Super_Resolution_CI_Model

February 8, 2021

[1]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import torch
from torchvision import datasets, transforms
import torchvision.utils as vutils
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F
import numpy as np
import torch.nn as nn
import torch.optim as optim
```

[2]:
```
workers = 8
ngpu = 1
beta1 = 0.5
lr = 0.0002
bs = 16
epochs = 60

path_train_x = "images/train/train_x"
path_train_y = "images/train/train_y_hr"

path_valid_x = "images/valid/valid_x"
path_valid_y = "images/valid/valid_y_hr"
```

[3]:
```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

imgs_train_x = datasets.ImageFolder(path_train_x, transform = transform)
imgs_train_y = datasets.ImageFolder(path_train_y, transform = transform)

imgs_valid_x = datasets.ImageFolder(path_valid_x, transform = transform)
imgs_valid_y = datasets.ImageFolder(path_valid_y, transform = transform)
```

[4]:
```
print(len(imgs_train_x))
print(len(imgs_train_y))
#imgs_train_x.classes
#train_ds = TensorDataset(imgs_train_x, imgs_train_y)
```

```
611
611
```

[5]:
```
imgs_train_x_dl = DataLoader(imgs_train_x, batch_size = bs, num_workers =␣
↪workers)
imgs_train_y_dl = DataLoader(imgs_train_y, batch_size = bs, num_workers =␣
↪workers)

imgs_valid_x_dl = DataLoader(imgs_valid_x, batch_size = bs, num_workers =␣
↪workers)
imgs_valid_y_dl = DataLoader(imgs_valid_y, batch_size = bs, num_workers =␣
↪workers)
```

[6]:
```
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else␣
↪"cpu")
```

[7]:
```
class SuperResolution(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv = nn.Conv2d(3, 12, kernel_size = 5, padding = 2)
        self.upsample = nn.PixelShuffle(upscale_factor = 2)

    def forward(self, xb):

        xb = torch.tanh(self.conv(xb))
        xb = self.upsample(xb)
        xb = torch.sigmoid(self.conv(xb))

        return self.upsample(xb)
```

[8]:
```
def preprocess(x, y):
    return x.to(device), y.to(device)
```

[9]:
```
def get_model():
    model = SuperResolution().to(device)
    return model, optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

loss_func = nn.MSELoss(reduction='mean')
```

[10]:
```
class WrappedDataLoader:
    def __init__(self, dl_x, dl_y, func):
        assert len(dl_x) == len(dl_y)
```

```
        self.dl_x = dl_x
        self.dl_y = dl_y
        self.func = func

    def __len__(self):
        return len(self.dl_x)

    def __iter__(self):
        batches_x = iter(self.dl_x)
        batches_y = iter(self.dl_y)

        for b_x, _ in batches_x:
            b_y, _ = batches_y.next()
            yield (self.func(b_x, b_y))
```

[11]:
```
def loss_batch(model, loss_func, xb, yb, opt=None):
    loss = loss_func(model(xb), yb)
    if opt is not None:
        loss.backward()
        opt.step()
        opt.zero_grad()

    return loss.item(), len(xb)
```

[12]:
```
def fit(epochs, model, loss_func, opt, train_dl, valid_dl, val_losses):
    for epoch in range(epochs):
        model.train()
        for xb, yb in train_dl:
            loss_batch(model, loss_func, xb, yb, opt)

        model.eval()
        with torch.no_grad():
            losses, nums = zip(
                *[loss_batch(model, loss_func, xb, yb) for xb, yb in valid_dl]
            )
        val_loss = np.sum(np.multiply(losses, nums)) / np.sum(nums)
        val_losses.append(val_loss)

        print(epoch, val_loss)
```

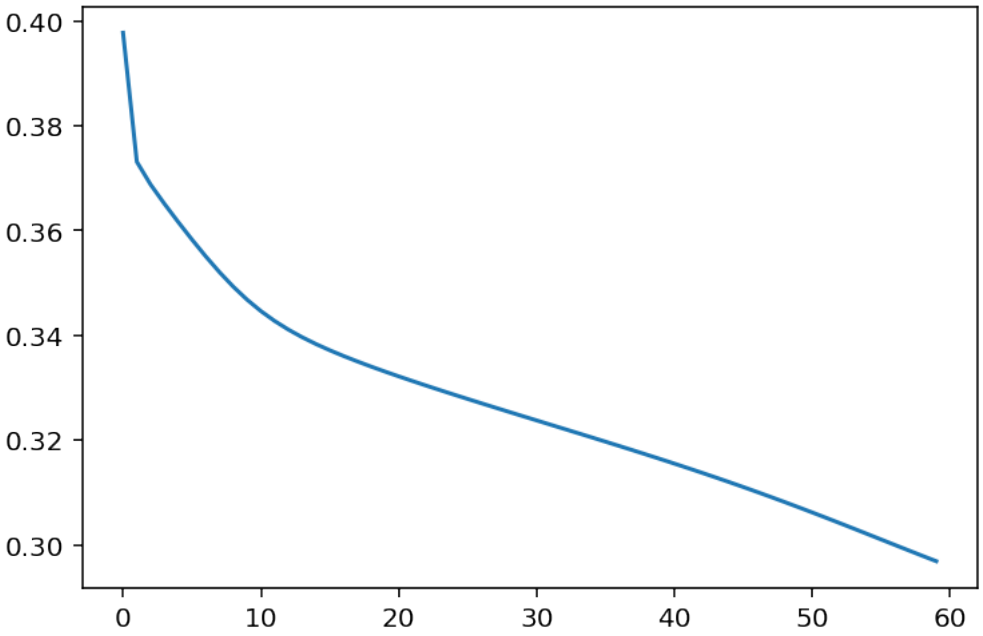
[13]:
```
train_dl = WrappedDataLoader(imgs_train_x_dl, imgs_train_y_dl, preprocess)
valid_dl = WrappedDataLoader(imgs_valid_x_dl, imgs_valid_y_dl, preprocess)

val_losses = []

model, opt = get_model()
```

```
fit(epochs, model, loss_func, opt, train_dl, valid_dl, val_losses)
```

```
0 0.39774992744127907
1 0.37311535437901816
2 0.36876518368721006
3 0.36509934544563294
4 0.3616246537367503
5 0.35826961040496824
6 0.3550539521376292
7 0.35203209598859153
8 0.34926259676615395
9 0.34678568800290427
10 0.34461071848869324
11 0.3427174361546834
12 0.34106691082318624
13 0.3396134094397227
14 0.33831310272216797
15 0.3371286038557688
16 0.3360305758317312
17 0.33499718268712364
18 0.33401267488797504
19 0.33306569894154864
20 0.33214810013771057
21 0.3312539045015971
22 0.33037856539090477
23 0.3295186086495717
24 0.3286712455749512
25 0.32783416589101155
26 0.32700547655423484
27 0.3261834700902303
28 0.32536659359931946
29 0.32455346862475076
30 0.3237427620093028
31 0.3229331549008687
32 0.3221233832836151
33 0.3213121827443441
34 0.3204982948303223
35 0.3196804360548655
36 0.31885737856229146
37 0.31802781820297243
38 0.31719051241874696
39 0.3163442083199819
40 0.31548772772153216
41 0.31461988647778827
42 0.3137395958105723
43 0.31284589529037476
44 0.3119378443559011
45 0.3110147913297017
```

```
46 0.3100761985778809
47 0.30912179867426554
48 0.30815159757932026
49 0.3071659656365712
50 0.3061656359831492
51 0.30515180865923563
52 0.304126105705897
53 0.30309072057406106
54 0.3020482965310415
55 0.3010020438830058
56 0.29995550751686095
57 0.29891268769900003
58 0.29787776350975037
59 0.2968550419807434
```

[14]:
```
plt.plot(val_losses)
plt.show()
```



[15]:
```
torch.save(model, "SR_model_3.0.ml")
```

[ ]:
```
%%javascript
Jupyter.notebook.session.delete();
```

```
<IPython.core.display.Javascript object>
```

[ ]: