torchero

Release 0.0.6

Aug 23, 2021

Contents

Torchero is a library that works on top of the *PyTorch* framework built to facilitate the training of Neural Networks.

CHAPTER 1

Features

It provides tools and utilities to:

- Set up a training process in a few lines of code.
- Monitor the training performance by checking several prebuilt metrics on a handy progress bar.
- Integrate a dashboard with *TensorBoard* to visualize those metrics in an online manner with a minimal setup.
- Add custom functionality via Callbacks.

CHAPTER 2

Installlation

You can install `torchero` either from pip or from source.

2.1 From pip

```
python3 -m pip install torchero
```

2.2 From source code

Download the source code using git

```
git clone https://github.com/juancruzsosa/torchero
```

or download the tarball if you don't have *git* installed

```
curl -OL https://github.com/juancruzsosa/torchero/tarball/master
```

Once you have the source code downloaded you can install it with

```
cd torchero
python3 setup.py install
```

CHAPTER 3

Quickstart

3.1 Loading the Data

```
import torch
from torch import nn

import torchero
from torchero.models.vision import ImageClassificationModel
from torchero.callbacks import ProgbarLogger, ModelCheckpoint, CSVLogger
from torchero.utils.data import train_test_split
from torchero.utils.vision import show_imagegrid_dataset, transforms, datasets,␣
↪download_image
from torchero.meters import ConfusionMatrix

from matplotlib import pyplot as plt
```

First we load the MNIST train and test datasets and visualize it using `show_imagegrid_dataset`.
The Data Augmentation for this case will be a RandomInvert to flip the grayscale levels.

```
train_ds = datasets.MNIST(root='/tmp/data/mnist', download=True, train=True,␣
↪transform=transforms.Compose([transforms.RandomInvert(),
test_ds = datasets.MNIST(root='/tmp/data/mnist', download=False, train=False,␣
↪transform=transforms.ToTensor())
show_imagegrid_dataset(train_ds)
plt.show()
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to /tmp/data/mnist/MNIST/raw/train-images-idx3-ubyte.gz
```

9920512/? [02:30<00:00, 71373.53it/s]

```
Extracting /tmp/data/mnist/MNIST/raw/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to /tmp/data/mnist/MNIST/raw/train-labels-idx1-ubyte.gz
```

32768/? [00:14<00:00, 80759.89it/s]

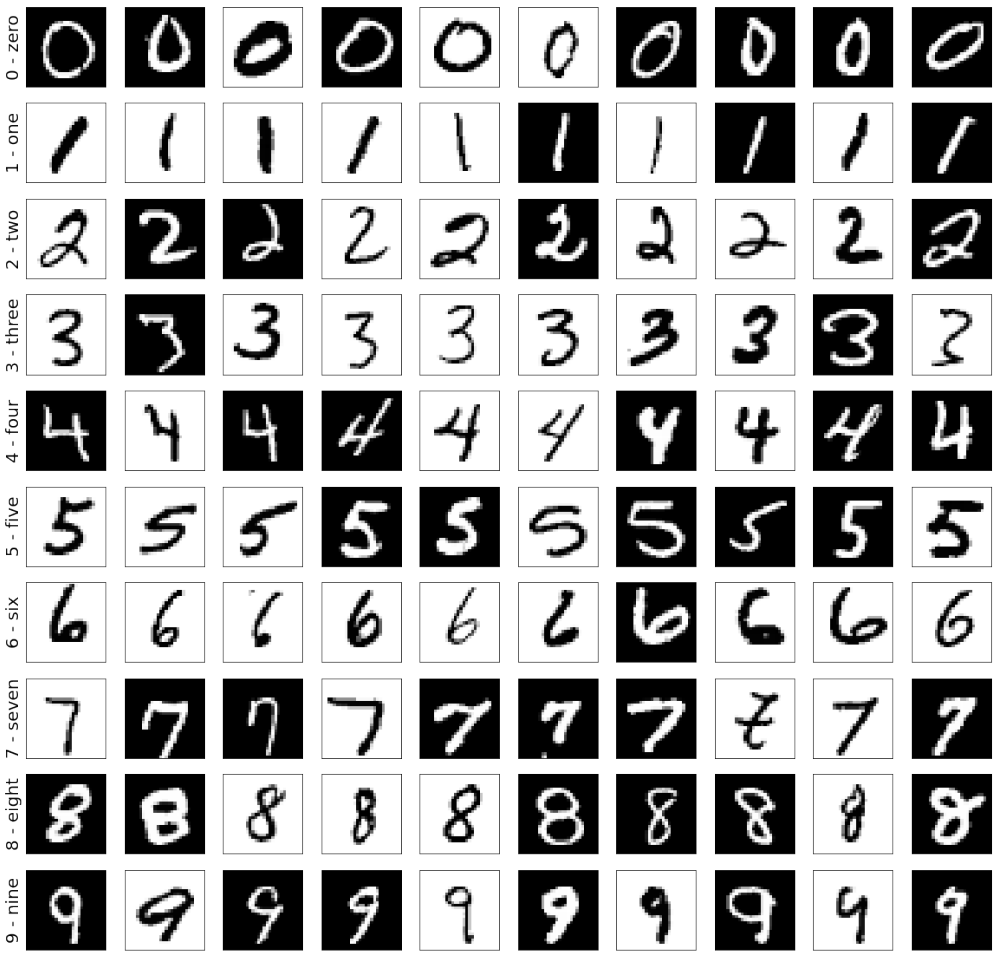
```
Extracting /tmp/data/mnist/MNIST/raw/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to /tmp/data/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz
```

1654784/? [00:43<00:00, 84501.91it/s]

```
Extracting /tmp/data/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to /tmp/data/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

8192/? [00:49<00:00, 41403.91it/s]

```
Extracting /tmp/data/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz
Processing...
Done!
```



3.2 Defining the Network

Let's define a Convolutional network of two layers followed by a Linear Module
as the classification layer.

```
model = nn.Sequential(nn.Conv2d(in_channels=1, out_channels=32, kernel_size=5),
                      nn.ReLU(inplace=True),
                      nn.MaxPool2d(2),
                      nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
                      nn.ReLU(inplace=True),
                      nn.MaxPool2d(2),
                      nn.Flatten(),
                      nn.Linear(5*5*64, 500),
                      nn.ReLU(inplace=True),
                      nn.Linear(500, 10))
```

3.3 Training the Model

The ImageClassificationModel is the module responsible to train the model,
evaluate a metric against a dataset, and predict from and input for multi-class classification tasks.

To train the model we need to compile it first with a:

- an optimizer: 'adam'
- a loss which will be defaulted to `cross_entropy`
- a list of metrics which will be defaulted to `categorical_accuracy`, `balanced_accuracy`)
- a list of callbacks:
    - ProgbarLogger to show training progress bar
    - ModelCheckpoint to make checkpoints if the model improves
    - CSVLogger to dump the metrics to a csv file after each epoch

```
model = ImageClassificationModel(model=network,
                                 transform=transforms.Compose([transforms.Grayscale(),
                                                               transforms.Resize((28,
↪28)),
                                                               transforms.
↪ToTensor()]),
                                 classes=[str(i) for i in range(10)])
model.compile(optimizer='adam',
              callbacks=[ProgbarLogger(notebook=True),
                         ModelCheckpoint('saved_model', mode='max', monitor='val_acc
↪'),
                         CSVLogger('training_results.xml')])

if torch.cuda.is_available():
    model.cuda()

history = model.fit(train_ds,
```

(continues on next page)

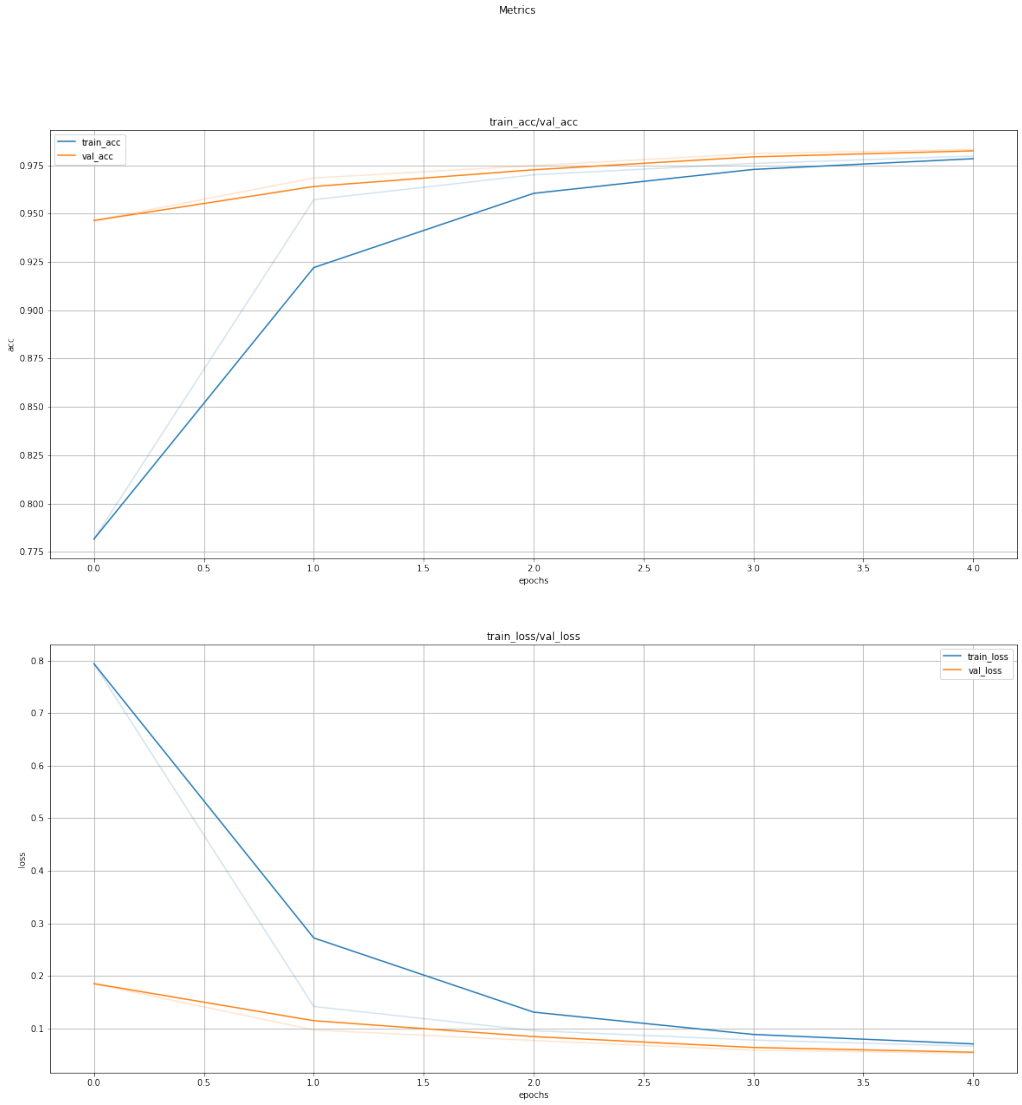
(continued from previous page)

```
                    test_ds,
                    batch_size=1024,
                    epochs=5)
```

3.4 Visualizing the training training results

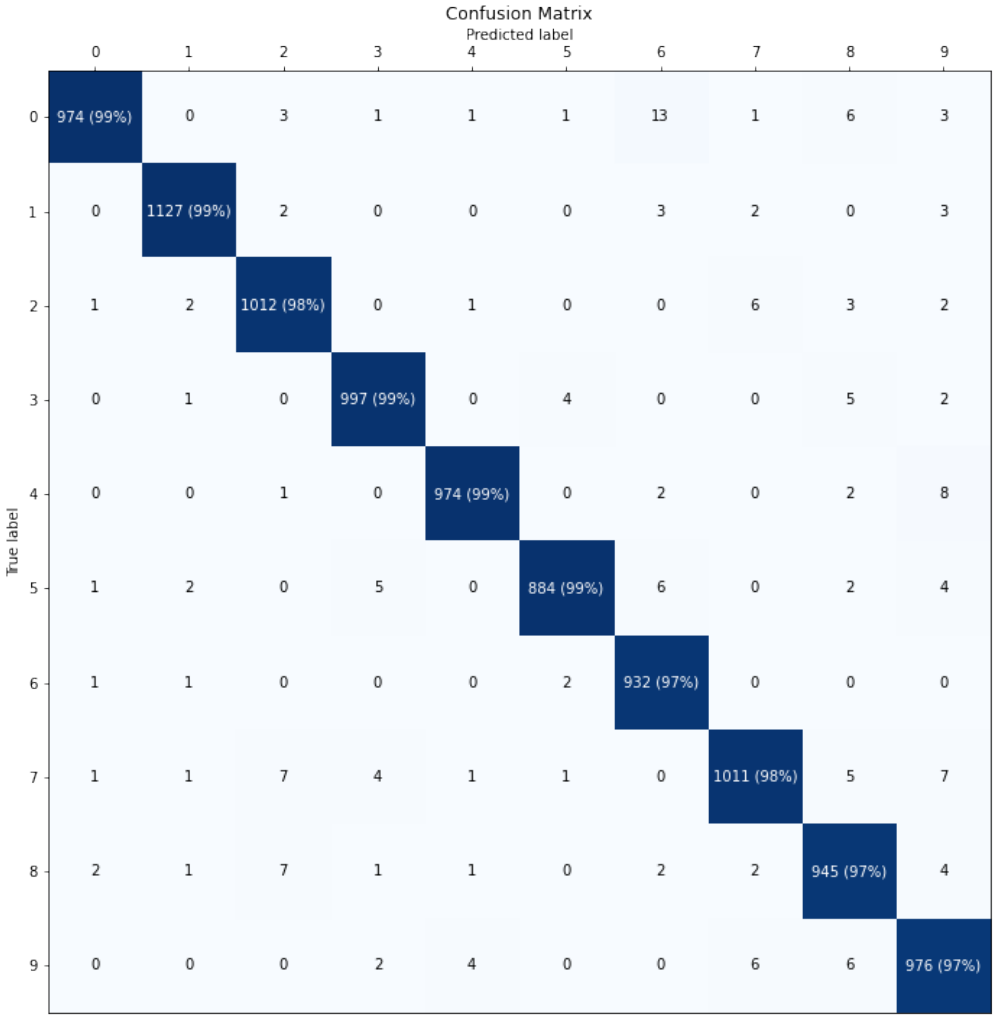
To visualize our loss and accuracy in each epoch we can execute:

```
history.plot(figsize=(20, 20), smooth=0.2)
plt.show()
```



The `.evaluate` returns the metrics for a new dataset.

```
results = trainer.evaluate(test_dl, ['categorical_accuracy_percentage', 'confusion_
↪matrix'])
plt.figure(figsize=(10, 10))
results['confusion_matrix'].plot(classes=train_ds.classes)
```


Confusion Matrix
Predicted label
True label
0 1 2 3 4 5 6 7 8 9
974 (99%) 0 3 1 1 1 13 1 6 3
0 1127 (99%) 2 0 0 0 3 2 0 3
1 2 1012 (98%) 0 1 0 0 6 3 2
0 1 0 997 (99%) 0 4 0 0 5 2
0 0 1 0 974 (99%) 0 2 0 2 8
1 2 0 5 0 884 (99%) 6 0 2 4
1 1 0 0 0 2 932 (97%) 0 0 0
1 1 7 4 1 1 0 1011 (98%) 5 7
2 1 7 1 1 0 2 2 945 (97%) 4
0 0 0 2 4 0 0 6 6 976 (97%)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search