00, 3.0831e+00, 6.4237e+00, 1.3040e+00, 3.9660e+00,
 4.5477e+00, 2.8380e+00, 3.9571e+00, 3.3985e+00, 2.2349e+00,
 3.6766e+00, 3.9995e+00, -1.7721e+00, -1.6683e+00, -2.7164e+00,
 2.2457e-01, 6.3626e-01, -1.2052e+00, 2.6477e+00, -1.8302e+00,
 4.9702e-01, 1.1247e-01, 1.5569e+00, 9.5052e-01, -3.6700e+00,
-3.1877e+00, -1.7665e+00, -2.7033e+00, -3.4571e+00, -3.5785e+00,
-1.9010e+00, -1.5935e+00, -3.5743e+00, -3.1215e+00, -3.6382e+00,
-2.6341e+00, -2.1734e+00, -3.8840e-02, -1.7440e+00, -2.9015e+00,
-4.9794e+00, 1.1275e+00, 1.3005e+00, 7.3729e-01, -4.9055e-01,
-1.2841e+00, -1.9573e+00, -2.2704e+00, -1.6693e+00, 1.2937e+00,
-1.0841e+00, -1.3393e+00, -1.0265e+00, -2.2098e+00, -1.1089e+00,
 1.1161e+00, 3.1732e+00, -1.0273e+00, 2.6542e+00, 1.0145e-01,
-2.0300e+00, 2.7593e+00, 3.1634e+00, -5.4136e-01, -1.1110e-01,
 1.8390e+00, 1.5918e-01, 4.6692e-01, -1.2319e+00, -3.4013e-01,
-2.8745e+00, -3.4177e+00, -6.2490e-01, -4.0008e+00, -1.8524e-01,
-1.1303e+00, -2.7934e+00, -2.1255e-01, -2.0569e+00, 2.9049e-01,
 1.1481e+00, -2.4081e+00, -1.4266e+00, 1.0793e+00, -1.7947e+00,
-2.3414e+00, -5.9929e-01, -1.5855e+00, -2.8784e+00, 3.8435e-01,
 6.9042e-01, -9.5497e-01, 9.6855e-01, -1.4197e+00, 1.6986e+00,
-7.4229e-01, -9.3178e-01, 2.7127e-01, -7.5850e-01, -2.2255e+00,
 3.7299e-01, -9.7144e-01, 7.6695e-01, -4.0378e-01, 9.5668e-01,
-1.1541e+00, 9.5352e-01, 1.7902e-02, -1.0777e+00, -1.7135e+00,
-1.8565e+00, 1.2985e+00, 9.1515e-01, 1.4147e+00, 1.2106e+00,
-1.5356e+00, 1.2126e+00, 3.3500e-01, 7.9068e-01, -1.5135e+00,
 3.0341e-01, 5.0358e-01, 8.8584e-01, 1.1772e+00, -2.6799e-01,
 2.6513e-01, -9.7882e-01, 5.2090e-01, -1.0027e+00, -2.0422e+00,
 3.1265e-01, -7.9011e-01, 1.2313e+00, 2.9487e+00, -5.3819e-01,
-1.7682e+00, -7.2621e-01, -2.1660e+00, 9.4119e-03, -1.6068e+00,
 1.5144e+00, 9.2985e-01, -5.7066e-01, -1.8586e+00, -1.8671e+00,
-1.1919e+00, -9.5745e-02, 7.3080e-01, 1.2032e+00, -6.3433e-02,
-2.2859e+00, -1.8527e+00, -1.7865e+00, -2.2341e+00, -7.3976e-02,
 1.1720e+00, -5.0116e-01, 1.8885e+00, 4.0839e-01, -2.2826e+00,
-1.7996e+00, -2.4916e+00, 6.6977e-01, -8.5671e-01, -1.2849e+00,
-4.0719e-01, -1.2414e+00, -3.8324e-01, -8.7943e-01, -6.0412e-01,
 6.2796e-01, -1.2802e+00, 1.7228e+00, 1.7828e+00, 1.0922e+00,
-9.8452e-01, -1.5416e-01, -1.8345e+00, -8.5284e-04, 3.4161e+00,
 3.9602e-01, -4.1509e+00, -1.0581e+00, -1.5327e+00, 7.5457e-01,
-2.4914e+00, -2.2186e+00, 2.4744e+00, 7.5748e-01, 7.8680e-01,
 1.3175e+00, 1.4186e+00, -1.4257e+00, 1.0759e-02, -8.0396e-01,
-2.4347e+00, -5.3377e-01, -1.0721e+00, -1.2238e+00, -6.0640e-01,
 3.4903e-01, 7.6731e-01, -1.0704e-01, -9.5850e-01, 6.1625e-01,
 2.7403e-01, -7.8778e-01, 4.6075e+00, 1.0026e+00, -9.2706e-01,
-1.3467e+00, 9.5185e-01, 3.7161e-01, -2.5577e-02, 4.7305e-01,
 1.5140e+00, -1.1051e-01, 1.4046e+00, -8.8619e-01, 5.4549e-01,
-7.6806e-01, -1.6561e+00, -2.8886e+00, -1.0954e-01, -1.2950e+00,
-1.2280e+00, 2.5970e+00, 2.0450e+00, -1.8872e-01, 9.4838e-01,
 2.2822e+00, -2.0662e+00, -3.1879e+00, -1.8688e-01, -1.1736e+00,
 3.0443e-02, 1.2168e+00, -1.2791e+00, -9.2589e-01, -2.2671e+00,
-2.2483e-01, 1.1376e+00, 8.9204e-01, -5.8203e-02, 3.0070e+00,

-2.3878e+00, -6.1674e-01, -1.7866e-01, -2.4418e-01, -1.0663e+00,
-1.8538e+00, 5.4506e-01, -2.5772e-01, -1.7434e+00, -1.2459e+00,
-5.6655e-01, -1.1770e+00, 1.4352e+00, -1.4701e+00, 4.9957e+00,
-1.1105e+00, -1.2675e+00, 2.2891e+00, -7.8745e-01, -5.7469e-01,
-6.4417e-01, -3.3285e-01, -9.6472e-01, 2.3649e+00, 2.5258e+00,
 5.3682e-01, 1.2559e+00, 1.1449e+00, 2.3181e+00, -8.5253e-01,
 1.7179e+00, 6.6746e-02, 1.1133e+00, -7.1914e-02, -9.1486e-01,
-5.6631e-03, 4.0435e-01, -3.5815e+00, -1.8863e-01, 1.3565e+00,
 1.6805e+00, -5.5696e-01, -7.1290e-01, -1.5845e+00, 4.2997e+00,
 1.3285e+00, 1.6716e+00, -1.9878e+00, -1.8681e+00, -3.2948e-01,
-1.2198e+00, 9.3075e-01, -1.2996e+00, 1.0783e+00, -1.4421e+00,
-1.7490e+00, 6.3101e-01, -1.2687e-01, 2.5230e+00, 2.0689e+00,
 7.0700e-01, 7.0958e-01, -2.6650e-01, -9.5475e-01, -2.3962e-01,
-1.4374e+00, 6.8002e-01, -2.1084e+00, -1.5500e+00, -5.7281e-01,
-1.2675e+00, 6.4236e-01, 4.1649e+00, 2.8574e+00, -1.2016e+00,
 2.5718e+00, -3.4394e+00, -7.3762e-01, 2.4695e-01, -7.0855e-01,
-2.2140e+00, 8.0768e-01, 4.5567e-01, -1.0040e+00, 2.2931e+00,
-1.2851e+00, -4.3993e-02, 1.3333e+00, -5.0200e-01, -2.6344e+00,
-4.6210e-01, -7.5076e-01, 2.7594e-01, 7.0892e-01, -2.1801e+00,
-1.3972e-01, -3.5897e-01, 6.9337e-01, -8.3048e-01, 3.6604e+00,
-5.1158e-01, -1.9922e+00, -1.4721e+00, -2.7763e+00, 8.2562e-01,
-1.3000e+00, 1.7264e+00, -9.7208e-01, -1.8244e+00, -6.6861e-01,
-1.4473e+00, -2.1187e+00, -3.9639e-02, 8.0735e-01, 2.7477e+00,
-2.2843e+00, -7.7595e-01, 3.8486e+00, 5.7629e-01, 2.5499e+00,
-2.7506e-01, 1.1463e+00, -8.6825e-01, 3.6492e-01, 5.2689e-01,
-1.2618e+00, -1.5272e+00, -8.1261e-01, 2.3123e-01, -1.7484e+00,
-1.2076e+00, -1.2236e+00, 7.7410e-01, 1.8333e+00, -3.2114e+00,
-1.4636e+00, 2.0775e+00, -6.4226e-01, -3.6768e-02, -1.1074e-01,
 1.3143e+00, -7.5248e-01, 4.4208e-01, 2.9316e-01, -1.3677e+00,
-5.0287e-01, -4.9506e-01, -2.6086e+00, 2.2787e+00, -4.2642e-01,
 9.7533e-01, 8.1457e-01, 1.0257e+00, 8.4044e-01, -1.7515e-02,
-2.1446e+00, -4.1651e-01, -1.7155e+00, -1.9601e+00, 1.2920e-01,
 6.3987e-01, 1.8786e-01, 1.7048e+00, 1.6352e+00, -1.0635e+00,
 3.4490e+00, 3.6874e-01, 1.1724e-01, -9.7294e-02, 2.8303e+00,
 1.5190e+00, 8.2541e-01, -1.1065e+00, 2.8292e+00, -1.4165e+00,
-1.1984e+00, 9.5951e-01, 1.3096e-01, 2.8130e+00, 2.4905e+00,
-4.2513e-01, -2.1850e+00, 5.7533e-01, 1.3009e+00, 1.2262e+00,
-1.1841e-01, -3.0517e-01, -9.3429e-01, -1.6588e+00, 7.1822e-01,
-1.7756e+00, -2.0565e+00, 5.9846e-01, -9.8793e-01, 3.0959e-02,
-1.2597e+00, -4.5453e-01, -1.6427e-01, 1.4441e+00, -2.5030e-01,
 7.0198e-01, 1.6836e+00, 5.5543e-01, -2.6694e+00, 5.5222e-01,
 1.8019e+00, 1.2554e+00, 6.2972e-01, 1.2704e-02, 1.2859e+00,
-1.0378e+00, -1.5718e+00, -4.5629e-01, 3.9134e+00, 9.6119e-01,
 6.1887e-01, -1.9374e+00, -1.8246e+00, 3.1611e+00, 4.2474e-01,
 1.4845e+00, -2.6283e-01, 1.4691e-01, 9.3884e-01, 1.8381e+00,
-2.2306e+00, 7.2297e-01, 8.6994e-01, -1.1632e+00, 1.1850e+00,
-6.7746e-01, -6.6928e-01, 1.5163e+00, -4.9555e-01, -8.3070e-01,
-6.1779e-01, -1.5689e+00, -1.0669e+00, -4.5873e-01, 1.9548e-01,
 3.5391e-01, -2.4986e+00, -1.5707e+00, -1.7010e+00, 1.6493e+00,
-5.1382e-01, -1.5638e+00, 1.7481e+00, -7.9060e-01, 1.6477e+00,
 1.8917e+00, 1.0017e+00, 4.1047e-01, 4.4179e-01, 2.1061e+00,
 1.0714e+00, 1.1588e+00, -2.3940e+00, 3.1249e+00, -1.5398e+00,
-2.6056e+00, -2.4803e+00, 5.6064e-01, 8.8393e-01, -1.9732e+00,
-9.1020e-01, 1.0490e-01, 1.3344e+00, 2.0221e+00, -2.2557e+00,
 1.4470e+00, -6.3687e-01, -2.7263e+00, -1.9922e+00, -6.0421e-01,
 1.1329e+00, 4.7832e-01, 2.8510e-01, 3.0578e-01, -2.2362e+00,

```
 4.4982e-01, -2.7373e+00,  9.9796e-01, -1.3709e-01, -7.4802e-01,
-6.4859e-02,  3.2238e+00, -1.4283e+00,  1.9904e-02,  1.7180e+00,
 2.9704e-01,  1.7343e+00,  2.0894e+00, -2.1764e-01, -2.7993e+00,
 2.6619e+00, -1.6940e+00, -1.8881e+00, -7.0489e-01,  1.5122e+00,
-6.1146e-01,  1.4681e+00,  8.0688e-01, -1.5400e+00, -1.6163e+00,
-1.8947e+00,  7.6521e-01, -1.6764e+00,  4.1942e+00, -2.1901e+00,
-1.4241e+00, -1.4469e+00,  7.0191e-01, -1.3866e+00, -2.8448e+00,
-9.2212e-02,  8.5133e-01, -3.5889e-01,  5.7083e-01, -1.4241e+00,
-2.2782e+00, -2.4160e+00,  1.3270e+00,  3.7678e+00, -1.1960e+00,
-2.4203e+00, -1.2760e+00, -7.9340e-01, -1.9453e+00, -1.2073e+00,
-2.4606e+00,  1.1299e-01,  3.5146e+00, -1.1762e+00, -2.5517e-01,
 4.2317e-01,  6.0871e-01, -4.1350e-01,  1.8510e+00,  3.6808e+00,
 5.3187e-01, -1.0934e+00,  5.3546e-01, -1.8079e-01,  1.9056e+00,
-2.5954e-02, -1.6800e+00,  2.6916e-01,  9.4566e-01,  7.7844e-01,
 2.2482e+00,  5.2081e-01,  2.2988e+00, -1.2942e+00, -1.4424e+00,
-7.5189e-01, -6.7188e-01, -2.5646e+00, -9.4936e-01, -4.8072e-01,
-5.7943e-01,  3.0336e-01, -2.4917e+00,  2.8202e+00,  1.7634e+00,
 5.6364e-01,  3.8508e-01,  5.1279e-01, -4.5960e-01,  1.1692e+00,
 7.0737e-01,  6.2264e-02, -4.1567e-01, -2.4086e+00, -1.2254e+00,
 9.5580e-01, -3.9474e-01,  1.7323e+00, -4.2429e-01, -2.9365e-01,
 1.4983e+00,  1.0131e+00,  1.6991e+00,  1.7620e+00,  1.1579e+00,
 4.8169e-02,  1.4915e+00,  3.5153e+00,  3.5270e+00,  1.8194e+00,
 6.1482e-01,  1.6415e+00,  4.2881e+00,  8.7787e-01,  1.3485e+00,
-1.7249e+00,  3.1921e+00,  3.2431e+00, -5.2259e-01, -1.4089e+00,
-1.1786e-01,  5.2279e-01,  9.0613e-01, -1.4583e+00,  1.5852e+00,
 3.9912e-01, -2.6468e-01, -1.3776e+00,  1.3819e+00,  1.1122e+00,
-1.4710e-01, -9.4923e-01, -1.4933e+00, -2.5270e-01, -1.9681e+00,
 2.9495e+00, -3.6040e-01,  3.4254e-01,  1.6510e-01, -1.1873e+00,
-1.2063e+00,  1.7036e-01, -2.6753e-01, -2.4681e+00,  2.9400e+00,
 1.1826e-01, -1.9301e+00,  2.6466e+00,  5.1163e+00,  3.1189e+00,
 1.4744e+00, -1.3048e+00,  1.0539e-01,  2.3443e-01,  9.5747e-02,
 2.7871e+00,  1.5257e+00,  9.7584e-01,  2.9081e+00, -1.7030e+00]],
grad_fn=<AddmmBackward0>)
```

In [ ]:
```
# Load text labels presented to model while training
with open('/content/imagenet_classes.txt') as f:
  labels = [line.strip() for line in f.readlines()]
```

In [ ]:
```
_, index = torch.max(out, 1)
```

In [ ]:
```
# Index is not a number, its a one dimensional tensor, so to get numericla v
# We also use torch.nn.functional.softmax to normalize our output to a range
# and divide by the sum to give us a confidence that model has in the predict
percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
labels[index[0]], percentage[index[0]].item()
```

Out[ ]:
```
('house finch, linnet, Carpodacus mexicanus', 47.013099670410156)
```

In [ ]:
```
# To find second and third best we can us sort function

_, indices = torch.sort(out, descending=True) # Corrected the spelling of 'a
[(labels[idx], percentage[idx].item()) for idx in indices[0][:5]]
```

Out[ ]:
```
[('house finch, linnet, Carpodacus mexicanus', 47.013099670410156),
 ('indigo bunting, indigo finch, indigo bird, Passerina cyanea',
  15.15851879119873),
 ('brambling, Fringilla montifringilla', 14.547850608825684),
 ('junco, snowbird', 6.647336483001709),
 ('bulbul', 3.9646034240722656)]
```

CycleGan

In [ ]:
```
import torch
import torch.nn as nn

class ResNetBlock(nn.Module): # <1>

    def __init__(self, dim):
        super(ResNetBlock, self).__init__()
        self.conv_block = self.build_conv_block(dim)

    def build_conv_block(self, dim):
        conv_block = []

        conv_block += [nn.ReflectionPad2d(1)]

        conv_block += [nn.Conv2d(dim, dim, kernel_size=3, padding=0, bias=Tr
                       nn.InstanceNorm2d(dim),
                       nn.ReLU(True)]

        conv_block += [nn.ReflectionPad2d(1)]

        conv_block += [nn.Conv2d(dim, dim, kernel_size=3, padding=0, bias=Tr
                       nn.InstanceNorm2d(dim)]

        return nn.Sequential(*conv_block)

    def forward(self, x):
        out = x + self.conv_block(x) # <2>
        return out

# Ensure the indentation of this class definition matches the ResNetBlock cla
class ResNetGenerator(nn.Module):

    def __init__(self, input_nc=3, output_nc=3, ngf=64, n_blocks=9): # <3>

        assert(n_blocks >= 0)
        super(ResNetGenerator, self).__init__()

        self.input_nc = input_nc
        self.output_nc = output_nc
        self.ngf = ngf

        model = [nn.ReflectionPad2d(3),
                 nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0, bias=Tru
                 nn.InstanceNorm2d(ngf),
                 nn.ReLU(True)]
```

```
        n_downsampling = 2
        for i in range(n_downsampling):
            mult = 2**i
            model += [nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size=3,
                                stride=2, padding=1, bias=True),
                      nn.InstanceNorm2d(ngf * mult * 2),
                      nn.ReLU(True)]

        mult = 2**n_downsampling
        for i in range(n_blocks):
            model += [ResNetBlock(ngf * mult)]

        for i in range(n_downsampling):
            mult = 2**(n_downsampling - i)
            model += [nn.ConvTranspose2d(ngf * mult, int(ngf * mult / 2),
                                         kernel_size=3, stride=2,
                                         padding=1, output_padding=1,
                                         bias=True),
                      nn.InstanceNorm2d(int(ngf * mult / 2)),
                      nn.ReLU(True)]

        model += [nn.ReflectionPad2d(3)]
        model += [nn.Conv2d(ngf, output_nc, kernel_size=7, padding=0)]
        model += [nn.Tanh()]

        self.model = nn.Sequential(*model)

    def forward(self, input): # <3>
        return self.model(input)
```

In [ ]:
```
netG = ResNetGenerator()
```

In [ ]:
```
model_path = '/content/horse2zebra_0.4.0.pth'

model_data = torch.load(model_path)

# Load the state dictionary into the generator network.
netG.load_state_dict(model_data)
```

Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
netG.eval()
```

Out[ ]:
```
ResNetGenerator(
  (model): Sequential(
    (0): ReflectionPad2d((3, 3, 3, 3))
    (1): Conv2d(3, 64, kernel_size=(7, 7), stride=(1, 1))
    (2): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_runn
ing_stats=False)
    (3): ReLU(inplace=True)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (5): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_run
ning_stats=False)
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
    (8): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_run
ning_stats=False)
    (9): ReLU(inplace=True)
    (10): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (11): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (12): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (13): ResNetBlock(
```

```
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (14): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (15): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (16): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (17): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
```

```
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (18): ResNetBlock(
      (conv_block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
      )
    )
    (19): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padd
ing=(1, 1), output_padding=(1, 1))
    (20): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_ru
nning_stats=False)
    (21): ReLU(inplace=True)
    (22): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), paddi
ng=(1, 1), output_padding=(1, 1))
    (23): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_run
ning_stats=False)
    (24): ReLU(inplace=True)
    (25): ReflectionPad2d((3, 3, 3, 3))
    (26): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1))
    (27): Tanh()
  )
)
```

In [ ]:
```
# Now that we have loaded the pickle files with model weights, we can show th
from PIL import Image
from torchvision import transforms

preprocess2 = transforms.Compose([transforms.Resize(256),
                                  transforms.ToTensor()])
```

In [ ]:
```
img = Image.open('/content/horse2.jpg')
img
```

Output hidden; open in https://colab.research.google.com to view.

In [ ]:
```
img_t2 = preprocess2(img)
batch_t2 = torch.unsqueeze(img_t2, 0)
```

In [ ]:
```
# batch_out is now the output of the generator, which we can convert back to
batch_out = netG(batch_t2)
```

In [ ]:
```
out_t = (batch_out.data.squeeze() + 1.0) / 2.0
out_img = transforms.ToPILImage()(out_t)

out_img
```

Out[ ]:

Pytorch Fundamentals

In [ ]:
```
import torch
a = torch.ones(3)
a
```

Out[ ]:
```
tensor([1., 1., 1.])
```

In [ ]:
```
a[1]
```

Out[ ]:
```
tensor(1.)
```

In [ ]:
```
a[2] = 2
a
```

Out[ ]:
```
tensor([1., 1., 2.])
```

In [ ]:
```
points = torch.zeros(6)
points[0] = 4.0
points[1] = 1.0
points[2] = 5.0
points[3] = 3.0
points[4] = 2.0
points[5] = 1.0
```

In [ ]:
```
points = torch.tensor([4.0, 1.0, 5.0, 3.0, 2.0, 1.0])
points
```

Out[ ]:
```
tensor([4., 1., 5., 3., 2., 1.])
```

In [ ]:
```
float(points[0]), float(points[1])
```

Out[ ]:
```
(4.0, 1.0)
```

In [ ]:
```
points = torch.zeros(3, 2)
points
```

Out[ ]:
```
tensor([[0., 0.],
        [0., 0.],
        [0., 0.]])
```

In [ ]:
```
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
points
```

Out[ ]:
```
tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

In [ ]:
```
points.shape
```

Out[ ]:
```
torch.Size([3, 2])
```

In [ ]:
```
points[0,1]
```

Out[ ]:
```
tensor(1.)
```

In [ ]:
```
points[0]
```

Out[ ]:
```
tensor([4., 1.])
```

Indexing Tensors

In [ ]:
```
some_list = list(range(6))
```

In [ ]:
```
some_list[:]
```

Out[ ]:
```
[0, 1, 2, 3, 4, 5]
```

In [ ]:
```
some_list[1:4]
```

Out[ ]:
```
[1, 2, 3]
```

In [ ]:
```
some_list[:-1]
```

Out[ ]:
```
[0, 1, 2, 3, 4]
```

In [ ]:
```
some_list[1:4:2] # 1 to 4(exluding), step size = 2
```

Out[ ]:
```
[1, 3]
```

In [ ]:
```
points
```

Out[ ]:
```
tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

In [ ]:
```
points[1:]
```

Out[ ]:
```
tensor([[5., 3.],
        [2., 1.]])
```

In [ ]:
```
points[1:,0]
```

Out[ ]:
```
tensor([5., 2.])
```

In [ ]:
```
points[None] # adds an extra dimension of size 1, like unsqueeze
```

Out[ ]:
```
tensor([[[4., 1.],
         [5., 3.],
         [2., 1.]]])
```

In [ ]:
```
# Named Tensors
img_t = torch.randn(3,5,5) # Shape [ channel, rows, columns]
weights = torch.tensor([0.2126, 0.7152, 0.0722])
```

In [ ]:
```
batch_t = torch.randn(2,3,5,5) # Shape [batch, channes, rows, columns]
```

In [ ]:
```
img_gray_naive = img_t.mean(-3)
batch_gray_naive = batch_t.mean(-3)
img_gray_naive.shape, batch_gray_naive.shape
# The -3 in img_t.mean(-3) and batch_t.mean(-3) in PyTorch signifies the
# third dimension from the end (or the third to last dimension.
# When you specify a dimension in PyTorch, you can use:

# Positive indices: 0 for the first dimension, 1 for the second, and so on.
# Negative indices: -1 for the last dimension, -2 for the second to last,
# -3 for the third to last, and so on.
```

Out[ ]:
```
(torch.Size([5, 5]), torch.Size([2, 5, 5]))
```

Now, let's apply this to your tensors:

img_t = torch.randn(3,5,5)

Shape: [channel, rows, columns]

Dimensions:

0: channel (size 3)

1: rows (size 5)

2: columns (size 5)

Negative indices:

-1: columns

-2: rows

-3: channel

So, img_t.mean(-3) is taking the mean along the channel dimension. This makes sense for converting a color image (channels) to grayscale by averaging the channel values.

batch_t = torch.randn(2,3,5,5)

Shape: [batch, channels, rows, columns]

Dimensions:

0: batch (size 2)

1: channels (size 3)

2: rows (size 5)

3: columns (size 5)

Negative indices:

-1: columns

-2: rows

-3: channels

-4: batch

Similarly, batch_t.mean(-3) is taking the mean along the channels dimension for each image in the batch.

Why use negative indices?

Negative indices are particularly useful when you're working with tensors of varying numbers of dimensions, but you always want to perform an operation on a specific dimension relative to the end.

For example, if you always want to operate on the "channel" dimension, using -3 is robust whether your input is [channel, H, W] or [batch, channel, H, W].

If you used 0 for img_t and 1 for batch_t, your code would be less general. By using -3, you consistently target the "channel" dimension regardless of whether there's a batch dimension preceding it.

In summary: -3 here specifically targets the channel dimension, allowing the mean to be calculated across the color channels to produce a grayscale representation.

In PyTorch (and other libraries like NumPy), when you specify a dimension using dim in functions like mean(), sum(), max(), etc., you can use negative indices.

Here's what -3 signifies:

Negative Indexing for Dimensions:

Positive indices start from 0 for the first dimension, 1 for the second, and so on. Negative indices count from the last dimension backwards. So: -1 refers to the last dimension. -2 refers to the second to last dimension. -3 refers to the third to last dimension.

In [ ]:
```
unsqueezed_weights = weights.unsqueeze(-1).unsqueeze_(-1)
# unsqueeze adds a new dimension of size 1 at the specified position. (-1) me
# "add a new dimension at the last position." unsqueeze_ means it's an in-pl
# operation. It modifies unsqueezed_weights directly rather than returning a
# new tensor. Applying unsqueeze(-1) again to [3, 1] adds another dimension
# size 1 at the new last position. The shape [3, 1] becomes [3, 1, 1].

img_weights = (batch_t * unsqueezed_weights)
batch_weights = (batch_t * unsqueezed_weights)
img_gray_weighted = img_weights.sum(-3)
batch_gray_weighted = batch_t * unsqueezed_weights

batch_weights.shape, batch_t.shape, unsqueezed_weights.shape
```

Out[ ]:
```
(torch.Size([2, 3, 5, 5]), torch.Size([2, 3, 5, 5]), torch.Size([3, 1, 1]))
```

torch.einsum line is a fantastic, more advanced way to achieve the same weighted grayscale conversion, and it's definitely worth understanding!

torch.einsum (Einstein summation) is a very powerful and flexible function in PyTorch that allows you to express many common tensor operations (like dot products, matrix multiplication, batch multiplication, transposing, summing over specific dimensions, etc.) in a single, concise string notation.

'...chw,c->...hw'

**input signature left of -> ** describes the dimension of the input tensor. chw,c.

... (Ellipsis): This is the key to flexibility. It represents "any number of leading dimensions." It's a wildcard for dimensions that you want to keep without explicitly naming them.

chw: These are named dimensions for the first input tensor (e.g., img_t or batch_t).

They typically stand for:

c: channels

h: height (rows)

w: width (columns)

,: Separates the signatures of the input tensors.

c: This is the named dimension for the second input tensor (weights). Since weights has shape [3], it's just a single dimension, and we're calling it c to signify it corresponds to the c in chw.

Output Signature (right of ->): ...hw

...: Again, the ellipsis means "keep the same leading dimensions from the input.

hw: These are the dimensions that remain after the operation. Notice that c is not in the output signature. This implies that the c dimension will be summed over.

einsum implicitly performs two key operations based on the string:

Element-wise Multiplication: Any dimension names that appear in both input signatures are multiplied element-wise. In '...chw,c', the c appears in both (...chw and c). So, **for each image, for each channel, it will multiply the pixel values (h, w) by the corresponding c weight.**

Summation: Any dimension names that appear in the input signatures but not in the output signature are summed over.

Here, c is in the input (...chw,c) but not in the output (...hw). Therefore, after the multiplication, einsum sums along the c dimension.

In [ ]:
```
img_gray_weighted_fancy = torch.einsum('...chw,c->...hw', img_t, weights)
batch_gray_weighted_fancy = torch.einsum('...chw,c->...hw', batch_t, weights
batch_gray_weighted_fancy.shape
```

Out[ ]:
```
torch.Size([2, 5, 5])
```

In [ ]:
```
weights_named = torch.tensor([0.2126, 0.7152, 0.0722], names=['channels'])
weights_named
```

```
<ipython-input-50-847ddfeff394>:1: UserWarning: Named tensors and all their as
sociated APIs are an experimental feature and subject to change. Please do no
t use them for anything important until they are released as stable. (Trigger
ed internally at /pytorch/c10/core/TensorImpl.h:1935.)
  weights_named = torch.tensor([0.2126, 0.7152, 0.0722], names=['channels'])
```

Out[ ]:
```
tensor([0.2126, 0.7152, 0.0722], names=('channels',))
```

In [ ]:
```
# refine_names is used to assign or check names for the dimensions of an exis
# The ... (ellipsis) in refine_names acts as a placeholder for any leading di
# that you don't want to name explicitly, or whose names you want to leave a
img_named =  img_t.refine_names(..., 'channels', 'rows', 'columns')
```

```
batch_named = batch_t.refine_names(..., 'channels', 'rows', 'columns')
print("img named:", img_named.shape, img_named.names)
print("batch named:", batch_named.shape, batch_named.names)
```

```
img named: torch.Size([3, 5, 5]) ('channels', 'rows', 'columns')
batch named: torch.Size([2, 3, 5, 5]) (None, 'channels', 'rows', 'columns')
```

In [ ]:
```
# align_as(other_tensor) takes the names of the other_tensor's dimensions as
# target. It then transforms the tensor it's called on (weights_named) so th
# Dimension Order: Its named dimensions appear in the same order as in other
# Missing Dimensions: If weights_named is missing a dimension that is presen
# img_named, align_as adds a new dimension of size 1 at the correct named po
# This is equivalent to unsqueeze. Extra Dimensions: If weights_named has di
# that img_named does not, those dimensions are kept as is. (Not the case he
# weights_named has fewer dimensions)

weights_aligned = weights_named.align_as(img_named)
weights_aligned.shape, weights_aligned.names
```

Out[ ]:
```
(torch.Size([3, 1, 1]), ('channels', 'rows', 'columns'))
```

In [ ]:
```
gray_named = (img_named * weights_aligned).sum('channels')
gray_named.shape, gray_named.names
```

Out[ ]:
```
(torch.Size([5, 5]), ('rows', 'columns'))
```

In [ ]:
```
# If we want to use tensors outside functions that operate on named tensors,
# we need to drop the names by renaming them to None. The following gets us
# back into the world of unnamed dimensions

gray_plain = gray_named.rename(None)
gray_plain.shape, gray_plain.names
```

Out[ ]:
```
(torch.Size([5, 5]), (None, None))
```

In [ ]:
```
double_points = torch.ones(10, 2, dtype=torch.double)
short_points = torch.tensor([[1, 2], [3, 4]], dtype=torch.short)

print(double_points.dtype)
print(short_points.dtype)
```

```
torch.float64
torch.int16
```

In [ ]:
```
double_points = torch.zeros(10, 2).double()
short_points = torch.ones(10, 2).short()

print(double_points.dtype)
print(short_points.dtype)
```

```
torch.float64
torch.int16
```

In [ ]:
```
double_points = torch.zeros(10, 2).to(torch.double)
short_points = torch.ones(10, 2).to(dtype=torch.short)
```

```
print(double_points.dtype)
print(short_points.dtype)
```

```
torch.float64
torch.int16
```

In [ ]:
```
points_64 = torch.rand(5, dtype=torch.double)
points_short = points_64.to(torch.short)
points_64 * points_short  # works from PyTorch 1.3 onwards
```

Out[ ]:
```
tensor([0., 0., 0., 0., 0.], dtype=torch.float64)
```

In [ ]:
```
a = torch.ones(3, 2)
a_t = torch.transpose(a, 0, 1) #  swap dimension 0 with dimension 1.

a.shape, a_t.shape
```

Out[ ]:
```
(torch.Size([3, 2]), torch.Size([2, 3]))
```

In [ ]:
```
c = torch.randn(3, 4)
print("c.is_contiguous():", c.is_contiguous()) # True (by default)

c_t = c.t()
print("c_t.is_contiguous():", c_t.is_contiguous()) # False (after transpose)

c_t_contiguous = c_t.contiguous()
print("c_t_contiguous.is_contiguous():", c_t_contiguous.is_contiguous()) # T
print("Storage of c_t_contiguous (different from c):", c_t_contiguous.storage
```

```
c.is_contiguous(): True
c_t.is_contiguous(): False
c_t_contiguous.is_contiguous(): True
Storage of c_t_contiguous (different from c):  -1.6353520154953003
 0.3765576481819153
 0.009221578016877174
 1.2455615997314453
 -0.8631849884986877
 1.6234872341156006
 0.9897697567939758
 0.22726058959960938
 1.3420432806015015
 -0.6307931542396545
 -0.26443690061569214
 -0.2580842077732086
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 12]
```

```
<ipython-input-60-5d2925d31323>:9: UserWarning: TypedStorage is deprecated. I
t will be removed in the future and UntypedStorage will be the only storage c
lass. This should only matter to you if you are using storages directly.  To
access UntypedStorage directly, use tensor.untyped_storage() instead of tenso
r.storage()
  print("Storage of c_t_contiguous (different from c):", c_t_contiguous.storag
e()) # New storage
```

In [ ]:
```
a = torch.tensor([[1, 2], [3, 4]], dtype=torch.float32)
print("a's storage:", a.storage())
print("a's shape:", a.shape)
```

```
print("a's stride:", a.stride())

# Create a slice - this often results in a tensor with a conceptual offset
b = a[1, :] # This slice refers to the second row: [3, 4]
print("\n--- Tensor 'b' (a slice of 'a') ---")
print("b's storage (same as a's):", b.storage()) # Same storage instance
print("b's shape:", b.shape)
print("b's stride:", b.stride()) # Stride might change for slice
# Conceptually, b starts at the 2nd element of 'a's storage (index 2: '3.0')
# Index:  0    1    2    3
# Value: 1.0  2.0  3.0  4.0
```

```
a's storage:  1.0
 2.0
 3.0
 4.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 4]
a's shape: torch.Size([2, 2])
a's stride: (2, 1)

--- Tensor 'b' (a slice of 'a') ---
b's storage (same as a's):  1.0
 2.0
 3.0
 4.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 4]
b's shape: torch.Size([2])
b's stride: (1,)
```

In [ ]:
```
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
points.storage()
```

Out[ ]:
```
 4.0
 1.0
 5.0
 3.0
 2.0
 1.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 6]
```

In [ ]:
```
points_storage = points.storage()
points_storage[0]
```

Out[ ]:
```
4.0
```

In [ ]:
```
points.storage()[1]
```

Out[ ]:
```
1.0
```

In [ ]:
```
a = torch.ones(3, 2)

# zero_ works inplace by modifying the input instead of creating a new outpu
a.zero_()
```

Out[ ]:
```
tensor([[0., 0.],
        [0., 0.],
        [0., 0.]])
```

In [ ]:
```
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
second_point = points[1]
second_point
```

Out[ ]:
```
tensor([5., 3.])
```

In [ ]:
```
second_point.storage_offset()
# Storage Index: 0     1     2     3     4     5
# Storage Value: 4.0   1.0   5.0   3.0   2.0   1.0
```

Out[ ]:
```
2
```

In [ ]:
```
second_point.size()
```

Out[ ]:
```
torch.Size([2])
```

In [ ]:
```
second_point.shape
```

Out[ ]:
```
torch.Size([2])
```

In [ ]:
```
points.stride()
```

Out[ ]:
```
(2, 1)
```

In [ ]:
```
print("--- 1. Original Tensor 'a' ---")
a = torch.tensor([[10, 11, 12],
                  [13, 14, 15],
                  [16, 17, 18]], dtype=torch.float32)

print("Tensor 'a':\n", a)
print("a.shape (size):", a.shape)
print("a.stride():", a.stride()) # (3, 1) means: to move to next row, skip 3
print("a.storage():\n", a.storage()) # Shows the full 1D memory block

# Conceptual offset for 'a' is 0, as it starts from the beginning of its stor

print("\n--- 2. Transposed Tensor 'b' (a view) ---")
b = a.t() # Transpose 'a'
print("Tensor 'b' (a.t()):\n", b)
print("b.shape (size):", b.shape)
print("b.stride():", b.stride()) # (1, 3) means: to move to next row, skip 1
print("b.storage() (same as a's): \n", b.storage()) # Same storage instance!
# Conceptual offset for 'b' is also 0, as it still starts from the beginning
```

```
--- 1. Original Tensor 'a' ---
Tensor 'a':
 tensor([[10., 11., 12.],
        [13., 14., 15.],
        [16., 17., 18.]])
a.shape (size): torch.Size([3, 3])
a.stride(): (3, 1)
a.storage():
  10.0
 11.0
 12.0
 13.0
 14.0
 15.0
 16.0
 17.0
 18.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 9]

--- 2. Transposed Tensor 'b' (a view) ---
Tensor 'b' (a.t()):
 tensor([[10., 13., 16.],
        [11., 14., 17.],
        [12., 15., 18.]])
b.shape (size): torch.Size([3, 3])
b.stride(): (1, 3)
b.storage() (same as a's):
  10.0
 11.0
 12.0
 13.0
 14.0
 15.0
 16.0
 17.0
 18.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 9]
```

In [ ]:

```python
print("--- 1. Original Tensor 'a' ---")
a = torch.tensor([[10, 11, 12],
                  [13, 14, 15],
                  [16, 17, 18]], dtype=torch.float32)

print("Tensor 'a':\n", a)
print("\n--- 3. Sliced Tensor 'c' (another view) ---")
# Slice the second row of 'a'
c = a[1, :] # This takes the row [13, 14, 15]
print("Tensor 'c' (a[1, :]):\n", c)
print("c.shape (size):", c.shape)
print("c.stride():", c.stride()) # (1,) means: to move to next element, skip
print("c.storage() (same as a's): \n", c.storage()) # Still the same storage
# Conceptual offset for 'c':
# 'a' is 3x3, stride (3,1).
# a[1,:] means skip 1 row from start (1 * stride[0] = 1 * 3 = 3 elements).
# So, 'c' conceptually starts at storage index 3 (which holds 13.0).
```

```
--- 1. Original Tensor 'a' ---
Tensor 'a':
 tensor([[10., 11., 12.],
        [13., 14., 15.],
        [16., 17., 18.]])

--- 3. Sliced Tensor 'c' (another view) ---
Tensor 'c' (a[1, :]):
 tensor([13., 14., 15.])
c.shape (size): torch.Size([3])
c.stride(): (1,)
c.storage() (same as a's):
  10.0
 11.0
 12.0
 13.0
 14.0
 15.0
 16.0
 17.0
 18.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 9]
```

In [ ]:
```python
print("--- 1. Original Tensor 'a' ---")
a = torch.tensor([[10, 11, 12],
                  [13, 14, 15],
                  [16, 17, 18]], dtype=torch.float32)

print("Tensor 'a':\n", a)
print("\n--- 4. Sliced Tensor 'd' (a single element view) ---")
# Slice a single element
d = a[0, 1] # This takes the element 11
print("Tensor 'd' (a[0, 1]):\n", d)
print("d.shape (size):", d.shape) # Scalar tensor
print("d.stride():", d.stride()) # () for a scalar
print("d.storage() (same as a's): \n", d.storage()) # Still the same storage
# Conceptual offset for 'd':
# a[0,1] means 0 rows down (0 * stride[0]) + 1 column right (1 * stride[1])
# So, 'd' conceptually starts at storage index 1 (which holds 11.0).
```

```
--- 1. Original Tensor 'a' ---
Tensor 'a':
 tensor([[10., 11., 12.],
         [13., 14., 15.],
         [16., 17., 18.]])

--- 4. Sliced Tensor 'd' (a single element view) ---
Tensor 'd' (a[0, 1]):
 tensor(11.)
d.shape (size): torch.Size([])
d.stride(): ()
d.storage() (same as a's):
  10.0
 11.0
 12.0
 13.0
 14.0
 15.0
 16.0
 17.0
 18.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 9]
```

In [ ]:
```python
print("--- 1. Original Tensor 'a' ---")
a = torch.tensor([[10, 11, 12],
                  [13, 14, 15],
                  [16, 17, 18]], dtype=torch.float32)

print("Tensor 'a':\n", a)
print("\n--- 5. Contiguous Tensor 'e' (new storage) ---")
# Make the non-contiguous tensor 'b' contiguous
e = b.contiguous()
print("Tensor 'e' (b.contiguous()):\n", e)
print("e.shape (size):", e.shape)
print("e.stride():", e.stride()) # (3, 1) - now back to standard row-major f
print("e.storage() (NEW storage!):\n", e.storage()) # Will be a DIFFERENT st
# Conceptual offset for 'e' is 0, as it's a new, independently allocated stor
```

```
--- 1. Original Tensor 'a' ---
Tensor 'a':
 tensor([[10., 11., 12.],
         [13., 14., 15.],
         [16., 17., 18.]])

--- 5. Contiguous Tensor 'e' (new storage) ---
Tensor 'e' (b.contiguous()):
 tensor([[10., 13., 16.],
         [11., 14., 17.],
         [12., 15., 18.]])
e.shape (size): torch.Size([3, 3])
e.stride(): (3, 1)
e.storage() (NEW storage!):
  10.0
 13.0
 16.0
 11.0
 14.0
 17.0
 12.0
 15.0
 18.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 9]
```

In [ ]:
```
# Clone - so our original is not changed when we change an element
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
second_point = points[1].clone()
second_point[0] = 10.0
points
```

Out[ ]:
```
tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

In [ ]:
```
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
points
```

Out[ ]:
```
tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

In [ ]:
```
points_t = points.t()
points_t
```

Out[ ]:
```
tensor([[4., 5., 2.],
        [1., 3., 1.]])
```

In [ ]:
```
id(points.storage()) == id(points_t.storage())
```

Out[ ]:
```
False
```

In [ ]:
```
points.stride()
```

Out[ ]:
```
(2, 1)
```

In [ ]:
```
points_t.stride()
```

Out[ ]:
```
(1, 2)
```

In [ ]:
```
# Transposing in Higher dimension
some_t = torch.ones(3, 4, 5)
some_t
```

Out[ ]:
```
tensor([[[1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.]],

        [[1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.]],

        [[1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.]]])
```

In [ ]:
```
some_t.shape
```

Out[ ]:
```
torch.Size([3, 4, 5])
```

In [ ]:
```
some_t.stride()
```

Out[ ]:
```
(20, 5, 1)
```

In [ ]:
```
transpose_t = some_t.transpose(0, 2)
transpose_t
```

Out[ ]:
```
tensor([[[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]],

        [[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]],

        [[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]],

        [[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]],

        [[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]]])
```

In [ ]:
```
transpose_t.shape
```

Out[ ]:
```
torch.Size([5, 4, 3])
```

In [ ]:
```
transpose_t.stride()
```

Out[ ]:
```
(1, 5, 20)
```

In [ ]:
```
# Contiguous - usually default while transpose is not on
points.is_contiguous()
```

Out[ ]:
```
True
```

In [ ]:
```
points_t.is_contiguous()
```

Out[ ]:
```
False
```

In [ ]:
```
points = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]])
points
```

Out[ ]:
```
tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

In [ ]:
```
points.stride()
```

Out[ ]:
```
(2, 1)
```

In [ ]:
```
points_t = points.t()
points_t
```

Out[ ]:
```
tensor([[4., 5., 2.],
        [1., 3., 1.]])
```

In [ ]:
```
points_t.shape
```

Out[ ]:
```
torch.Size([2, 3])
```

In [ ]:
```
points_t.storage()
```

Out[ ]:
```
 4.0
 1.0
 5.0
 3.0
 2.0
 1.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 6]
```

In [ ]:
```
points_t.stride()
```

Out[ ]:
```
(1, 2)
```

In [ ]:
```
points_t_cont = points_t.contiguous()
points_t_cont
```

Out[ ]:
```
tensor([[4., 5., 2.],
        [1., 3., 1.]])
```

In [ ]:
```
points_t_cont.stride()
```

Out[ ]:
```
(3, 1)
```

In [ ]:
```
points_t_cont.storage()
```

Out[ ]:
```
 4.0
 5.0
 2.0
 1.0
 3.0
 1.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 6]
```

Tensor on GPU

In [ ]:
```
points_gpu = torch.tensor([[4.0, 1.0], [5.0, 3.0], [2.0, 1.0]], device='cuda
```

In [ ]:
```
# Copying tensor from cpu to gpu - returns a new tensor with same data
points_gpu = points.to(device='cuda')
```

In [ ]:
```
# To access different gpu cores - indexed from 0
points_gpu = points.to(device='cuda:0')
```

In [ ]:
```
points_gpu = 2 * points.to(device='cuda')
points_gpu
```

Out[ ]:
```
tensor([[ 8.,  2.],
        [10.,  6.],
        [ 4.,  2.]], device='cuda:0')
```

In [ ]:
```
points_gpu = points_gpu + 4
points_gpu
```

Out[ ]:
```
tensor([[12.,  6.],
        [14., 10.],
        [ 8.,  6.]], device='cuda:0')
```

In [ ]:
```
# Info does not flow from CPU to GPU unless specified
points_cpu = points_gpu.to(device='cpu')
```

In [ ]:
```
# Alternate to 'to' method.
points_gpu = points.cuda()
points_gpu = points.cuda(0)
points_cpu = points_gpu.cpu()
```

Numpy interoperability

In [ ]:
```
points = torch.ones(3, 4)
points_np = points.numpy()
points_np # operation happens in cpu
```

Out[ ]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]], dtype=float32)
```

In [ ]:
```
points = torch.from_numpy(points_np)
"""
“While the default numeric type in PyTorch is 32-bit floating-point, for NumP
it is 64-bit. As discussed in section 3.5.2, we usually want to use 32-bit
floating-points, so we need to make sure we have tensors of dtype
torch.float after converting.
"""
```

Out[ ]:
```
'\n“While the default numeric type in PyTorch is 32-bit floating-point, for
NumPy \nit is 64-bit. As discussed in section 3.5.2, we usually want to use
32-bit \nfloating-points, so we need to make sure we have tensors of dtype
\ntorch.float after converting. \n'
```

In [ ]:
```
# Serealize a tensor
torch.save(points, 'ourpoints.t')
# Alternatively
with open('ourpoints.t', 'wb') as f:
    torch.save(points, f)
```

In [ ]:
```
# loading out points tensor back
points = torch.load('ourpoints.t')

# Alternatively
with open('ourpoints.t', 'rb') as f:
    points = torch.load(f)
```

Real world data representation using tensors

In [ ]:
```
import imageio.v2 as imageio # Imageio handles different data tpes with a uni

img_arr = imageio.imread('/content/bobby.jpg')
img_arr.shape
```

Out[ ]:
```
(720, 1280, 3)
```

In [ ]:
```
# PyTorch modules dealing with image data require tensors to be laid out
# as C × H × W : channels, height, and width, respectively

img = torch.from_numpy(img_arr)
out = img.permute(2,0,1) # this operation does not make a copy of the tensor
# out uses the same underlying storage as img and only plays with size and s

# we store the images in a batch along the first dimension to obtain an N × C
# other efficient alternative is to use a stack to build up a tensor (pre allo
batch_size = 3
batch = torch.zeros(batch_size, 3, 256, 256, dtype = torch.uint8)
# each color is expected to be 8 bit integer
```

In [ ]:
```
import os
import imageio.v2 as imageio
data_dir = '/content/image-cats/'

# Initialize 'batch' as a list to append images.
# In a real scenario, you'd likely pre-allocate a tensor or use a DataLoader
batch = []

# List all .png files in the specified directory
filenames = [name for name in os.listdir(data_dir)
             if os.path.splitext(name)[-1] == '.png']

print(f"Found {len(filenames)} .png files in {data_dir}")

# Loop through each file, load, process, and add to batch
for i, filename in enumerate(filenames):
    img_path = os.path.join(data_dir, filename)
    img_arr = imageio.imread(img_path)

    # Convert NumPy array to PyTorch tensor
    img_t = torch.from_numpy(img_arr)

    # Permute dimensions from HWC (Height, Width, Channel) to CHW (Channel,
    # This is the standard format for PyTorch convolutional layers
```

```
    img_t = img_t.permute(2, 0, 1)

    # Take only the first 3 channels (e.g., handles RGBA images by taking RGB
    img_t = img_t[:3]

    # Add the processed image tensor to your batch list
    batch.append(img_t)

print(f"Successfully loaded {len(batch)} images into 'batch'.")
# You can inspect the shape of the first image if the batch is not empty
if batch:
   print(f"Shape of first image in batch: {batch[0].shape}")

# --- Snapshot 1: Raw Integer Batch (after stacking, before float conversion)
batch_raw_int = torch.stack(batch)
print("\n--- BEFORE NORMALIZATION (Raw Integer Batch) ---")
print(f"Shape: {batch_raw_int.shape}")
print(f"Data Type: {batch_raw_int.dtype}")
print(f"Min value: {batch_raw_int.min().item()}")
print(f"Max value: {batch_raw_int.max().item()}")
# Mean and Std for uint8 data might not be as intuitive, but we can compute
print(f"Mean value: {batch_raw_int.float().mean().item():.4f}") # Convert to
print(f"Std value: {batch_raw_int.float().std().item():.4f}")
```

```
Found 3 .png files in /content/image-cats/
Successfully loaded 3 images into 'batch'.
Shape of first image in batch: torch.Size([3, 256, 256])

--- BEFORE NORMALIZATION (Raw Integer Batch) ---
Shape: torch.Size([3, 3, 256, 256])
Data Type: torch.uint8
Min value: 0
Max value: 255
Mean value: 117.6989
Std value: 57.8453
```

In [ ]:
```
# Normalizing - Neural network works best with float tensors and when tensor
# is 0 to 1, -1 to 1. Casting to float is easy but normalizing is trickier.
# depends on what range of i/p we decide should lie between 0 and 1 or -1 to

# --- MODIFIED PART FOR NORMALIZATION ---
# 1. Stack the list of tensors into a single batch tensor
batch = torch.stack(batch)

# 2. Now 'batch' is a PyTorch tensor, so you can call .float() and divide
batch = batch.float()
batch /= 255.0
```

In [ ]:
```
# --- Snapshot 2: After 0-1 Normalization ---
batch_0_1_normalized = batch.clone() # Clone to save its current state

print("\n--- AFTER 0-1 NORMALIZATION ---")
print(f"Shape: {batch_0_1_normalized.shape}")
print(f"Data Type: {batch_0_1_normalized.dtype}")
print(f"Min value: {batch_0_1_normalized.min().item()}")
print(f"Max value: {batch_0_1_normalized.max().item()}")
```

```
print(f"Mean value: {batch_0_1_normalized.mean().item():.4f}")
print(f"Std value: {batch_0_1_normalized.std().item():.4f}")
```

```
--- AFTER 0-1 NORMALIZATION ---
Shape: torch.Size([3, 3, 256, 256])
Data Type: torch.float32
Min value: 0.0
Max value: 1.0
Mean value: 0.4616
Std value: 0.2268
```

In [ ]:
```
# Other method is to computer mean and stdof i/p data
n_channels = batch.shape[1]
for c in range(n_channels):
  mean = torch.mean(batch[:,c])
  std = torch.std(batch[:,c])
  batch[:,c] = (batch[:,c] - mean) / std
```

In [ ]:
```
# --- Snapshot 3: After Standardization ---
batch_standardized = batch.clone() # Clone to save its current state

print("\n--- AFTER STANDARDIZATION (Mean/Std Normalization) ---")
print(f"Shape: {batch_standardized.shape}")
print(f"Data Type: {batch_standardized.dtype}")
print(f"Min value: {batch_standardized.min().item():.4f}") # Min/Max can be
print(f"Max value: {batch_standardized.max().item():.4f}")
print(f"Mean value: {batch_standardized.mean().item():.4f}") # Overall mean
print(f"Std value: {batch_standardized.std().item():.4f}")   # Overall std (
```

```
--- AFTER STANDARDIZATION (Mean/Std Normalization) ---
Shape: torch.Size([3, 3, 256, 256])
Data Type: torch.float32
Min value: 0.0000
Max value: 1.0000
Mean value: 0.4616
Std value: 0.2268
```

In [ ]:
```
# To verify per-channel, you can loop again:
print(f"\n--- After Standardization (Per-Channel Verification) ---")
for c in range(n_channels):
    channel_mean = batch_standardized[:, c].mean().item()
    channel_std = batch_standardized[:, c].std().item()
    print(f"Channel {c}: Mean={channel_mean:.4f}, Std={channel_std:.4f}")
```

```
--- After Standardization (Per-Channel Verification) ---
Channel 0: Mean=0.5799, Std=0.2212
Channel 1: Mean=0.4493, Std=0.2068
Channel 2: Mean=0.3554, Std=0.1931
```

3D images: Volumetric Data

In CT scans, the intensity represents the density of the different parts of the body--lungs, fat, water, muscle, and bone, in order of increasing density--mapped from dark to bright when the CT scan is displayed on a clinical workstation.

The density at each point is computed from the amount of X-rays reaching a detector after crossing through the body, with some complex math to deconvolve the raw sensor data into the full volume.

CTs have only a single intensity channel, similar to a grayscale image. This means that often, the channel dimension is left out in native data formats

By stacking individual 2D slices into a 3D tensor, we can build volumetric data representing the 3D anatomy of a subject

In [ ]:
```
# Let's load a sample CT scan using the volread function in the imageio modu

import torch
import imageio.v2 as imageio


dir_path = '/content/volumetric-dicom'
vol_arr = imageio.volread(dir_path, 'DICOM')
vol_arr.shape
```

```
Reading DICOM (examining files): 1/99 files (1.0%)                    99/99 files
(100.0%)
  Found 1 correct series.
Reading DICOM (loading data): 46/99  (46.5%)                99/99  (100.0%)
```

Out[ ]:
```
(99, 512, 512)
```

In [ ]:
```
vol = torch.from_numpy(vol_arr).float()
vol = torch.unsqueeze(vol,0)

vol.shape
```

Out[ ]:
```
torch.Size([1, 99, 512, 512])
```

In [ ]:
```
import matplotlib.pyplot as plt # Import matplotlib for plotting
import numpy as np # Import numpy for array conversion
# 1. Basic Data Exploration (Numerical Properties)
print("\n--- Basic Numerical Properties ---")
print(f"Data Type: {vol.dtype}")
print(f"Min value: {vol.min().item()}")
print(f"Max value: {vol.max().item()}")
print(f"Mean value: {vol.mean().item():.4f}")
print(f"Std value: {vol.std().item():.4f}")
```

```
--- Basic Numerical Properties ---
Data Type: torch.float32
Min value: -1024.0
Max value: 3071.0
Mean value: -354.7244
Std value: 478.6137
```

In [ ]:
```
# 2. Visualize a single 2D slice (Axial View)
# Get rid of the batch dimension (squeeze)
# Then pick a slice from the depth/slice dimension (the first actual spatial
# Assuming shape is (1, D, H, W)
```

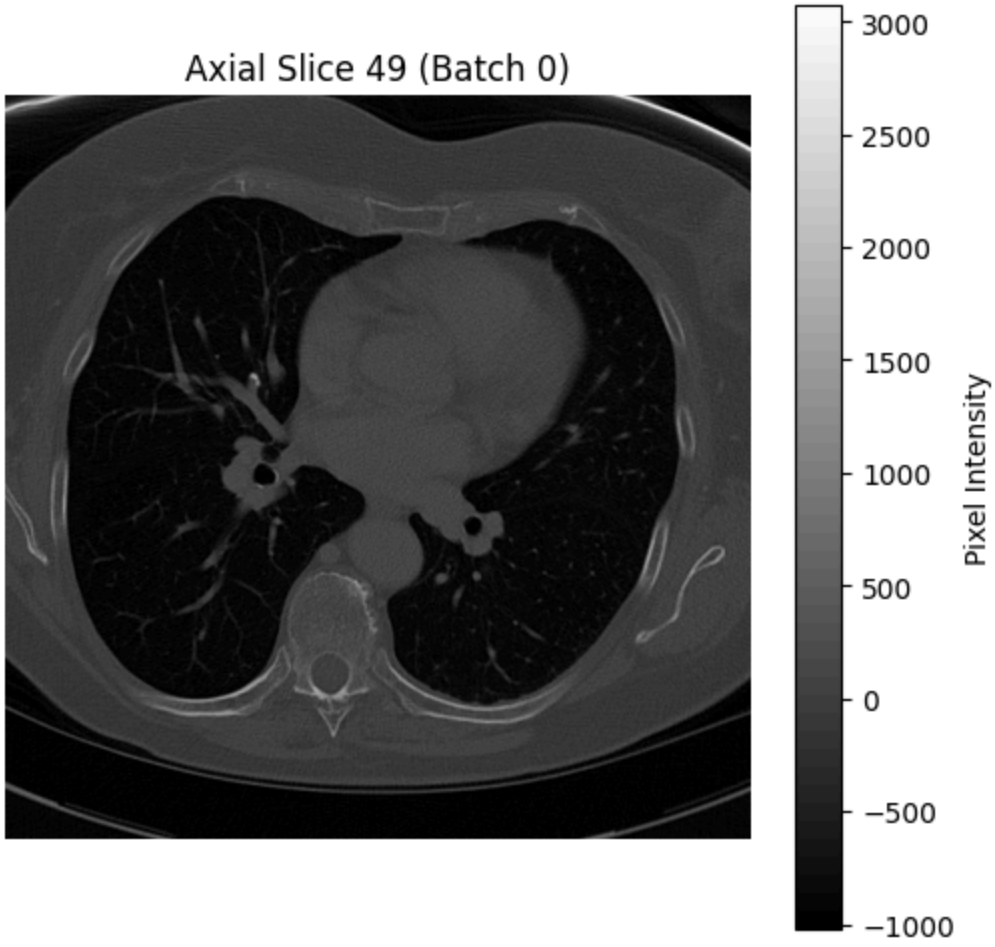
```
slice_index = vol.shape[1] // 2 # Pick the middle slice for example
slice_2d = vol[0, slice_index, :, :] # Select batch 0, chosen slice, all row

# If your 'vol' has a channel dimension (e.g., (1, 1, D, H, W)), you'd adjus
# slice_2d = vol[0, 0, slice_index, :, :] # Select batch 0, channel 0, chose
# Adjust '0' for channel index if you have multiple channels

print(f"\n--- Visualizing a single 2D slice (Slice {slice_index}) ---")
print(f"Shape of 2D slice: {slice_2d.shape}")

plt.figure(figsize=(6, 6))
plt.imshow(slice_2d.cpu().numpy(), cmap='gray') # Move to CPU, convert to Nu
plt.title(f"Axial Slice {slice_index} (Batch 0)")
plt.axis('off') # Hide axes for cleaner image
plt.colorbar(label='Pixel Intensity')
plt.show()
```

```
--- Visualizing a single 2D slice (Slice 49) ---
Shape of 2D slice: torch.Size([512, 512])
```



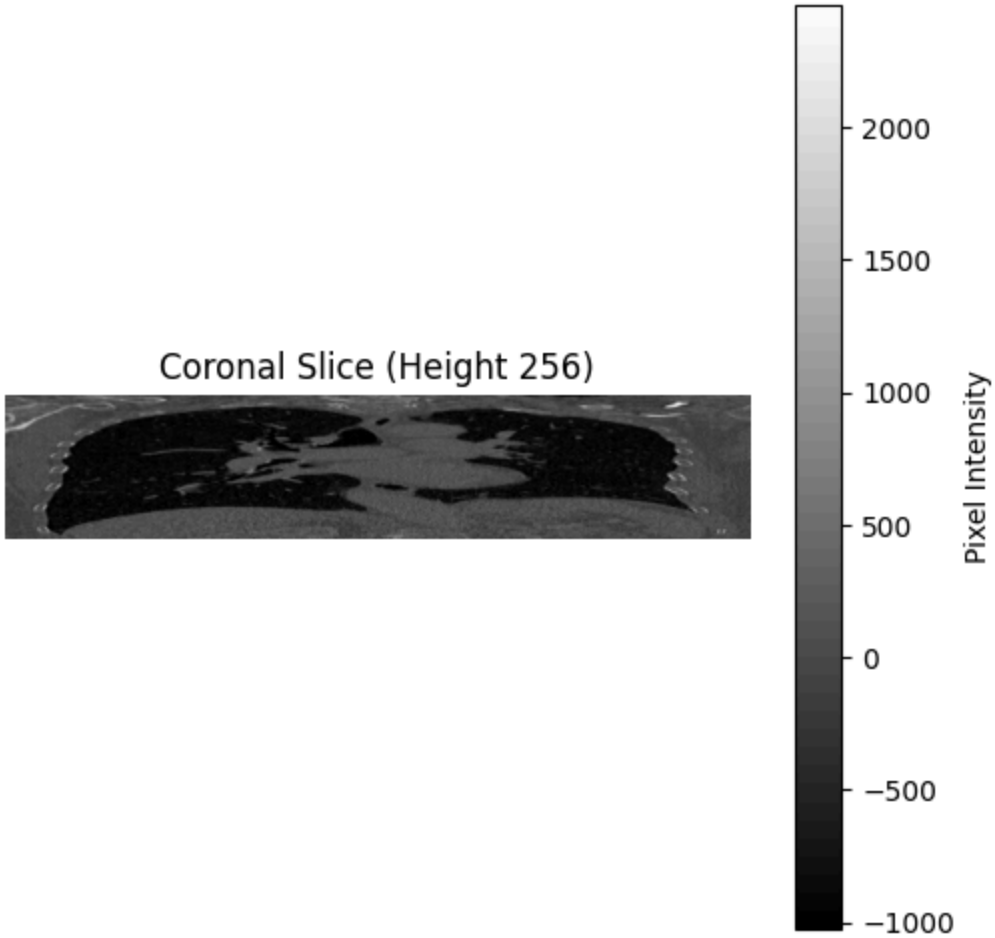
In [ ]:

```
# 3. Visualize other orthogonal 2D slices (Coronal / Sagittal Views)
# This gives a better sense of the 3D structure.

# Coronal slice (e.g., a slice along the 'height' dimension)
# Assuming shape (1, D, H, W)
coronal_slice_index = vol.shape[2] // 2 # Pick middle height slice
coronal_slice = vol[0, :, coronal_slice_index, :] # Select batch 0, all dept
```

```
print(f"\n--- Visualizing a Coronal Slice (Height {coronal_slice_index}) ---
print(f"Shape of Coronal Slice: {coronal_slice.shape}")
plt.figure(figsize=(6, 6))
plt.imshow(coronal_slice.cpu().numpy(), cmap='gray')
plt.title(f"Coronal Slice (Height {coronal_slice_index})")
plt.axis('off')
plt.colorbar(label='Pixel Intensity')
plt.show()
```

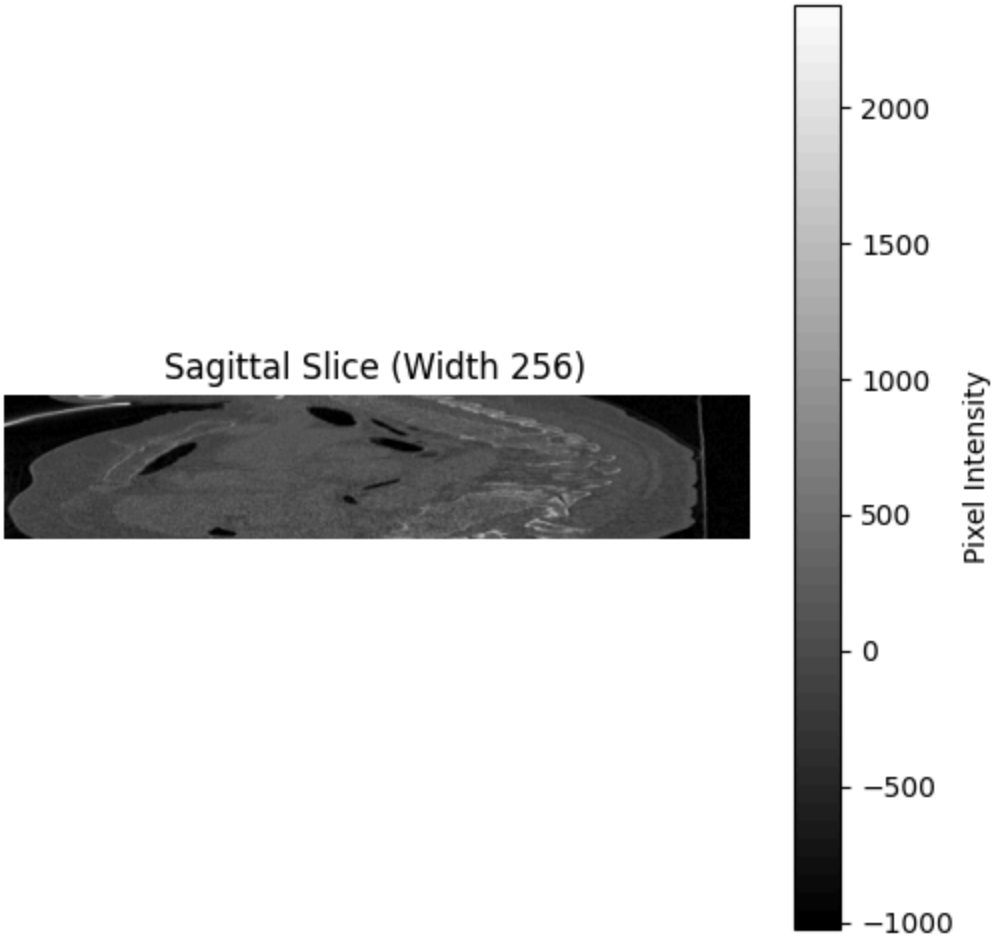
```
--- Visualizing a Coronal Slice (Height 256) ---
Shape of Coronal Slice: torch.Size([99, 512])
```



In [ ]:
```
# Sagittal slice (e.g., a slice along the 'width' dimension)
# Assuming shape (1, D, H, W)
sagittal_slice_index = vol.shape[3] // 2 # Pick middle width slice
sagittal_slice = vol[0, :, :, sagittal_slice_index] # Select batch 0, all de

print(f"\n--- Visualizing a Sagittal Slice (Width {sagittal_slice_index}) --
print(f"Shape of Sagittal Slice: {sagittal_slice.shape}")
plt.figure(figsize=(6, 6))
plt.imshow(sagittal_slice.cpu().numpy(), cmap='gray')
plt.title(f"Sagittal Slice (Width {sagittal_slice_index})")
plt.axis('off')
plt.colorbar(label='Pixel Intensity')
plt.show()
```

```
--- Visualizing a Sagittal Slice (Width 256) ---
Shape of Sagittal Slice: torch.Size([99, 512])
```



Tabular data

In [ ]:
```
import csv
wine_path = '/content/winequality-white.csv'

wineq_numpy = np.loadtxt(wine_path, dtype =np.float32, delimiter =';', skipro

wineq_numpy
```

Out[ ]:
```
array([[ 7.  ,  0.27,  0.36, ...,  0.45,  8.8 ,  6.  ],
       [ 6.3 ,  0.3 ,  0.34, ...,  0.49,  9.5 ,  6.  ],
       [ 8.1 ,  0.28,  0.4 , ...,  0.44, 10.1 ,  6.  ],
       ...,
       [ 6.5 ,  0.24,  0.19, ...,  0.46,  9.4 ,  6.  ],
       [ 5.5 ,  0.29,  0.3 , ...,  0.38, 12.8 ,  7.  ],
       [ 6.  ,  0.21,  0.38, ...,  0.32, 11.8 ,  6.  ]], dtype=float32)
```

In [ ]:
```
col_list = next(csv.reader(open(wine_path), delimiter=';'))
wineq_numpy.shape, col_list
```

Out[ ]:
```
((4898, 12),
 ['fixed acidity',
  'volatile acidity',
  'citric acid',
  'residual sugar',
  'chlorides',
  'free sulfur dioxide',
  'total sulfur dioxide',
  'density',
  'pH',
  'sulphates',
  'alcohol',
  'quality'])
```

In [ ]:
```
wineq = torch.from_numpy(wineq_numpy)

wineq.shape, wineq.dtype
```

Out[ ]:
```
(torch.Size([4898, 12]), torch.float32)
```

In [ ]:
```
data = wineq[:, :-1]
data, data.shape
```

Out[ ]:
```
(tensor([[ 7.0000,  0.2700,  0.3600,  ...,  3.0000,  0.4500,  8.8000],
         [ 6.3000,  0.3000,  0.3400,  ...,  3.3000,  0.4900,  9.5000],
         [ 8.1000,  0.2800,  0.4000,  ...,  3.2600,  0.4400, 10.1000],
         ...,
         [ 6.5000,  0.2400,  0.1900,  ...,  2.9900,  0.4600,  9.4000],
         [ 5.5000,  0.2900,  0.3000,  ...,  3.3400,  0.3800, 12.8000],
         [ 6.0000,  0.2100,  0.3800,  ...,  3.2600,  0.3200, 11.8000]]),
 torch.Size([4898, 11]))
```

In [ ]:
```
target = wineq[:, -1]
target, target.shape
```

Out[ ]:
```
(tensor([6., 6., 6.,  ..., 6., 7., 6.]), torch.Size([4898]))
```

In [ ]:
```
target = wineq[:, -1].long()
target
```

Out[ ]:
```
tensor([6, 6, 6,  ..., 6, 7, 6])
```

scatter_: This is a powerful PyTorch function used to "scatter" values from a source tensor (or a single value) into specific indices of the input tensor (which is target_onehot in this case). The underscore _ suffix in scatter_ indicates that it's an in-place operation, meaning it modifies target_onehot directly rather than returning a new tensor.

In [ ]:
```
target_onehot = torch.zeros(target.shape[0], 10)
target_onehot.scatter_(1, target.unsqueeze(1), 1.0)
```

Out[ ]:
```
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 1., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
```

In [ ]:
```
target_unsqueezed = target.unsqueeze(1)
target_unsqueezed
```

Out[ ]:
```
tensor([[6],
        [6],
        [6],
        ...,
        [6],
        [7],
        [6]])
```

In [ ]:
```
data_mean = torch.mean(data, dim=0)
data_mean
```

Out[ ]:
```
tensor([6.8548e+00, 2.7824e-01, 3.3419e-01, 6.3914e+00, 4.5772e-02, 3.5308e
+01,
        1.3836e+02, 9.9403e-01, 3.1883e+00, 4.8985e-01, 1.0514e+01])
```

In [ ]:
```
data_var = torch.var(data, dim=0)
data_var
```

Out[ ]:
```
tensor([7.1211e-01, 1.0160e-02, 1.4646e-02, 2.5726e+01, 4.7733e-04, 2.8924e
+02,
        1.8061e+03, 8.9455e-06, 2.2801e-02, 1.3025e-02, 1.5144e+00])
```

In [ ]:
```
data_normalized = (data - data_mean) / torch.sqrt(data_var)
data_normalized
```

Out[ ]:
```
tensor([[ 1.7208e-01, -8.1761e-02,  2.1326e-01,  ..., -1.2468e+00,
         -3.4915e-01, -1.3930e+00],
        [-6.5743e-01,  2.1587e-01,  4.7996e-02,  ...,  7.3995e-01,
          1.3422e-03, -8.2419e-01],
        [ 1.4756e+00,  1.7450e-02,  5.4378e-01,  ...,  4.7505e-01,
         -4.3677e-01, -3.3663e-01],
        ...,
        [-4.2043e-01, -3.7940e-01, -1.1915e+00,  ..., -1.3130e+00,
         -2.6153e-01, -9.0545e-01],
        [-1.6054e+00,  1.1666e-01, -2.8253e-01,  ...,  1.0049e+00,
         -9.6251e-01,  1.8574e+00],
        [-1.0129e+00, -6.7703e-01,  3.7852e-01,  ...,  4.7505e-01,
         -1.4882e+00,  1.0448e+00]])
```

In [ ]:
```
bad_indexes = target <= 3
bad_indexes.shape, bad_indexes.dtype, bad_indexes.sum()
```

Out[ ]:
```
(torch.Size([4898]), torch.bool, tensor(20))
```

In [ ]:
```
bad_data = data[bad_indexes]
bad_data.shape
```

Out[ ]:
```
torch.Size([20, 11])
```

In [ ]:
```
bad_data = data[target <= 3]
mid_data = data[(target > 3) & (target < 7)]
good_data = data[target >= 7]

bad_mean = torch.mean(bad_data, dim=0)
mid_mean = torch.mean(mid_data, dim=0)
good_mean = torch.mean(good_data, dim=0)

for i, args in enumerate(zip(col_list, bad_mean, mid_mean, good_mean)):
    print('{:2} {:20} {:6.2f} {:6.2f} {:6.2f}'.format(i, *args))
```

```
 0 fixed acidity           7.60   6.89   6.73
 1 volatile acidity        0.33   0.28   0.27
 2 citric acid             0.34   0.34   0.33
 3 residual sugar          6.39   6.71   5.26
 4 chlorides               0.05   0.05   0.04
 5 free sulfur dioxide    53.33  35.42  34.55
 6 total sulfur dioxide  170.60 141.83 125.25
 7 density                 0.99   0.99   0.99
 8 pH                      3.19   3.18   3.22
 9 sulphates               0.47   0.49   0.50
10 alcohol                10.34  10.26  11.42
```

torch.lt() stands for "less than." It creates a boolean tensor where True indicates that the total_sulfur_data for that sample is less than the total_sulfur_threshold

In [ ]:
```
total_sulfur_threshold = 141.83
total_sulfur_data = data[:,6]
predicted_indexes = torch.lt(total_sulfur_data, total_sulfur_threshold)
predicted_indexes.shape, predicted_indexes.dtype, predicted_indexes.sum()
```

Out[ ]:
```
(torch.Size([4898]), torch.bool, tensor(2727))
```

In [ ]:
```
actual_indexes = target > 5

actual_indexes.shape, actual_indexes.dtype, actual_indexes.sum()
```

Out[ ]:
```
(torch.Size([4898]), torch.bool, tensor(3258))
```

In [ ]:
```
n_matches = torch.sum(actual_indexes & predicted_indexes).item()
n_predicted = torch.sum(predicted_indexes).item()
n_actual = torch.sum(actual_indexes).item()

n_matches, n_matches / n_predicted, n_matches / n_actual
```

Out[ ]:
```
(2018, 0.74000733406674, 0.6193984039287906)
```

Working with Time series

Our goal will be to take a flat, 2D dataset and transform it into a 3D one, as shown in figure.

In [ ]:
```
import numpy as np
import torch

bikes_numpy = np.loadtxt('/content/hour-fixed.csv',
                         dtype=np.float32,
                         delimiter =',',
                         skiprows=1,
                         converters={1:lambda x:float(x[8:10])})
# x[8:10]: This is string slicing. It extracts a substring from x starting
# at index 8 and going up to (but not including) index 10.

bikes = torch.from_numpy(bikes_numpy)
bikes
```

Out[ ]:
```
tensor([[1.0000e+00, 1.0000e+00, 1.0000e+00,  ..., 3.0000e+00, 1.3000e+01,
         1.6000e+01],
        [2.0000e+00, 1.0000e+00, 1.0000e+00,  ..., 8.0000e+00, 3.2000e+01,
         4.0000e+01],
        [3.0000e+00, 1.0000e+00, 1.0000e+00,  ..., 5.0000e+00, 2.7000e+01,
         3.2000e+01],
        ...,
        [1.7377e+04, 3.1000e+01, 1.0000e+00,  ..., 7.0000e+00, 8.3000e+01,
         9.0000e+01],
        [1.7378e+04, 3.1000e+01, 1.0000e+00,  ..., 1.3000e+01, 4.8000e+01,
         6.1000e+01],
        [1.7379e+04, 3.1000e+01, 1.0000e+00,  ..., 1.2000e+01, 3.7000e+01,
         4.9000e+01]])
```

For every hour, the dataset reports the following variables:

Index of record: instant

Day of month: day

Season: season (1: spring, 2: summer, 3: fall, 4: winter)

Year: yr (0: 2011, 1: 2012)

Month: mnth (1 to 12)”

Hour: hr (0 to 23)

Holiday status: holiday

Day of the week: weekday

Working day status: workingday

Weather situation: weathersit (1: clear, 2:mist, 3: light rain/snow, 4: heavy rain/snow)

```
Temperature in °C: temp

Perceived temperature in °C: atemp

Humidity: hum

Wind speed: windspeed

Number of casual users: casual

Number of registered users: registered

Count of rental bikes: cnt
```

The existence of an ordering gives us the opportunity to exploit causal relationships across time. For instance, it allows us to predict bike rides at one time based on the fact that it was raining at an earlier time. For the time being, we're going to focus on learning how to turn our bike-sharing dataset into something that our neural network will be able to ingest in fixed-size chunks

This neural network model will need to see a number of sequences of values for each different quantity, such as ride count, time of day, temperature, and weather conditions: N parallel sequences of size C. C stands for channel, in neural network parlance, and is the same as column for 1D data like we have here. The N dimension represents the time axis, here one entry per hour.

We might want to break up the two-year dataset into wider observation periods, like days. This way we'll have N (for number of samples) collections of C sequences of length L. In other words, our time series dataset would be a tensor of dimension 3 and shape N × C × L. The C would remain our 17 channels, while L would be 24: 1 per hour of the day. There's no particular reason why we must use chunks of 24 hours, though the general daily rhythm is likely to give us patterns we can exploit for predictions. We could also use 7 × 24 = 168 hour blocks to chunk by week instead, if we desired.

In [ ]:
```
bikes.shape, bikes.stride()
```

Out[ ]:
```
(torch.Size([17520, 17]), (17, 1))
```

That's 17,520 hours, 17 columns. Now let's reshape the data to have 3 axes--day, hour, and then our 17 columns.

Calling view on a tensor returns a new tensor that changes the number of dimensions and the striding information, without changing the storage.

bikes.view(...): The view() method in PyTorch creates a new tensor that shares the same underlying data with the original tensor, but with a different shape. This operation is only possible if the tensor is contiguous in memory.

-1: When you use -1 in view(), PyTorch automatically calculates the size of that dimension based on the total number of elements in the tensor and the sizes of the other specified dimensions.

24: This sets the second dimension (the middle dimension) to 24. Given this is time-series data for bikes, this likely represents 24 hours in a day.

bikes.shape[1]: This gets the size of the second dimension of the original bikes tensor, which is 17. This means the last dimension will represent the 17 features for each hour.

Calculations for daily_bikes.shape:

Original bikes shape: (17520, 17) Total number of elements in bikes: 17520 * 17 = 297840

New shape specified: (-1, 24, 17)

To find the -1 dimension, divide the total number of elements by the product of the other dimensions: 297840 / (24 * 17) = 297840 / 408 = 730.

So, the expected shape of daily_bikes would be (730, 24, 17).

This means you're reshaping the data into 730 "days", where each "day" has 24 "hours", and each "hour" has 17 "features".

In [ ]:
```
daily_bikes = bikes.view(-1,24, bikes.shape[1])
daily_bikes.shape, daily_bikes.stride()
```

Out[ ]:
```
(torch.Size([730, 24, 17]), (408, 17, 1))
```

We see that the rightmost dimension is the number of columns in the original dataset. Then, in the middle dimension, we have time, split into chunks of 24 sequential hours. In other words, we now have N sequences of L hours in a day, for C channels. To get to our desired N × C × L ordering, we need to transpose the tensor

In [ ]:
```
daily_bikes = daily_bikes.transpose(1, 2)
daily_bikes.shape, daily_bikes.stride()
```

Out[ ]:
```
(torch.Size([730, 17, 24]), (408, 1, 17))
```

bikes[:24] likely extracts all the data for the very first day in your dataset.

.long(): This method casts the data type of the selected 24 rows to a 64-bit integer (torch.int64). This is often done when working with categorical data or indices that need to be integer types for operations like one-hot encoding or as input to certain neural network layers.

first_day becomes a new tensor containing all the feature data (17 columns) for the first 24 hours, with its values converted to long integers. Its shape would be (24, 17).

4: This number indicates that you are expecting 4 distinct categories for the feature you are about to one-hot encode. In this context, given the variable name weather_onehot, these 4 categories represent different weather conditions.

first_day[:,9] first_day: This refers to the tensor we just created, containing the first 24 hours of data.

[:,9]: This is another slicing operation: The : before the comma means "select all rows."

The 9 after the comma means "select the column at index 9."

In [ ]:
```
first_day = bikes[:24].long()
weather_onehot = torch.zeros(first_day.shape[0], 4)
first_day[:,9]
```

Out[ ]:
```
tensor([1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 2, 2,
        2, 2])
```

Then we scatter ones into our matrix according to the corresponding level at each row.

weather_onehot.scatter_(:

This is the tensor we initialized with zeros earlier, which has a shape of (24, 4). This is where the one-hot encoded values will be placed. The underscore _ in scatter_ indicates that this is an in-place operation, meaning it modifies weather_onehot directly.

dim=1:

This specifies the dimension along which to scatter the values. dim=1 refers to the columns of the weather_onehot tensor. So, for each row, the 1.0 will be placed in one of the 4 columns

index=first_day[:,9].unsqueeze(1).long() - 1:

This is the most critical part, as it determines where to place the 1.0. first_day[:,9]: As we discussed, this extracts the 10th column (index 9) from the first_day tensor. This column contains the raw numerical codes for the weather condition for each of the 24 hours. For example, if a weather code is 1, 2, 3, or 4.

.unsqueeze(1): This adds a new dimension of size 1 at index 1 to the first_day[:,9] tensor. If first_day[:,9] has a shape of (24,), unsqueeze(1) transforms it to (24, 1). This is necessary because scatter_ expects the index tensor to have specific dimensions that align with the input tensor (weather_onehot) and the dim argument.

.long(): This ensures that the tensor containing the indices is of type torch.long, which is required by scatter_ for index tensors.

-1: This is a very important adjustment! If your weather codes in first_day[:,9] are 1-based (e.g., 1, 2, 3, 4 representing different weather types), but PyTorch's tensor indexing is 0-

based (meaning valid indices are 0, 1, 2, 3), then subtracting 1 converts the 1-based codes into the correct 0-based indices for placing the 1.0 in the weather_onehot tensor. For example, if a weather code is 1, it becomes 0 and the 1.0 goes in the first column; if it's 4, it becomes 3 and the 1.0 goes in the fourth column.

value=1.0:

This is the value that will be placed into weather_onehot at the specified indices. For one-hot encoding, this value is typically 1.0.

In [ ]:
```
weather_onehot.scatter_(
                        dim=1,
                        index=first_day[:,9].unsqueeze(1).long() - 1, value=1
```

Out[ ]:
```
tensor([[1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.]])
```

(bikes[:24], weather_onehot): This is the tuple of tensors that are being concatenated

bikes[:24]: This refers to the data for the first 24 hours (the first "day") of your dataset, containing all 17 original features. Its shape is (24, 17). weather_onehot: This is the one-hot encoded weather data for those same 24 hours. Its shape is (24, 4).

1: This is the dim argument. It specifies that the concatenation should happen along dimension 1 (the columns). Since both bikes[:24] and weather_onehot have 24 rows (dim=0), they can be joined side-by-side along their columns.

[:1] means "select rows from the beginning up to (but not including) index 1." In simple terms, it selects only the first row of the concatenated tensor.

The new one-hot-encoded columns are appended to the original dataset. For cat to succeed, it is required that the tensors have the same size along the other dimensions--the row dimension

We could have done the same with the reshaped daily_bikes tensor. Remember that it is shaped (B, C, L), where L = 24. We first create the zero tensor, with the same B and L, but with the number of additional columns as C

In [ ]:
```
torch.cat((bikes[:24], weather_onehot), 1)[:1]
```

Out[ ]:
```
tensor([[ 1.0000,  1.0000,  1.0000,  0.0000,  1.0000,  0.0000,  0.0000,  6.
0000,
          0.0000,  1.0000,  0.2400,  0.2879,  0.8100,  0.0000,  3.0000, 13.
0000,
         16.0000,  1.0000,  0.0000,  0.0000,  0.0000]])
```

daily_bikes.shape[0]: This is the size of the first dimension of your daily_bikes tensor. As we established, daily_bikes has a shape of (730, 24, 17), so daily_bikes.shape[0] is 730. This will be the number of "days" in your new tensor.

4: This number represents the number of distinct weather categories you are one-hot encoding (e.g., clear, cloudy, light rain, heavy rain). So, the second dimension will be 4.

daily_bikes.shape[2]: This refers to the size of the third dimension of your daily_bikes tensor. From daily_bikes.shape being (730, 24, 17), daily_bikes.shape[2] is 17 (the number of features per hour).

In [ ]:
```
daily_weather_onehot = torch.zeros(daily_bikes.shape[0], 4,
                                   daily_bikes.shape[2])
daily_weather_onehot.shape
```

Out[ ]:
```
torch.Size([730, 4, 24])
```

This code snippet continues the one-hot encoding process for weather conditions, but now it's applying it to your entire daily_bikes tensor, structuring the output in the daily_weather_onehot tensor you initialized.

daily_weather_onehot.scatter_(:

This is the target tensor you created with zeros, which has the shape (730, 4, 24). The scatter_ operation will modify this tensor in-place.

1 (for dim):

This specifies the dimension along which the scattering occurs. dim=1 means that for each combination of "day" (dimension 0) and "hour" (dimension 2), a 1.0 will be placed in one of the 4 "weather category" slots (dimension 1).

daily_bikes[:,9,:].long().unsqueeze(1) - 1 (for index):

daily_bikes[:,9,:]:
daily_bikes has a shape of (730, 24, 17) (days, hours, features).

for the first dimension selects all 730 days.

9 for the second dimension selects the 10th feature column (index=9), which, as we identified, contains the weather condition codes.

for the third dimension selects all 24 hours within each day. This expression extracts all the raw weather codes for every hour of every day. The resulting tensor will have a shape of (730, 24). Each element in this (730, 24) tensor is a weather code (e.g., 1, 2, 3, or 4).

.long(): Converts these extracted weather codes to torch.long data type, which is necessary for them to be used as indices.

.unsqueeze(1): This is crucial for matching the dimensions for the scatter_ operation. It adds a new dimension of size 1 at index 1 to the (730, 24) tensor, transforming it into (730, 1, 24). This new dimension aligns with the dim=1 argument of scatter_, allowing it to correctly map the weather codes to the 4 weather categories.

-1: As before, this adjusts the 1-based weather codes (e.g., 1, 2, 3, 4) from your dataset to 0-based indices (0, 1, 2, 3) that PyTorch uses for indexing. 1.0 (for value):

This is the value that will be scattered (placed) at the calculated indices.

In [ ]:
```
daily_weather_onehot.scatter_(
    1, daily_bikes[:,9,:].long().unsqueeze(1) - 1, 1.0)
daily_weather_onehot.shape
```

Out[ ]:
```
torch.Size([730, 4, 24])
```

In [ ]:
```
daily_bikes = torch.cat((daily_bikes, daily_weather_onehot), dim=1)
```

In [ ]:
```
daily_bikes[:, 9, :] = (daily_bikes[:, 9, :] - 1.0) / 3.0
```

daily_bikes[:, 9, :] = (daily_bikes[:, 9, :] - 1.0) / 3.0

Purpose: This line performs a min-max-like normalization on the feature at index 9 (which is likely the original numerical weather code, if it hasn't been removed after one-hot encoding, or it could be another categorical feature that still needs scaling).

daily_bikes[:, 9, :]: Selects all data for the feature at index 9 across all days and all hours.

(daily_bikes[:, 9, :] - 1.0) / 3.0: This mathematical operation scales the values. If the original values at index 9 were 1, 2, 3, 4 (common for weather codes), then: Subtracting 1.0 shifts them to 0.0, 1.0, 2.0, 3.0.

Dividing by 3.0 scales them to 0.0, 0.333..., 0.666..., 1.0. This transforms the numerical weather codes into a range between 0 and 1, which can help neural networks learn more

effectively.

In [ ]:
```
temp = daily_bikes[:, 10, :]
temp_min = torch.min(temp)
temp_max = torch.max(temp)
daily_bikes[:, 10, :] = ((daily_bikes[:, 10, :] - temp_min)
                         / (temp_max - temp_min))
```

This block performs Z-score Standardization (also known as Standardization) on the already Min-Max scaled temperature feature. It transforms the data to have a mean of 0 and a standard deviation of 1.

temp = daily_bikes[:, 10, :]: Re-extracts the temperature values (which are now Min-Max scaled).

torch.mean(temp) and torch.std(temp): Calculate the mean and standard deviation of the (now Min-Max scaled) temperature feature.

daily_bikes[:, 10, :] = ((...) / (...)): Applies the Z-score standardization formula: (value - mean) / standard_deviation.

In [ ]:
```
temp = daily_bikes[:, 10, :]
daily_bikes[:, 10, :] = ((daily_bikes[:, 10, :] - torch.mean(temp))
                         / torch.std(temp))
```

Representing text

In [ ]:
```
with open('/content/1342-0.txt', encoding='utf8') as f:
    text = f.read()
```

we need to parse through the characters in the text and provide a one-hot encoding for each of them. Each character will be represented by a vector of length equal to the number of different characters in the encoding.

This vector will contain all zeros except a one at the index corresponding to the location of the character in the encoding.

In [ ]:
```
lines = text.split('\n')
line = lines[200]
line
```

Out[ ]:
```
'"Impossible, Mr. Bennet, impossible, when I am not acquainted with him'
```

letter_t = torch.zeros(len(line), 128): This initializes a new PyTorch tensor called letter_t filled with zeros.

len(line): The first dimension of the tensor is the number of characters in the line (each character will get its own row).

128: The second dimension is 128. This suggests that the encoding will be based on the ASCII character set, which has 128 standard characters (from 0 to 127). Each of these 128 positions represents a unique character.

This tensor will store the one-hot encoding of each character in the line. For example, if the line has 5 characters, letter_t will be (5, 128)

In [ ]:
```
letter_t = torch.zeros(len(line), 128)
letter_t.shape
```

Out[ ]:
```
torch.Size([70, 128])
```

Note that letter_t holds a one-hot-encoded character per row. Now we just have to set a one on each row in the correct position so that each row represents the correct character. The index where the one has to be set corresponds to the index of the character in the encoding.

ord(letter): This built-in Python function returns the ASCII (or Unicode) numerical value of a character. For example, ord('a') is 97, ord(' ') is 32.

if ord(letter) < 128 else 0: This is a conditional expression. It checks if the character's ASCII value is within the standard ASCII range (0-127). If it is, that ASCII value becomes the letter_index.

If the character is a non-ASCII character (e.g., a special symbol outside the first 128 ASCII characters), its index is set to 0 (a fallback for unknown or extended characters).

letter_t[i][letter_index] = 1: This is the core of the one-hot encoding. For the i-th character in the line, it sets the value at its specific letter_index position in letter_t to 1. All other positions in that row remain 0

In [ ]:
```
for i, letter in enumerate(line.lower().strip()):
  letter_index = ord(letter) if ord(letter) < 128 else 0
  letter_t[i][letter_index] = 1
```

word_list = input_str.lower().replace('\n',' ').split():

input_str.lower(): Converts the entire input string to lowercase.

.replace('\n',' '): Replaces any newline characters with spaces.

.split(): Splits the string into a list of words, using whitespace as the delimiter.

word_list = [word.strip(punctuation) for word in word_list]: This is a list comprehension that iterates through each word in the word_list. For each word, word.strip(punctuation) removes any leading or trailing punctuation characters defined in punctuation. This ensures words like "hello." become "hello".

return word_list: Returns the list of cleaned, lowercase words.

words_in_line = clean_words(line): This calls the clean_words function, passing the line (the 201st line from the file) as input. The result is a list of cleaned, lowercase words from that specific line.

line, words_in_line: This prints both the original line string and the words_in_line list to show the effect of the cleaning.

In [ ]:
```
def clean_words(input_str):
  punctuation = '.,;:"!?”“_-'
  word_list = input_str.lower().replace('\n',' ').split()
  word_list = [word.strip(punctuation) for word in word_list]
  return word_list

words_in_line = clean_words(line)
line, words_in_line
```

Out[ ]:
```
('“Impossible, Mr. Bennet, impossible, when I am not acquainted with him',
 ['impossible',
  'mr',
  'bennet',
  'impossible',
  'when',
  'i',
  'am',
  'not',
  'acquainted',
  'with',
  'him'])
```

word_list = sorted(set(clean_words(text))): This is a crucial step for building your vocabulary:

clean_words(text): First, the clean_words function is applied to the entire text from the file, producing a long list of all cleaned words.

set(...): Converts this list into a set. A set automatically removes all duplicate words, leaving only unique words. This forms your vocabulary.

sorted(...): Sorts the unique words alphabetically. This ensures that the word-to-index mapping will be consistent and reproducible.

word2index_dict = {word: i for (i, word) in enumerate(word_list)}: This is a dictionary comprehension that creates a mapping from each unique word in your word_list to a unique integer index (i).

enumerate(word_list) assigns a sequential index to each word in the sorted unique word_list. This word2index_dict is your vocabulary lookup table.

len(word2index_dict), word2index_dict['impossible']: len(word2index_dict): Prints the total number of unique words (the size of your vocabulary).

word2index_dict['impossible']: Demonstrates how to look up the integer index for a specific word, like "impossible".

In [ ]:
```
word_list = sorted(set(clean_words(text)))
word2index_dict = {word: i for (i, word) in enumerate(word_list)}

len(word2index_dict), word2index_dict['impossible']
```

Out[ ]:
```
(7261, 3394)
```

word_t = torch.zeros(len(words_in_line), len(word2index_dict)): This initializes a new PyTorch tensor word_t filled with zeros.

len(words_in_line): The first dimension is the number of words in the specific line we're processing (e.g., if the line has 10 words, this will be 10 rows).

len(word2index_dict): The second dimension is the total size of your vocabulary (number of unique words from the entire text).

This tensor will store the one-hot encoding of each word in the line. for i, word in enumerate(words_in_line):: This loop iterates through each word in the words_in_line list.

enumerate() provides both the position (i) and the word itself. word_index = word2index_dict[word]: Looks up the unique integer index for the current word using the word2index_dict created earlier.

word_t[i][word_index] = 1: This is the core of the one-hot encoding. For the i-th word in the line, it sets the value at its specific word_index position in word_t to 1. All other positions in that row remain 0.

print('{:2} {:4} {}'.format(i, word_index, word)): This line prints the sequential index of the word in the line (i), its corresponding numerical index from the vocabulary (word_index), and the word itself. This helps to visualize the encoding process.

print(word_t.shape): Prints the final shape of the word_t tensor, which will be (number_of_words_in_line, vocabulary_size).

In [ ]:
```
word_t = torch.zeros(len(words_in_line), len(word2index_dict))
for i, word in enumerate(words_in_line):
    word_index = word2index_dict[word]
    word_t[i][word_index] = 1
    print('{:2} {:4} {}'.format(i, word_index, word))

print(word_t.shape)
```

```
 0 3394 impossible
 1 4305 mr
 2  813 bennet
 3 3394 impossible
 4 7078 when
 5 3315 i
 6  415 am
 7 4436 not
 8  239 acquainted
 9 7148 with
10 3215 him
torch.Size([11, 7261])
```

Converting to PDF

In [10]:
```
!sudo apt-get update
!sudo apt-get install texlive-xetex pandoc
```

```
0% [Working]

Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86
_64  InRelease
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRe
lease
Hit:9 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Hit:10 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[1,553 kB]
Fetched 1,682 kB in 3s (578 kB/s)
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'htt
ps://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide i
t (sources.list entry misspelt?)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
```

In [19]:
```
!sudo apt-get update
!sudo apt-get install -y chromium-browser
```

```
0% [Working]

Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86
_64  InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRe
lease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repository 'htt
ps://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide i
t (sources.list entry misspelt?)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apparmor libfuse3-3 libudev1 snapd squashfs-tools systemd-hwe-hwdb udev
Suggested packages:
  apparmor-profiles-extra apparmor-utils fuse3 zenity | kdialog
The following NEW packages will be installed:
  apparmor chromium-browser libfuse3-3 snapd squashfs-tools systemd-hwe-hwdb
  udev
The following packages will be upgraded:
  libudev1
1 upgraded, 7 newly installed, 0 to remove and 34 not upgraded.
Need to get 30.3 MB of archives.
After this operation, 123 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 apparmor amd6
4 3.0.4-2ubuntu2.4 [598 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 squashfs-tools amd64
1:4.5-3build1 [159 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libudev1 amd6
4 249.11-0ubuntu3.15 [76.6 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 udev amd64 24
9.11-0ubuntu3.15 [1,557 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 libfuse3-3 amd64 3.1
0.5-1build1 [81.2 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 snapd amd64
2.67.1+22.04 [27.8 MB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 chromium-
browser amd64 1:85.0.4183.83-0ubuntu2.22.04.1 [49.2 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 systemd-hwe-h
wdb all 249.11.5 [3,228 B]
Fetched 30.3 MB in 4s (8,098 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based fro
ntend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78,
<> line 8.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
```

```
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package apparmor.
(Reading database ... 163160 files and directories currently installed.)
Preparing to unpack .../apparmor_3.0.4-2ubuntu2.4_amd64.deb ...
Unpacking apparmor (3.0.4-2ubuntu2.4) ...
Selecting previously unselected package squashfs-tools.
Preparing to unpack .../squashfs-tools_1%3a4.5-3build1_amd64.deb ...
Unpacking squashfs-tools (1:4.5-3build1) ...
Preparing to unpack .../libudev1_249.11-0ubuntu3.15_amd64.deb ...
Unpacking libudev1:amd64 (249.11-0ubuntu3.15) over (249.11-0ubuntu3.12) ...
Setting up libudev1:amd64 (249.11-0ubuntu3.15) ...
Selecting previously unselected package udev.
(Reading database ... 163360 files and directories currently installed.)
Preparing to unpack .../udev_249.11-0ubuntu3.15_amd64.deb ...
Unpacking udev (249.11-0ubuntu3.15) ...
Selecting previously unselected package libfuse3-3:amd64.
Preparing to unpack .../libfuse3-3_3.10.5-1build1_amd64.deb ...
Unpacking libfuse3-3:amd64 (3.10.5-1build1) ...
Selecting previously unselected package snapd.
Preparing to unpack .../snapd_2.67.1+22.04_amd64.deb ...
Unpacking snapd (2.67.1+22.04) ...
Setting up apparmor (3.0.4-2ubuntu2.4) ...
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based fro
ntend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 7
8.)
debconf: falling back to frontend: Readline
Created symlink /etc/systemd/system/sysinit.target.wants/apparmor.service → /
lib/systemd/system/apparmor.service.
Setting up squashfs-tools (1:4.5-3build1) ...
Setting up udev (249.11-0ubuntu3.15) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Setting up libfuse3-3:amd64 (3.10.5-1build1) ...
Setting up snapd (2.67.1+22.04) ...
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.apparmor.se
rvice → /lib/systemd/system/snapd.apparmor.service.
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.autoimport.
service → /lib/systemd/system/snapd.autoimport.service.
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.core-fixup.s
ervice → /lib/systemd/system/snapd.core-fixup.service.
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.recovery-ch
ooser-trigger.service → /lib/systemd/system/snapd.recovery-chooser-trigger.se
rvice.
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.seeded.serv
ice → /lib/systemd/system/snapd.seeded.service.
Created symlink /etc/systemd/system/cloud-final.service.wants/snapd.seeded.ser
vice → /lib/systemd/system/snapd.seeded.service.
Unit /lib/systemd/system/snapd.seeded.service is added as a dependency to a n
on-existent unit cloud-final.service.
Created symlink /etc/systemd/system/multi-user.target.wants/snapd.service → /
lib/systemd/system/snapd.service.
Created symlink /etc/systemd/system/timers.target.wants/snapd.snap-repair.tim
er → /lib/systemd/system/snapd.snap-repair.timer.
```

```
Created symlink /etc/systemd/system/sockets.target.wants/snapd.socket → /lib/
systemd/system/snapd.socket.
Created symlink /etc/systemd/system/final.target.wants/snapd.system-shutdown.s
ervice → /lib/systemd/system/snapd.system-shutdown.service.
Selecting previously unselected package chromium-browser.
(Reading database ... 163589 files and directories currently installed.)
Preparing to unpack .../chromium-browser_1%3a85.0.4183.83-0ubuntu2.22.04.1_am
d64.deb ...
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based fro
ntend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 7
8.)
debconf: falling back to frontend: Readline
=> Installing the chromium snap
==> Checking connectivity with the snap store
===> System doesn't have a working snapd, skipping
Unpacking chromium-browser (1:85.0.4183.83-0ubuntu2.22.04.1) ...
Selecting previously unselected package systemd-hwe-hwdb.
Preparing to unpack .../systemd-hwe-hwdb_249.11.5_all.deb ...
Unpacking systemd-hwe-hwdb (249.11.5) ...
Setting up systemd-hwe-hwdb (249.11.5) ...
Setting up chromium-browser (1:85.0.4183.83-0ubuntu2.22.04.1) ...
update-alternatives: using /usr/bin/chromium-browser to provide /usr/bin/x-ww
w-browser (x-www-browser) in auto mode
update-alternatives: using /usr/bin/chromium-browser to provide /usr/bin/gnom
e-www-browser (gnome-www-browser) in auto mode
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
/sbin/ldconfig.real: /usr/local/lib/libur_loader.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic
link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a sym
bolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libumf.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_opencl.so.0 is not a symboli
c link

/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link
```

```
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
Processing triggers for udev (249.11-0ubuntu3.15) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
```

In [21]:
```
!pip install nbconvert[webpdf]
```

```
Requirement already satisfied: nbconvert[webpdf] in /usr/local/lib/python3.11/
dist-packages (7.16.6)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dis
t-packages (from nbconvert[webpdf]) (4.13.4)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist
-packages (from bleach[css]!=5.0.0->nbconvert[webpdf]) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-pa
ckages (from nbconvert[webpdf]) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.11/dist-p
ackages (from nbconvert[webpdf]) (3.1.6)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.11/
dist-packages (from nbconvert[webpdf]) (5.8.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.1
1/dist-packages (from nbconvert[webpdf]) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.11/di
st-packages (from nbconvert[webpdf]) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.11/
dist-packages (from nbconvert[webpdf]) (3.1.3)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.11/di
st-packages (from nbconvert[webpdf]) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.11/dist
-packages (from nbconvert[webpdf]) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pac
kages (from nbconvert[webpdf]) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.1
1/dist-packages (from nbconvert[webpdf]) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.11/di
st-packages (from nbconvert[webpdf]) (2.19.1)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.11/dis
t-packages (from nbconvert[webpdf]) (5.7.1)
Collecting playwright (from nbconvert[webpdf])
  Downloading playwright-1.52.0-py3-none-manylinux1_x86_64.whl.metadata (3.5
kB)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-
packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert[webpdf]) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.
11/dist-packages (from bleach[css]!=5.0.0->nbconvert[webpdf]) (1.4.0)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.11/
dist-packages (from jupyter-core>=4.7->nbconvert[webpdf]) (4.3.8)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python
3.11/dist-packages (from nbclient>=0.5.0->nbconvert[webpdf]) (6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.
11/dist-packages (from nbformat>=5.7->nbconvert[webpdf]) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.11/di
st-packages (from nbformat>=5.7->nbconvert[webpdf]) (4.24.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist
-packages (from beautifulsoup4->nbconvert[webpdf]) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/pyth
on3.11/dist-packages (from beautifulsoup4->nbconvert[webpdf]) (4.13.2)
Collecting pyee<14,>=13 (from playwright->nbconvert[webpdf])
  Downloading pyee-13.0.0-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: greenlet<4.0.0,>=3.1.1 in /usr/local/lib/python
3.11/dist-packages (from playwright->nbconvert[webpdf]) (3.2.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist
-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert[webpdf]) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/loc
```

```
al/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat>=5.7->nbconve
rt[webpdf]) (2025.4.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.1
1/dist-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert[webpdf]) (0.3
6.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dis
t-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert[webpdf]) (0.25.1)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.11/dist-pac
kages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert[webpdf]) (24.
0.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.
11/dist-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert[web
pdf]) (2.9.0.post0)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.11/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert[webpdf])
(6.4.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pack
ages (from python-dateutil>=2.1->jupyter-client>=6.1.12->nbclient>=0.5.0->nbc
onvert[webpdf]) (1.17.0)
Downloading playwright-1.52.0-py3-none-manylinux1_x86_64.whl (45.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.1/45.1 MB 16.2 MB/s eta 0:00:
00
Downloading pyee-13.0.0-py3-none-any.whl (15 kB)
Installing collected packages: pyee, playwright
Successfully installed playwright-1.52.0 pyee-13.0.0
```

In [ ]:
```
!playwright install
```

Downloading Chromium 136.0.7103.25 (playwright build v1169) from https://cdn.playwright.dev/dbazure/download/playwright/builds/chromium/1169/chromium-linux.zip
167.7 MiB [] 0% 0.0s167.7 MiB [] 0% 225.3s167.7 MiB [] 0% 255.8s167.7 MiB [] 0% 137.3s167.7 MiB [] 0% 151.4s167.7 MiB [] 0% 113.0s167.7 MiB [] 0% 98.0s167.7 MiB [] 0% 69.6s167.7 MiB [] 0% 57.8s167.7 MiB [] 0% 43.7s167.7 MiB [] 0% 36.9s167.7 MiB [] 0% 35.4s167.7 MiB [] 0% 34.2s167.7 MiB [] 1% 27.9s167.7 MiB [] 1% 20.1s167.7 MiB [] 1% 18.8s167.7 MiB [] 2% 13.6s167.7 MiB [] 3% 11.1s167.7 MiB [] 4% 10.0s167.7 MiB [] 4% 8.7s167.7 MiB [] 5% 7.5s167.7 MiB [] 6% 7.1s167.7 MiB [] 6% 6.7s167.7 MiB [] 7% 6.5s167.7 MiB [] 7% 6.3s167.7 MiB [] 8% 5.6s167.7 MiB [] 9% 5.3s167.7 MiB [] 9% 5.2s167.7 MiB [] 10% 5.0s167.7 MiB [] 11% 4.6s167.7 MiB [] 12% 4.4s167.7 MiB [] 13% 4.2s167.7 MiB [] 14% 4.0s167.7 MiB [] 14% 3.9s167.7 MiB [] 15% 3.8s167.7 MiB [] 16% 3.7s167.7 MiB [] 17% 3.5s167.7 MiB [] 18% 3.4s167.7 MiB [] 19% 3.3s167.7 MiB [] 19% 3.2s167.7 MiB [] 20% 3.1s167.7 MiB [] 21% 3.1s167.7 MiB [] 22% 3.0s167.7 MiB [] 22% 2.9s167.7 MiB [] 23% 2.9s167.7 MiB [] 23% 2.8s167.7 MiB [] 24% 2.7s167.7 MiB [] 25% 2.7s167.7 MiB [] 26% 2.6s167.7 MiB [] 27% 2.6s167.7 MiB [] 28% 2.5s167.7 MiB [] 29% 2.4s167.7 MiB [] 30% 2.4s167.7 MiB [] 31% 2.3s167.7 MiB [] 32% 2.3s167.7 MiB [] 33% 2.2s167.7 MiB [] 34% 2.2s167.7 MiB [] 35% 2.1s167.7 MiB [] 36% 2.1s167.7 MiB [] 37% 2.0s167.7 MiB [] 38% 2.0s167.7 MiB [] 39% 1.9s167.7 MiB [] 40% 1.9s167.7 MiB [] 41% 1.9s167.7 MiB [] 42% 1.8s167.7 MiB [] 43% 1.7s167.7 MiB [] 44% 1.7s167.7 MiB [] 45% 1.7s167.7 MiB [] 46% 1.7s167.7 MiB [] 47% 1.6s167.7 MiB [] 48% 1.6s167.7 MiB [] 49% 1.5s167.7 MiB [] 50% 1.5s167.7 MiB [] 51% 1.4s167.7 MiB [] 52% 1.4s167.7 MiB [] 53% 1.4s167.7 MiB [] 54% 1.3s167.7 MiB [] 55% 1.3s167.7 MiB [] 56% 1.3s167.7 MiB [] 57% 1.2s167.7 MiB [] 58% 1.2s167.7 MiB [] 59% 1.2s167.7 MiB [] 60% 1.1s167.7 MiB [] 61% 1.1s167.7 MiB [] 62% 1.1s167.7 MiB [] 63% 1.0s167.7 MiB [] 64% 1.0s167.7 MiB [] 65% 1.0s167.7 MiB [] 66% 1.0s167.7 MiB [] 66% 0.9s167.7 MiB [] 67% 0.9s167.7 MiB [] 68% 0.9s167.7 MiB [] 69% 0.9s167.7 MiB [] 70% 0.9s167.7 MiB [] 70% 0.8s167.7 MiB [] 71% 0.8s167.7 MiB [] 72% 0.8s167.7 MiB [] 73% 0.8s167.7 MiB [] 74% 0.7s167.7 MiB [] 75% 0.7s167.7 MiB [] 76% 0.7s167.7 MiB [] 77% 0.7s167.7 MiB [] 77% 0.6s167.7 MiB [] 78% 0.6s167.7 MiB [] 79% 0.6s167.7 MiB [] 80% 0.6s167.7 MiB [] 81% 0.6s167.7 MiB [] 81% 0.5s167.7 MiB [] 82% 0.5s167.7 MiB [] 83% 0.5s167.7 MiB [] 84% 0.5s167.7 MiB [] 84% 0.4s167.7 MiB [] 85% 0.4s167.7 MiB [] 86% 0.4s167.7 MiB [] 87% 0.4s167.7 MiB [] 88% 0.4s167.7 MiB [] 88% 0.3s167.7 MiB [] 89% 0.3s167.7 MiB [] 90% 0.3s167.7 MiB [] 91% 0.3s167.7 MiB [] 91% 0.2s167.7 MiB [] 92% 0.2s167.7 MiB [] 93% 0.2s167.7 MiB [] 94% 0.2s167.7 MiB [] 95% 0.1s167.7 MiB [] 96% 0.1s167.7 MiB [] 97% 0.1s167.7 MiB [] 98% 0.1s167.7 MiB [] 98% 0.0s167.7 MiB [] 99% 0.0s167.7 MiB [] 100% 0.0s