`Super Pixel Pooling` in pytorch

zeakey KAI ZHAO **Nov '17**

I'm now implementing a pooling layer similar to **'super-pixel Pooling'** which has pre-computed superpixel masks to guide the pooling.

Firstly I read the document about **extending pytorch** which says

> You can extend it in both ways, but we recommend using modules for all kinds of layers, that hold any parameters or buffers, and recommend using a functional form parameter-less operations like activation functions, pooling, etc.

So I got confused here, because the operator I want to implement has no *parameter*, but do have a buffer (the superpixel masks to guide the pooling).

Now I just finished my implementation based on 'Module', by inheriting the `nn.Module` and define the `forward()`:

```
class SuperPixelPooling(nn.Module):
  def __init__(self. input, mask, output):
    super(SuperPixelPooling, self).__init__()
  def forward(self, input, mask)
    # perform superpixel pooling on input according to mask
    output = do_superpixel_pooling(input, mask)
    return output
```

All the operations in `forward()` are based on `Variables`.

Another problem occurs: when I'm using the new '*SuperpixelPooling*' module in my model, it seems impossible to put such module with multiple inputs into my model.

```
class LeNet(nn.Module):
  def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
    self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
    # seems not right?###############
    self.sp_pool2 = SuperPixelPooling()
    # ##############################
    self.conv2_drop = nn.Dropout2d()
    self.fc1 = nn.Linear(320, 50)
    self.fc2 = nn.Linear(50, 10)

  def forward(self, x):
    x = F.relu(F.max_pool2d(self.conv1(x), 2))
    x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
    # or like this ? #################
    x = SuperPixelPooling(x, mask)
    ##################################
    x = x.view(-1, 320)
    x = F.relu(self.fc1(x))
    x = F.dropout(x, training=self.training)
    x = self.fc2(x)
    return F.log_softmax(x)
```

I have to use this module with multiple inputs as a Function in my model's `forward()`, right?

Nov 2017

1 / 7
Nov 2017

Nov 2017

chenyuntc Yun Chen **Nov '17**

where is your `forward`? Do you want something like this:

```
class SuperpixelPooling(nn.Module):
  def __init__(self. input, mask, output):
        super(SuperpixelPooling, self).__init__()
        self.register_buffer('mask', torch.zeros([2222]))
        # do nothing
        # return nothing
   def forward(self, input):
        mask = self.mask
        output,new_mask  = do_superpixel_pooling(input,mask)
        self.mask = new_mask
        return output
```

---

zeakey KAI ZHAO **Nov '17**

Hi Yun,
Yeah you just got what I want to do. I have updated the code in the question.

It seems impossible to put layers with multiple inputs, just like my `SuperPixelPooling` into my model with `nn.Module`, right?

I have to implement the `Function` version and plug it into my model with `super_pixel_pooling = SuperPixelPoolingFunction(input, mask)`, right?

---

chenyuntc Yun Chen **Nov '17**

Of course module could have multi inputs.

> zeakey:
>
> x = SuperPixelPooling(x, mask)

it should be

```
x = self.sp_pool2(x,mask)
```

---

zeakey KAI ZHAO **Nov '17**

OK, I will try again.

Thank you!

---

zeakey KAI ZHAO **Nov '17**

Hi Chen Yun:
Now I have rewritten `SuperPixelPooling` operator and it seems work (I got reasonable training loss and testing accuracy),
but I'm not sure whether it is really perfect.

I'm a caffe user before, where I have to write both the `forward()` and `backward()` for a new layer.

Here I only implement the `forward()` for the `SuperPixelPooling` operator, all the operations in `forward()` are based on
`torch.autograd.Variable`.

So do I need to write `backward()` for the new operator?

---

chenyuntc Yun Chen **Nov '17**

No you don't need it