Colab Notebook GitHub

Learn the Basics || Quickstart || Tensors || Datasets & DataLoaders || Transforms || Build Model || Autograd || Optimization || **Save & Load Model**

Save and Load the Model

Created On: Feb 09, 2021 | Last Updated: Sep 25, 2025 | Last Verified: Nov 05, 2024

In this section we will look at how to persist model state with saving, loading and running model predictions.

```
import torch
import torchvision.models as models
```

Saving and Loading Model Weights

PyTorch models store the learned parameters in an internal state dictionary, called `state_dict`. These can be persisted via the `torch.save` method:

```
model = models.vgg16(weights='IMAGENET1K_V1')
torch.save(model.state_dict(), 'model_weights.pth')
```

Out:

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /var/lib/

  0%|          | 0.00/528M [00:00<?, ?B/s]
  8%|▊         | 40.2M/528M [00:00<00:01, 421MB/s]
 16%|█▌        | 82.0M/528M [00:00<00:01, 430MB/s]
 24%|██▍       | 125M/528M [00:00<00:00, 440MB/s]
 32%|███▏      | 168M/528M [00:00<00:00, 444MB/s]
 40%|████      | 211M/528M [00:00<00:00, 447MB/s]
 48%|████▊     | 254M/528M [00:00<00:00, 448MB/s]
 56%|█████▋    | 298M/528M [00:00<00:00, 449MB/s]
 65%|██████▍   | 341M/528M [00:00<00:00, 450MB/s]
 73%|███████▎  | 384M/528M [00:00<00:00, 451MB/s]
 81%|████████  | 427M/528M [00:01<00:00, 452MB/s]
 89%|████████▉ | 470M/528M [00:01<00:00, 452MB/s]
 97%|█████████▋| 514M/528M [00:01<00:00, 452MB/s]
100%|██████████| 528M/528M [00:01<00:00, 448MB/s]
```

To load model weights, you need to create an instance of the same model first, and then load the parameters using `load_state_dict()` method.

In the code below, we set `weights_only=True` to limit the functions executed during unpickling to only those necessary for loading weights. Using `weights_only=True` is considered a best practice when loading weights.

```
model = models.vgg16() # we do not specify ``weights``, i.e. create untrained mod
model.load_state_dict(torch.load('model_weights.pth', weights_only=True))
model.eval()
```

Out:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Note

be sure to call `model.eval()` method before inferencing to set the dropout and batch normalization layers to evaluation mode. Failing to do this will yield inconsistent inference results.

Saving and Loading Models with Shapes

Rate this Page ★ ★ ★ ★ ★

When loading model weights, we nee Send Feedback ıodel class first, because the class unsubscribe at any time using the links in the footers of the emails I receive. **Privacy Policy**.