PyTorch

Ta le of Contents

VIS ALIZING MODELS, DATA, AND TRAINING WITH TENSORBOARD

In the 60 Minute Blitz, we show you how to loa in ata, fee it through a mo el we efine as a su class of `nn.Module`, train this mo el on training ata, an test it on test ata. To see what's happening, we print out some statistics as the mo el is training to get a sense for whether training is progressing. However, we can o much etter than that: PyTorch integrates with TensorBoar , a tool esigne for visualizing the results of neural network training runs. This tutorial illustrates some of its functionality, using the Fashion-MNIST ataset which can e rea into PyTorch using *torchvision.datasets*.

In this tutorial, we'll learn how to:

1. Rea in ata an with appropriate transforms (nearly i entical to the prior tutorial).
2. Set up TensorBoar .
3. Write to TensorBoar .
4. Inspect a mo el architecture using TensorBoar .
5. se TensorBoar to create interactive versions of the visualizations we create in last tutorial, with less co e

Specifically, on point #5, we'll see:

- A couple of ways to inspect our training ata
- How to track our mo el's performance as it trains
- How to assess our mo el's performance once it is traine .

We'll egin with similar oilerplate co e as in the CIFAR-10 tutorial:

```python
# imports
import matplotlib.pyplot as plt
import numpy as np

import torch
import torchvision
import torchvision.transforms as transforms

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# transforms
transform = transforms.Compose(
    [transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))])

# datasets
trainset = torchvision.datasets.FashionMNIST('./data',
    download=True,
    train=True,
    transform=transform)
testset = torchvision.datasets.FashionMNIST('./data',
    download=True,
    train=False,
    transform=transform)

# dataloaders
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                        shuffle=True, num_workers=2)


testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                        shuffle=False, num_workers=2)

# constant for classes
classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')

# helper function to show an image
# (used in the `plot_classes_preds` function below)
def matplotlib_imshow(img, one_channel=False):
    if one_channel:
        img = img.mean(dim=0)
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    if one_channel:
        plt.imshow(npimg, cmap="Greys")
    else:
        plt.imshow(np.transpose(npimg, (1, 2, 0)))
```

We'll efine a similar mo el architecture from that tutorial, making only minor mo ifications to account for the fact that the images are now one channel instea of three an 28x28 instea of 32x32:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

We'll efine the same `optimizer` an `criterion` from efore:

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

1. TensorBoar setu

Now we'll set up TensorBoar , importing `tensorboard` from `torch.utils` an efining a `SummaryWriter`, our key o ect for writing information to TensorBoar .

```python
from torch.utils.tensorboard import SummaryWriter

# default `log_dir` is "runs" - we'll be more specific here
writer = SummaryWriter('runs/fashion_mnist_experiment_1')
```

Note that this line alone creates a `runs/fashion_mnist_experiment_1` fol er.

2. Writing to TensorBoar

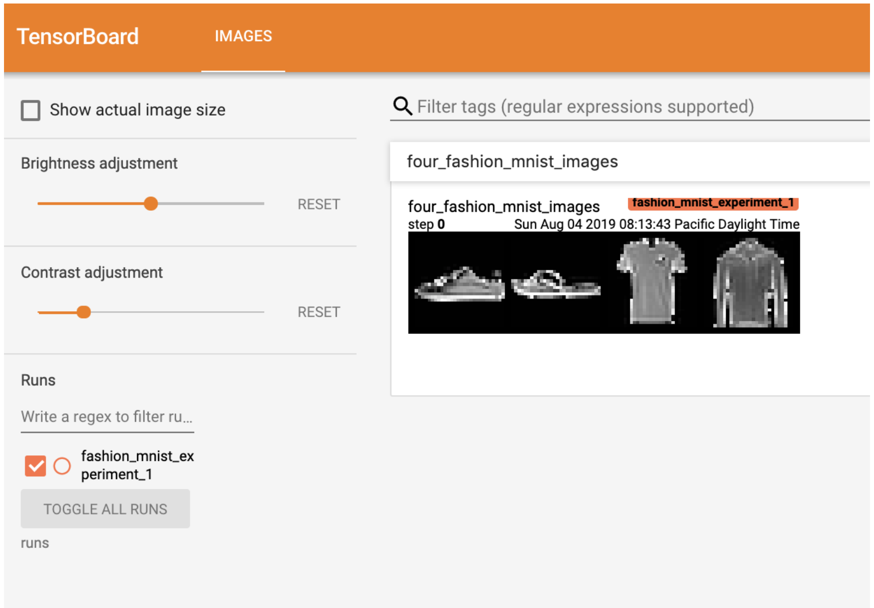
Now let's write an image to our TensorBoar - specifically, a gri - using make_gri .

```python
# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# create grid of images
img_grid = torchvision.utils.make_grid(images)

# show images
matplotlib_imshow(img_grid, one_channel=True)

# write to tensorboard
writer.add_image('four_fashion_mnist_images', img_grid)
```

Now running

```
tensorboard --logdir=runs
```

from the comman line an then navigating to https://localhost:6006 shoul show the following.


TensorBoard
IMAGES
Show actual image size
Filter tags (regular expressions supported)
Brightness adjustment
RESET
four_fashion_mnist_images
four_fashion_mnist_images
fashion_mnist_experiment_1
step 0
Sun Aug 04 2019 08:13:43 Pacific Daylight Time
Contrast adjustment
RESET
Runs
Write a regex to filter ru...
fashion_mnist_experiment_1
TOGGLE ALL RUNS
runs

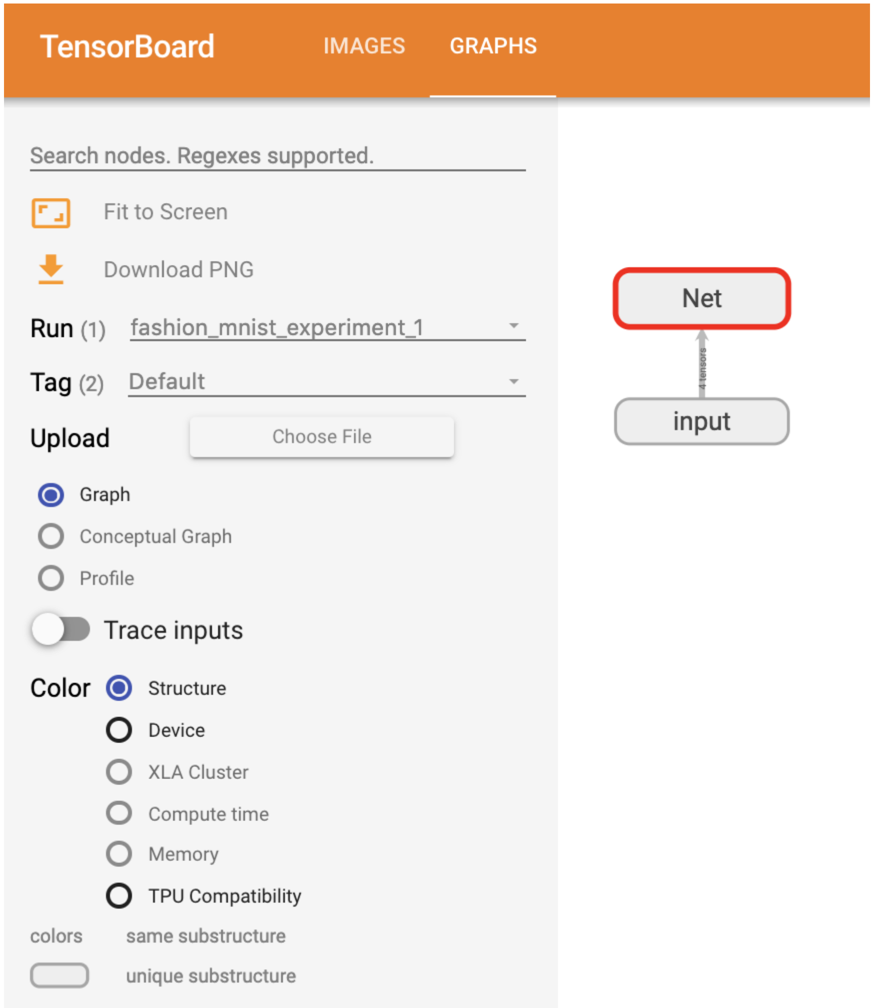
Now you know how to use TensorBoar ! This example, however, coul e one in a Jupyter Note ook - where TensorBoar really excels is in creating interactive visualizations. We'll cover one of those next, an several more y the en of the tutorial.

3. Ins ect the mo el using TensorBoar

One of TensorBoar 's strengths is its a ility to visualize complex mo el structures. Let's visualize the mo el we uilt.

```
writer.add_graph(net, images)
writer.close()
```

Now upon refreshing TensorBoar you shoul see a “Graphs” ta that looks like this:



Go ahea an ou le click on “Net” to see it expan , seeing a etaile view of the in ivi ual operations that make up the mo el.

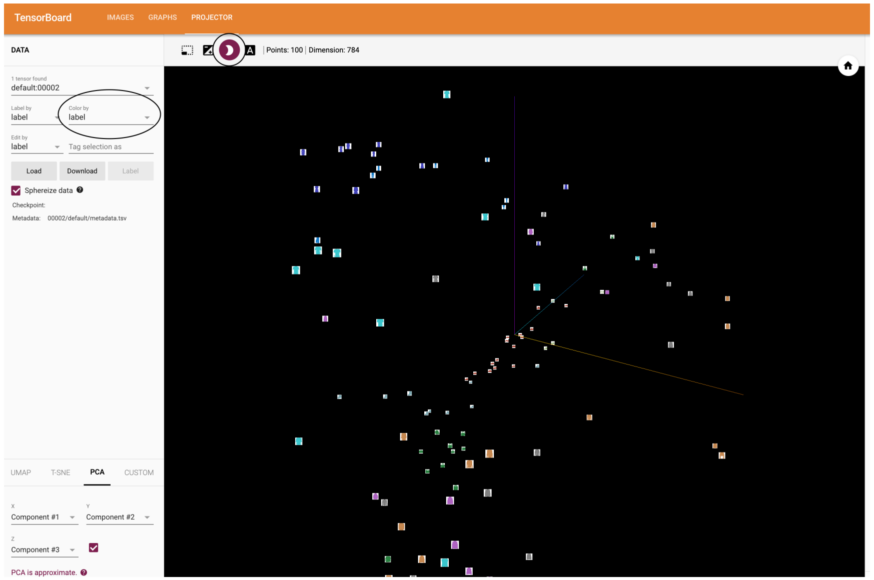
TensorBoar has a very han y feature for visualizing high imensional ata such as image ata in a lower imensional space; we'll cover this next.

4. A ing a “Pro ector” to TensorBoar

We can visualize the lower imensional representation of higher imensional ata via the a _em e ing metho

```
# helper function
def select_n_random(data, labels, n=100):
    '''
    Selects n random datapoints and their corresponding labels from a dataset
    '''
    assert len(data) == len(labels)

    perm = torch.randperm(len(data))
    return data[perm][:n], labels[perm][:n]

# select random images and their target indices
images, labels = select_n_random(trainset.data, trainset.targets)

# get the class labels for each image
class_labels = [classes[lab] for lab in labels]

# log embeddings
features = images.view(-1, 28 * 28)
writer.add_embedding(features,
                    metadata=class_labels,
                    label_img=images.unsqueeze(1))
writer.close()
```

Now in the “Pro ector” ta of TensorBoar , you can see these 100 images - each of which is 784 imensional - pro ecte own into three imensional space. Furthermore, this is interactive: you can click an rag to rotate the three imensional pro ection. Finally, a couple of tips to make the visualization easier to see: select “color: la el” on the top left, as well as ena ling “night mo e”, which will make the images easier to see since their ackgroun is white:



Now we’ve thoroughly inspecte our ata, let’s show how TensorBoar can make tracking mo el training an evaluation clearer, starting with training.

5. Tracking mo el training with TensorBoar

In the previous example, we simply *printed* the mo el’s running loss every 2000 iterations. Now, we’ll instea log the running loss to TensorBoar , along with a view into the pre ictions the mo el is making via the `plot_classes_preds` function.

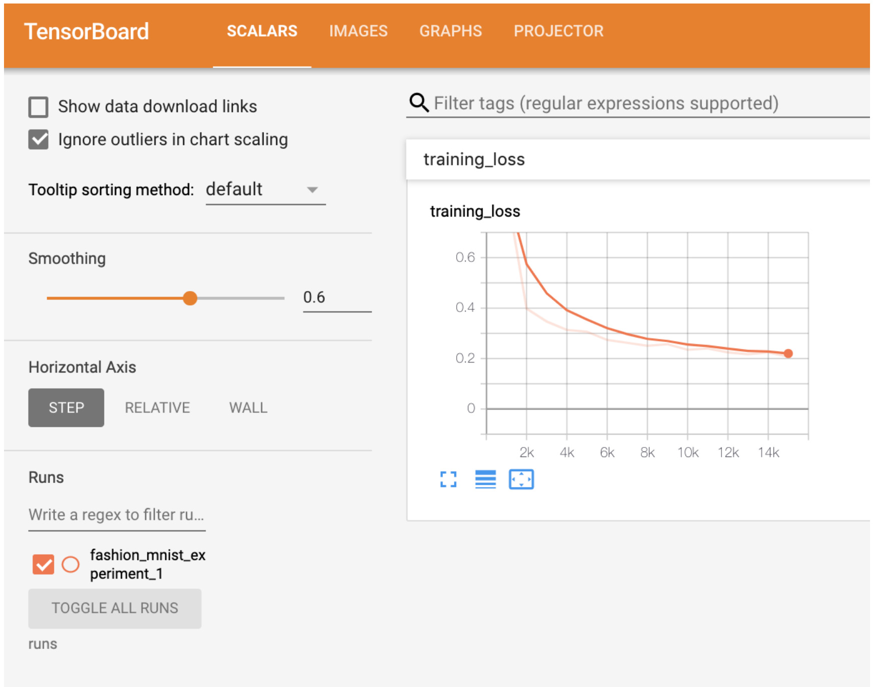
```
# helper functions

def images_to_probs(net, images):
    '''
    Generates predictions and corresponding probabilities from a trained
    network and a list of images
    '''
    output = net(images)
    # convert output probabilities to predicted class
    _, preds_tensor = torch.max(output, 1)
    preds = np.squeeze(preds_tensor.numpy())
    return preds, [F.softmax(el, dim=0)[i].item() for i, el in zip(preds, output)]


def plot_classes_preds(net, images, labels):
    '''
    Generates matplotlib Figure using a trained network, along with images
    and labels from a batch, that shows the network's top prediction along
    with its probability, alongside the actual label, coloring this
    information based on whether the prediction was correct or not.
    Uses the "images_to_probs" function.
    '''
    preds, probs = images_to_probs(net, images)
    # plot the images in the batch, along with predicted and true labels
    fig = plt.figure(figsize=(12, 48))
    for idx in np.arange(4):
        ax = fig.add_subplot(1, 4, idx+1, xticks=[], yticks=[])
        matplotlib_imshow(images[idx], one_channel=True)
        ax.set_title("{0}, {1:.1f}%\n(label: {2})".format(
            classes[preds[idx]],
            probs[idx] * 100.0,
            classes[labels[idx]]),
                    color=("green" if preds[idx]==labels[idx].item() else "red"))
    return fig
```

Finally, let’s train the mo el using the same mo el training co e from the prior tutorial, ut writing results to TensorBoar every 1000 atches instea of printing to console; this is one using the a _scalar function.

In a ition, as we train, we’ll generate an image showing the mo el’s pre ictions vs. the actual results on the four images inclu e in that atch.

```
running_loss = 0.0
for epoch in range(1):  # loop over the dataset multiple times

    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 1000 == 999:    # every 1000 mini-batches...

            # ...log the running loss
            writer.add_scalar('training loss',
                            running_loss / 1000,
                            epoch * len(trainloader) + i)

            # ...log a Matplotlib Figure showing the model's predictions on a
            # random mini-batch
            writer.add_figure('predictions vs. actuals',
                            plot_classes_preds(net, inputs, labels),
                            global_step=epoch * len(trainloader) + i)
            running_loss = 0.0
print('Finished Training')
```

You can now look at the scalars ta to see the running loss plotte over the 15,000 iterations of training:



□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

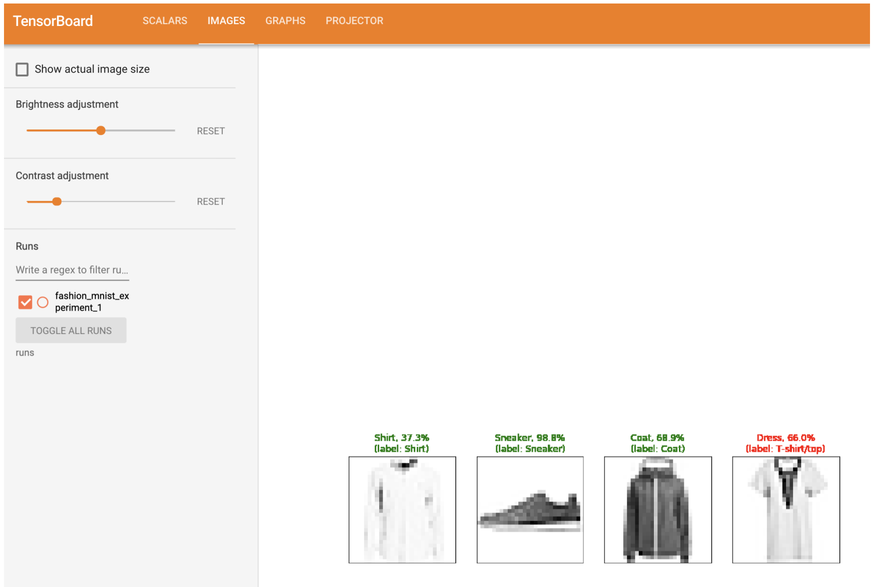
Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter ru...

☑ ○ fashion_mnist_experiment_1

In a ition, we can look at the pre ictions the mo el ma e on ar itrary atches throughout learning. See the “Images” ta an scroll own un er the “pre ictions vs. actuals” visualization to see this; this shows us that, for example, after ust 3000 training iterations, the mo el was alrea y a le to istinguish etween visually istinct classes such as shirts, sneakers, an coats, though it isn’t as confi ent as it ecomes later on in training:



□ Show actual image size

Brightness adjustment

Contrast adjustment

Runs

Write a regex to filter ru...

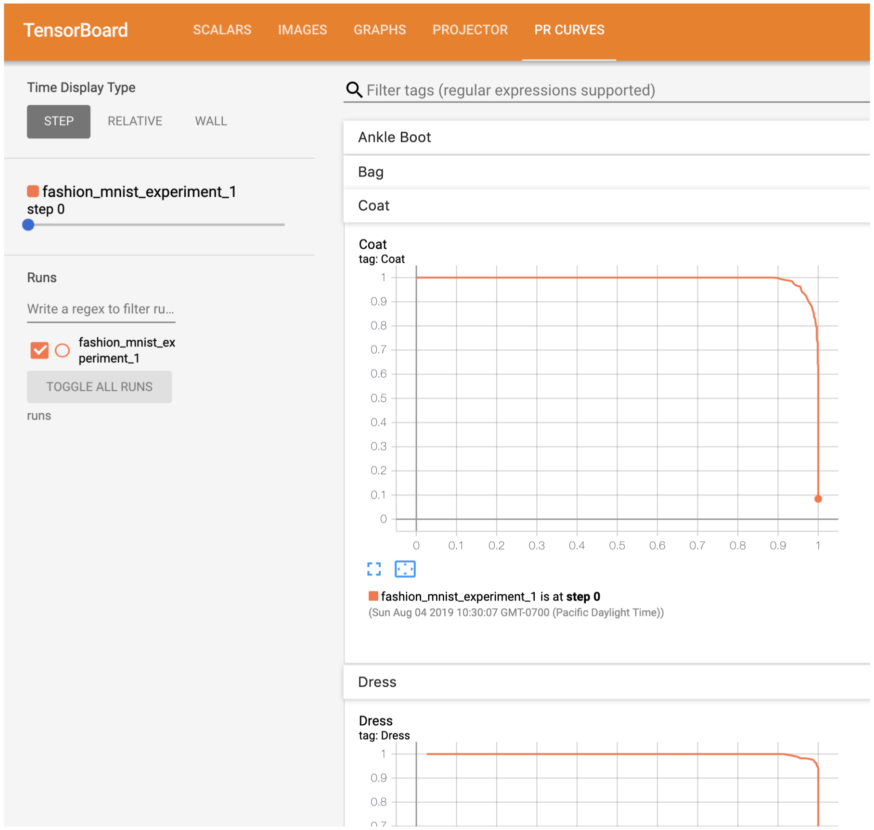
☑ ○ fashion_mnist_experiment_1

In the prior tutorial, we looke at per-class accuracy once the mo el ha een traine ; here, we’ll use TensorBoar to plot precision-recall curves (goo explanation here) for each class.

6. Assessing traine mo els with TensorBoar

```python
# 1. gets the probability predictions in a test_size x num_classes Tensor
# 2. gets the preds in a test_size Tensor
# takes ~10 seconds to run
class_probs = []
class_preds = []
with torch.no_grad():
    for data in testloader:
        images, labels = data
        output = net(images)
        class_probs_batch = [F.softmax(el, dim=0) for el in output]
        _, class_preds_batch = torch.max(output, 1)

        class_probs.append(class_probs_batch)
        class_preds.append(class_preds_batch)

test_probs = torch.cat([torch.stack(batch) for batch in class_probs])
test_preds = torch.cat(class_preds)

# helper function
def add_pr_curve_tensorboard(class_index, test_probs, test_preds, global_step=0):
    '''
    Takes in a "class_index" from 0 to 9 and plots the corresponding
    precision-recall curve
    '''
    tensorboard_preds = test_preds == class_index
    tensorboard_probs = test_probs[:, class_index]

    writer.add_pr_curve(classes[class_index],
                        tensorboard_preds,
                        tensorboard_probs,
                        global_step=global_step)
    writer.close()

# plot all the pr curves
for i in range(len(classes)):
    add_pr_curve_tensorboard(i, test_probs, test_preds)
```

You will now see a “PR Curves” ta that contains the precision-recall curves for each class. Go ahea an poke aroun ; you’ll see that on some classes the mo el has nearly 100% “area un er the curve”, whereas on others this area is lower:



fashion_mnist_experiment_1 is at **step 0**
(Sun Aug 04 2019 10:30:07 GMT-0700 (Pacific Daylight Time))

An that's an intro to TensorBoar an PyTorch's integration with it. Of course, you coul o everything TensorBoar oes in your Jupyter Note ook, ut with TensorBoar , you gets visuals that are interactive y efault.

< Previous Next >

Rate this Tutorial ☆☆☆☆☆