

Table of Contents

- Learn
- 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: Oct 15, 2024 | 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/ci-user/.cache/torch/hub/checkpoints/vgg16-397923af.pth

0%	0.00/528M [00:00<?, ?B/s]
4% 3	20.8M/528M [00:00<00:02, 216MB/s]
8% 7	42.1M/528M [00:00<00:02, 221MB/s]
12% #2	63.5M/528M [00:00<00:02, 222MB/s]
16% #6	84.9M/528M [00:00<00:02, 223MB/s]
20% ##	106M/528M [00:00<00:01, 223MB/s]
24% ###4	128M/528M [00:00<00:01, 223MB/s]
28% ###8	149M/528M [00:00<00:01, 224MB/s]
32% ###2	170M/528M [00:00<00:01, 224MB/s]
36% ###6	192M/528M [00:00<00:01, 224MB/s]
40% ####	213M/528M [00:01<00:01, 224MB/s]
44% ####4	235M/528M [00:01<00:01, 224MB/s]
49% ####8	256M/528M [00:01<00:01, 224MB/s]
53% ####2	277M/528M [00:01<00:01, 224MB/s]
57% ####6	299M/528M [00:01<00:01, 224MB/s]
61% ####	320M/528M [00:01<00:00, 224MB/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.

将反序列化过程中执行的函数限制为仅加载权重所需的那些函数

```
model = models.vgg16() # we do not specify ``weights``, i.e. create untrained model
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=False)

• 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

When loading model weights, we needed to instantiate the model class first, because the class defines the structure of a network. We might want to save the structure of this class together with the model, in which case we can pass `model` (and not `model.state_dict()`) to the saving function:

```
torch.save(model, 'model.pth')
```

We can then load the model as demonstrated below.

As described in [Saving and loading torch.nn.Modules](#), saving `state_dict` is considered the best practice. However, below we use `weights_only=False` because this involves loading the model, which is a legacy use case for `torch.save`.

```
model = torch.load('model.pth', weights_only=False),
```

• NOTE

This approach uses Python `pickle` module when serializing the model, thus it relies on the actual class definition to be available when loading the model.

## Related Tutorials


- [Saving and Loading a General Checkpoint in PyTorch](#)
- [Tips for loading an nn.Module from a checkpoint](#)

**Total running time of the script:** ( 0 minutes 8.093 seconds)

< Previous

Next >

Rate this Tutorial



### Docs

Access comprehensive developer documentation for PyTorch  
[View Docs](#)

### Tutorials

Get in-depth tutorials for beginners and advanced developers  
[View Tutorials](#)

### Resources

Find development resources and get your questions answered  
[View Resources](#)

PyTorch

Get Started

Features

Ecosystem

Blog

Contributing

Resources

Tutorials

Docs

Discuss

Github Issues

Brand Guidelines

Stay up to date

PyTorch Podcasts