neural_network_v2

December 10, 2019

1 Neural Network v.2

With PyTorch

[1]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torch.utils.data.sampler import SubsetRandomSampler
import torch.optim as optim
import torch.nn as nn
from torchvision import transforms, utils

# custom file
from dataset import FaceEmotionsDataset
from transform import Rescale, RandomCrop, ToTensor, Normalize
from network import Net
```

1.0.1 Load the data

To load the dataset I use a custom dataset class and custom transforms process our dataset.

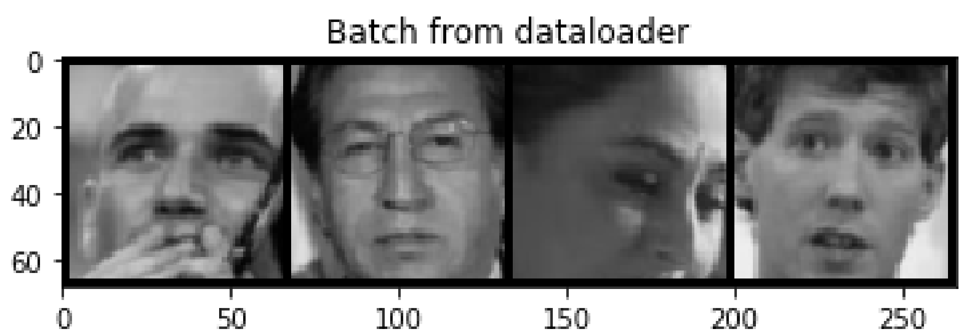
[2]:
```
data_transform = transforms.Compose([
    Rescale(68),
    RandomCrop(64),
    ToTensor()
])

emotions = ['neutral',
            'happiness',
            'surprise',
            'sadness',
            'anger',
            'disgust',
            'fear',
            'contempt']

dataset = FaceEmotionsDataset(csv_file='csv/balanced.csv',
```

```
                                                    root_dir='img/',
                                                    classes=emotions,
                                                    transform=data_transform)

dataloader = DataLoader(dataset, batch_size=4,
                        shuffle=True, num_workers=4)
```

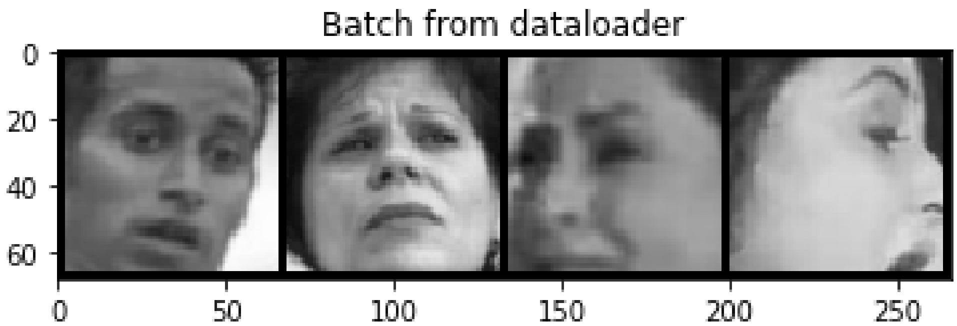
Batch preview

[3]:
```
for i_batch, sample_batched in enumerate(dataloader):
    images_batch, emotions_batch = sample_batched['image'],␣
 ↪sample_batched['emotion']
    print(i_batch, images_batch.size(),emotions_batch.size())

    # observe 4th batch and stop.

    plt.figure()
    batch_size = len(images_batch)
    im_size = images_batch.size(2)
    grid_border_size = 2

    grid = utils.make_grid(images_batch)
    plt.imshow(grid.numpy().transpose((1, 2, 0)))

    plt.title('Batch from dataloader')
    plt.show()

    if i_batch == 1:
        break
```

```
0 torch.Size([4, 1, 64, 64]) torch.Size([4])
```




```
1 torch.Size([4, 1, 64, 64]) torch.Size([4])
```



1.0.2 Split the dataset

[4]:
```
validation_split = 0.2
shuffle_dataset = True
random_seed = 0

# Creating data indices for training and validation splits:
dataset_size = len(dataset)
indices = list(range(dataset_size))
split = int(np.floor(validation_split * dataset_size))
if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]

# Creating PT data samplers and loaders:
train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)

train_loader = DataLoader(dataset, batch_size=batch_size, sampler=train_sampler)

test_loader = DataLoader(dataset, batch_size=batch_size, sampler=valid_sampler)
```

Train dataset size

[5]:
```
len(train_loader)
```

[5]: 331

Test dataset size

[6]:
```
len(test_loader)
```

[6]: 83

1.0.3 Define the network

[7]:
```
net = Net()
```

1.0.4 Define a Loss function and optimizer

[8]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

1.0.5 Train the network

[9]:
```
net = net.double()
for epoch in range(30):
    running_loss=0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        images, emotion_ids = data['image'], data['emotion']

        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = net(images.double())
        loss = criterion(outputs, emotion_ids)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 200 == 199:    # print every 200 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 200))
            running_loss = 0.0

print('Finished Training')
```

```
[1,   200] loss: 2.008
[2,   200] loss: 1.862
[3,   200] loss: 1.861
[4,   200] loss: 1.795
[5,   200] loss: 1.776
[6,   200] loss: 1.689
[7,   200] loss: 1.609
[8,   200] loss: 1.556
[9,   200] loss: 1.505
[10,   200] loss: 1.448
[11,   200] loss: 1.348
[12,   200] loss: 1.315
```

```
[13,   200] loss: 1.255
[14,   200] loss: 1.210
[15,   200] loss: 1.215
[16,   200] loss: 1.096
[17,   200] loss: 1.062
[18,   200] loss: 1.083
[19,   200] loss: 1.044
[20,   200] loss: 0.974
[21,   200] loss: 0.897
[22,   200] loss: 0.939
[23,   200] loss: 0.873
[24,   200] loss: 0.845
[25,   200] loss: 0.768
[26,   200] loss: 0.759
[27,   200] loss: 0.669
[28,   200] loss: 0.699
[29,   200] loss: 0.675
[30,   200] loss: 0.645
Finished Training
```

1.0.6 Save our trained model

[13]:
```
PATH = './models/network_v2.pth'
torch.save(net.state_dict(), PATH)
```

1.0.7 Test the network on the test data

Overall accuracy

[11]:
```
net = Net()
net.load_state_dict(torch.load(PATH))
net = net.double()
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, emotion_ids = data['image'], data['emotion']
        outputs = net(images.double())
        _, predicted = torch.max(outputs.data, 1)
        total += emotion_ids.size(0)
        correct += (predicted == emotion_ids).sum().item()

print('Accuracy of the network on the 2740 test images: %d %%' % (100*correct/
 ↪total))
```

```
Accuracy of the network on the 2740 test images: 53 %
```

Performance for each emotion

[12]:
```python
class_correct = list(0. for i in range(8))
class_total = list(0. for i in range(8))
with torch.no_grad():
    for data in test_loader:
        images, emotion_ids = data['image'], data['emotion']
        outputs = net(images.double())
        _, predicted = torch.max(outputs, 1)
        c = (predicted == emotion_ids).squeeze()
        try:
            for i in range(len(emotion_ids)):
                emotion_id = emotion_ids[i]
                class_correct[emotion_id] += c[i].item()
                class_total[emotion_id] += 1
        except:
            emotion_id = emotion_ids[0]
            class_correct[emotion_id] += c.item()
            class_total[emotion_id] += 1


for i in range(8):
    print('Accuracy of %5s : %2d %%' % (
        emotions[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of neutral : 66 %
Accuracy of happiness : 67 %
Accuracy of surprise : 50 %
Accuracy of sadness : 20 %
Accuracy of anger : 57 %
Accuracy of disgust : 65 %
Accuracy of  fear :  0 %
Accuracy of contempt :  0 %
```

1.0.8 Conclusion

The results are way better than in the previous release, but still I think we can make it better. For the emotions `fear` and `contempt` as we haven't enough data, the network is not able to learn to recognize it. That's the dataset fault. Maybe we could be to change the architecture of the network.