Super_Resolution_CI_Model

February 9, 2021

1 Super-resolución 4x:

1.1 2.- Modelado y Entranamiento:

Ejercicio de curso para la asignatura de Computación Inteligente perteneciente al Máster Universitario en Sistemas Inteligentes y Aplicaciones Numéricas para la Ingeniería (MUSIANI) en el curso 2020/21, realizado por Juan Sebastián Ramírez Artiles.

El ejercicio consiste en implementar el método de superresolución en imágenes descrito en el paper *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*. Concretamente, en este notebook se ha implementado un escalado de cuatro aumententos. El dataset usado fue el DIV2X de libre descarga. Las imgágenes usadas son las del track1 con un tamaño para las imágenes de alta resolución recortadas a 2040x1304.

[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
```

Las imágenes se cargan de cuatro directorios, cada directorio debe contener un subdirectorio de etiqueta donde se situarán las imágenes, en este caso sólo habrá un subdirectorio, por lo tanto una etiqueta cada uno. Las etiquetas de los dataset se ignorarán ya que no se van a usar. Las imágenes reales se situan en train_y_hr y valid_y_hr y tienen un tamaño de 2040x1304. Las imágenes reducidas tienen un tamaño cuatro veces menor. Para el acondicionamiento de las imágenes se usó el notebook Super_Resolution_CI_Preprocessed_1.0.

El programa se ejecutó en una máquina con un procesador Intel Core i7-7700HQ a 2.80GHz con una tarjeta de vídeo NVIDIA GeForce GTX 1050 de 4GB de memoria dedicada y con 32 GB de memoria RAM.

Equipo usado.

[2]:
```
workers = 8
ngpu = 1
beta1 = 0.5
lr = 0.0002
bs = 16
epochs = 80

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)
```

El dataset se dividió en 611 imágenes para entrenamiento y 75 imágenes para validación.

[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")
```

El modelo hace uso de la capa PixelShuffle que se encarga de realizar el escalado a nivel subpíxel tomando como entrada un tensor de tamaño $[Batch, Channels * R^2, H, W]$ y reordenándolo como

$[Batch, Channels, H * R, W * R]$

Se probaron una variedad de modelos diferentes. Se usaron kernels de 3x3, de 5x5 y de 7x7, siendo estos últimos los que mejor resultados dieron. También se probaron diferentes configuraciones de red, añadiendo capas convolutivas y modificando las funciones de activación. Esta configuración resultó la más adecuada.

En el proceso de entrenamiento se usaron gran variedad de combinaciones de tamaños de batches y de cantidad de épocas. Se empezó con tamaños de batch de 8 y se fue subiendo hasta 16, más de esta cantidad desborda la memoras de la GPU. En cuanto al número de épocas, se empezó con 20 iteraciones y se fue subiendo hasta la cantidad final de 80.

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

        self.conv = nn.Conv2d(3, 12, kernel_size = 7, padding = 3)
        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)
```

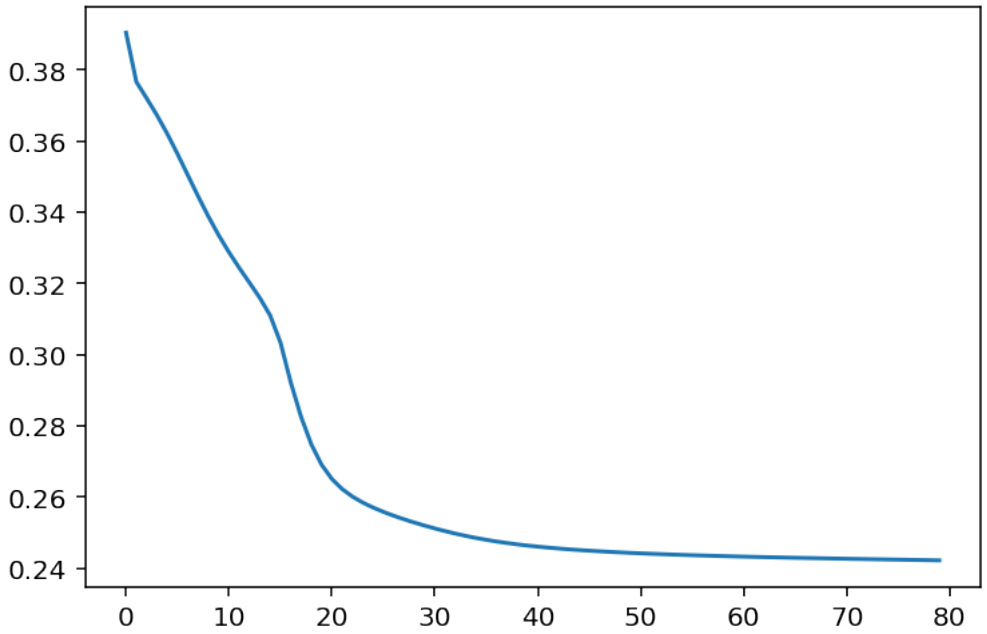
```
0 0.3904449633757273
1 0.3766347928841909
2 0.37203127225240074
3 0.36728435079256694
4 0.3620749584833781
5 0.35638980150222777
6 0.350439913670222
7 0.3445001455148061
8 0.33880977988243105
```

9 0.3335472146670024
10 0.3287638866901398
11 0.32436657071113584
12 0.32017687916755677
13 0.3159310368696849
14 0.31102603793144223
15 0.3034243297576904
16 0.2922222657998403
17 0.2825591762860616
18 0.2747908592224121
19 0.26906684080759685
20 0.2650679345925649
21 0.2622323445479075
22 0.2601172641913096
23 0.2584444457292557
24 0.2570503781239192
25 0.2558395673831304
26 0.2547554091612498
27 0.2537637275457382
28 0.2528439356883367
29 0.2519842094182968
30 0.251178994178772
31 0.25042720456918083
32 0.24973035673300426
33 0.24909045020739237
34 0.2485083947579066
35 0.24798316458861033
36 0.2475118339061737
37 0.24709011097749076
38 0.24671300967534382
39 0.24637534856796264
40 0.24607222576936086
41 0.24579916656017303
42 0.24555219908555348
43 0.24532793899377187
44 0.24512341956297556
45 0.2449361824989319
46 0.24476407965024313
47 0.24460532148679098
48 0.24445838908354442
49 0.2443219337860743
50 0.24419480045636496
51 0.24407603820165
52 0.24396475275357565
53 0.2438602230946223
54 0.243761714498202
55 0.24366870145003
56 0.2435806316137314

```
57 0.24349702636400858
58 0.24341749409834543
59 0.24334161202112833
60 0.24326907316843668
61 0.24319953600565591
62 0.24313274602095286
63 0.24306842009226481
64 0.24300632238388062
65 0.24294624547163646
66 0.2428879588842392
67 0.2428313034772873
68 0.24277609447638193
69 0.24272213757038116
70 0.24266932507356007
71 0.24261748492717744
72 0.24256647964318592
73 0.2425161749124527
74 0.2424664690097173
75 0.242417195836703
76 0.24236829777558644
77 0.2423196283976237
78 0.2422710996866226
79 0.2422225930293401
```

Se puede observar en el gráfico que la red aprende bien.

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



Para finalizar, salvo el modelo para poder usarlo en el notebook `SR_restore_model.ipynb` que se encargará de escalar las 75 imágenes de validación.

[15]:
```
torch.save(model, "SR_model_3.1.ml")
```

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

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

[ ]:
```

```