How to modify the final FC layer based on the torch.model

zhongtao93 Zhongtao **Feb '17**

Hi, everyone.

I want to use the VGG19 in my own dataset, which has 8 classes. S o I want to change the output of the last fc layer to 8. So what should I do to change the last fc layer to fit it.

Thank you very much!

panovr Yili Zhao **Feb '17**

You can get the idea from this post **https://discuss.pytorch.org/t/how-to-perform-finetuning-in-pytorch/419**

Ismail_Elezi **Feb '17**

Something like:

```
model = torchvision.models.vgg19(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
    # Replace the last fully-connected layer
    # Parameters of newly constructed modules have requires_grad=True by de
model.fc = nn.Linear(512, 8) # assuming that the fc7 layer has 512 neurons,
model.cuda()
```

zhongtao93 Zhongtao **Feb '17**

Thank you, I'll try it!

Ismail_Elezi **Feb '17**

Don't forget to retrain the last layer though. At the moment the weights of it have just random numbers, so you must retrain it. You can do the retraining the same way as normal, the only change is that now the weights of the other layers won't change because I set the requires_grad to False. If you have enough data to do a full training, then simply remove the

Skip to main content t it will surely take much longer to train.

zhongtao93 Zhongtao **Feb '17**

My torch==0.1.9 while torchvision==0.1.7, but the APIs of torchvision.models.vgg19() seems is different from the docs. Here is the **error.It** ’s the version problem?
I followed the solution from **Pre-trained VGG16**, but it also has problem.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-cbf958b7f4be> in <module>()
      3 import torch.nn as nn
      4 import torch.nn.functional as F
----> 5 model = torchvision.models.vgg19(pretrained=True)
      6 for param in model.parameters():
      7     param.requires_grad = False

TypeError: vgg19() takes no arguments (1 given)
```

Crazyai gg **Feb '17**

I ran into the same problem. I cloned the latest code repos of `pytorch` and `pytorchvision` and built both manually.
Or you can refer to **this issue**

Ismail_Elezi **Feb '17**

Hmm, I didn’t try the code yesterday and just changed the line from a residual network.

Now I tried it, and got the same error as you. It seems that you can get just the model, but not the weights of it. Which actually contradicts the documentation when it is written that you can give the pretrained argument.

As @Crazyai said, you can refer to that issue and download the model from github. However, bear in mind that they are not planning to release the version with batch normalization. Alternately, use a residual network and finetune it, probably getting even better results than with VGG.

apaszke **Adam Paszke** Chief Crazy Person @ PyTorch **Feb '17**

Pretrained VGG models are a new thing. Updating **torchvision** to the newest version should fix it.

zhongtao93 Zhongtao **Mar '17**

I uninstalled the current torchvision, and rebulid the newest version, and it worked. Thank you very much!

---

zhongtao93 Zhongtao Mar '17

My network initialized as below:

```
#VGG19
import torchvision
import torch.nn as nn
import torch.nn.functional as F
vgg19 = torchvision.models.vgg19(pretrained=True)
for param in vgg19.parameters():
    param.requires_grad = False
requires_grad=True by default
vgg19.fc = nn.Linear(1000, 8) neurons, otherwise change it
vgg19.cuda()
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(vgg19.fc.parameters(), lr=0.001, momentum=0.9)#lr 0.0
```

But when I train with code as below, there is a error in **loss.backward()**:
code

```
        inputs, labels = Variable(inputs.cuda()), Variable(labels.cuda())
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = vgg19(inputs)
        #print(type(outputs),type(inputs))
        loss = criterion(outputs, labels)
        #loss = F.nll_loss(outputs, labels)
        loss.backward()
        optimizer.step()
        step += 1
        # print statistics
        running_loss += loss.data[0]
```

error

```
RuntimeError: there are no graph nodes that require computing gradients
```

Cysu Tong Xiao **Mar '17**

Newly constructed layer has `requires_grad=True` by default. You don't need to do it manually.

micklexqg **Sep '17**

> Cysu:
>
> To replace the last linear layer

thank you. what should I change if I want to add dropout in 'fc'? the last classifier layer has been changed in terms of the number of classes and I want to add dropout before it.

chenglu ChengLu She **Mar '18**

Skip to main content ›ut layer before the final FC layer, the code is

```
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
```

you only need to replace the last 4096, num_classes to your own fc layer.

Praveen_raja_sekar Jul '18

i am trying to convert the classification network to regression netowork by replacing the last layer with number of outputs as one can you please suggest any solution ay help would be greatly appreciated i am new to pytorch

---

ptrblck Jul '18

You can just do exactly this by setting the last layer as:

```
nn.Linear(4096, 1)
```

In your training procedure you probably want to use `nn.MSELoss` as your criterion.

Praveen_raja_sekar Jul '18

Hi ,
Thank you for your response. However, i tried that here is my code but the model returns nothing

```
class FineTuneModel(nn.Module):
    def __init__(self, original_model, num_classes):
        super(FineTuneModel, self).__init__()
        # Everything except the last linear layer
        self.features = nn.Sequential(*list(original_model.children())[:-1]
        self.classifier = nn.Sequential(
            nn.Linear(512, 1)
        )
        self.modelName = 'LightCNN-29'
        # Freeze those weights
        for p in self.features.parameters():
            p.requires_grad = False
```

Skip to main content

```
    def forward(self, x):
        f = self.features(x)
        f = f.view(f.size(0), -1)
        y = self.classifier(f)
        return y

model = FineTuneModel(original_model, args.num_classes)
print(model)
```

Output:

```
FineTuneModel(
  (features): Sequential()
  (classifier): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

original model is pretrained resnet model

Praveen_raja_sekar **Jul '18**

```
class FineTuneModel(nn.Module):
    def __init__(self, original_model, num_classes):
        super(FineTuneModel, self).__init__()
        # Everything except the last linear layer
        self.features = nn.Sequential(*list(original_model.children())[:-1]
        self.classifier = nn.Sequential(
            nn.Linear(512, 1)
        )
        self.modelName = 'LightCNN-29'
        # Freeze those weights
        for p in self.features.parameters():
            p.requires_grad = False


    def forward(self, x):
        f = self.features(x)
        f = f.view(f.size(0), -1)
        y = self.classifier(f)
        return y
```

ptrblck **Jul '18**

`self.features` seems to be empty.
Could you check, that `oroginal_model` is a valid model?

PS: You can add code with three backticks `. I've formatted your code for better readability.

PabloRR100 **Aug '18**

I can do this with ResNet easily but apparently VGG has no fc attribute to call.
If I build:

```
resnet_baseline = models.resnet50(pretrained=True)
vgg_baseline = models.vgg16(pretrained=True)
```

I can see the last fc layer of ResNet with `resnet_baseline.fc.in_features`.
But I just can see the last fc layer with `list(vgg_baseline.children())[-1][-1]`

I even need the second index because the way the modules are constructed is different from ResNet.

I have attempt to recreate the functionality by:

`vgg_baseline.add_module(module=nn.Linear(list(vgg_baseline.children())[-1][-1].in_features, 75), name='fc')`

And now I can call `vgg_baseline.fc`

Any ideas why VGG behaves like that?

---

hktxt Max **May '19**

Just wanna say more…
suppose use resnet18:

```
import torchvision.models as models
net = models.resnet18()
```

as you know the output has 1000 classes:

> (fc): Linear(in_features=512, out_features=1000, bias=True)

if you want to change class to 10, someone may do:

```
net.fc.out_features = 10
```

I know it's not working, but if you print the net, it gives you a changed output:

> (fc): Linear(in_features=512, out_features=3, bias=True)

that is interesting. However you will get an error when training. The correct approach is:

```
net.fc = nn.Linear(512, 10)
```

so, if you want change the input channel from 3 to 1, use:

> net.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

rather than:

> net.conv1.in_channels = 1

That’s it. It bothered me somehow, when I learned PyTorch. Just wanna point out~~~

---

ksathur_14 Sathursan Kanagarajah **Apr '20**

Hi,
How to set gradients enabled for a particular class in the fc layer?
Ex:
Consider a resnet18 model with 10 classes.
Shape of the fc.weight layer is [10,512]
I want to train the only the weights corresponds to class0.
meaning only one vector of 512 elements.
How to do that?
Thanks

---

ptrblck **Apr '20**

You cannot set the `requires_grad` attribute on slices of a parameter and would need to zero out the gradients of the frozen part of the parameter.
Alternatively you could also create two parameters (frozen and trainable), concatenate them and use the functional API for the layer operation.
However, the first approach might be a bit simpler.

---

ksathur_14 Sathursan Kanagarajah **Apr '20**

Thank you very much!

---

sigma_x Alex **May '20**

First, set the nuew number of output features:

Skip to main content

```
vgg16.classifier[6].out_features = 8
```

Then, create two new Parameter tensors, one for weights and one for bias:

```
vgg16.classifier[6].weight = torch.nn.Parameter(torch.randn(7,4096))
vgg16.classifier[6].bias = torch.nn.Parameter(torch.ones(7))
```

check by running a random image:

```
im=torch.randn(1,3,224,224)
out=vgg16(im)
>>>tensor([[-19.0846,   2.8862,   4.4280,  -0.4347,   0.8369,  -8.8550,   2
       grad_fn=<AddmmBackward>)
```

Vania_Todorova May '20

how do i just simply print out the 4096 from fc3 ?

ptrblck May '20

Assuming the 4096 are used as the number of input features, you could use:

```
print(model.fc3.in_features)
```

Vania_Todorova May '20

that after training on the new dataset right?

ptrblck May '20

I'm not sure I understand the question.
My code snippet would just print the input features of the `fc3` layer. It wouldn't depend on the training and the new dataset.
Could you explain your current use case a bit and where you're stuck at the moment?

Would you like to replace this layer with a new one to fit new input shapes?

Vania_Todorova May '20

trying to apply pca on that layer and see what happens.

Jun '20

thanks bro you have solved my problem. op

---

aguennecjacq Antoine Guennec **Mar '21**

I seem to be a bit late to the party, but I just wanted to share a supplementary solution for this topic which should work for any VGG:

```
model = torchvision.models.vgg19_bn(pretrained=True)
n_feats = 1 #   whatever is your number of output features
last_item_index = len(model.classifier)-1
old_fc = model.classifier.__getitem__(last_item_index )
new_fc = nn.Linear(in_features=old_fc.in_features, out_features= n_feats, b
model.classifier.__setitem__(last_item_index , new_fc)
```

Sehaba95 **May 18**

> aguennecjacq:
>
> ```
> old_fc = model.classifier.__getitem__(last_item_index )
> new_fc = nn.Linear(in_features=old_fc.in_features, out_features= n_feats
> model.classifier.__setitem__(last_item_index , new_fc)
> ```

This worked well for me, thanks!