Lab04 - (Extra) Plotting with Tensorboard

To observe if the model is training well and identify if the model has converged, it is important to monitor the loss function vs iteration. To do this, we can use Tensorboard which is a tool designed for visualizing the results of neural network training runs. It provides the visualization needed for checking your machine learning model. Two main uses of tensorboard are to

1. Visualize the constructed computational graph via `writer.add_graph` method.
2. Track and visualize metrics such as loss and accuracy via `writer.add_scaler` method

Objectives:

1. Learn how to use Tensorboard to monitor training metrics such as training loss and accuracy and visualize the graph for our model.

Task:

Visualize the training of the CIFAR10 model using Tensorboard.

Content:

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [2]:
```
cd "/content/gdrive/My Drive/UCCD3074_Labs/UCCD3074_Lab4"
```

```
/content/gdrive/My Drive/UCCD3074_Labs/UCCD3074_Lab4
```

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt

from cifar10 import CIFAR10

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

import torch.optim as optim

import torchvision
import torchvision.transforms as transforms

from torch.utils.data import DataLoader

import time

%load_ext autoreload
%autoreload 2
```

---

1. Load Training Set and Construct Network

The following code loads the train and test set for CIFAR10.

In [4]:

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

In [16]:

```
trainset = CIFAR10(train=True,  transform=transform, num_samples=10000, download=True)
testset  = CIFAR10(train=False,  transform=transform, num_samples=2000, download=True)
classes = trainset.classes
```

```
trainloader = DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)
testloader  = DataLoader(testset, batch_size=4, shuffle=True, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

The following code constructs the network model

In [6]:
```
class Network(nn.Module):

    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(3*32*32, 50)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        return x

def build_model():
    net = Network()
    if torch.cuda.is_available():
        net = net.cuda()
    return net
```

In [7]:
```
net = build_model()
```

---

2. Visualize a Graph

One of TensorBoard's strengths is its ability to visualize complex model structures. Let's visualize the model we built. First, we import `SummaryWriter` from `torch.utils.tensorboard` and define a SummaryWriter object:

```
writer = SummaryWriter(log_dir)
```

where `log_dir` is the directory to store the written item.

In [40]:
```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter('runs/graph')
```

To create the computational graph, `summarywriter` requires you to supply a batch data.

In [41]:
```
X = torch.randn(4, 3, 32, 32)
if torch.cuda.is_available():
    X = X.cuda()
```

Now, let's add the graph to tensorboard through the command `add_graph`

In [42]:
```
writer.add_graph(net, X)
```

After we have finish updating the graph, close the writer.

In [43]:
```
writer.close()
```

Now, let's open up tensorboard on Colab to view the graph that we have just writtten to `runs/graph`.

In [56]:
```
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [76]:
```
%tensorboard --logdir runs
```

```
Output hidden; open in https://colab.research.google.com to view.
```

---

3. Monitoring a Scalar Value

Besides displaying a graph, Tensorboard can be be used to monitor a scalar value over different time step. We can use it to monitor the the training/testing accuracy/loss.

Add scalar during training

To monitor a variable, call the function `add_scalar` to update the variable continuously during training.

```
writer.add_scalar(tag, scalar_value, step)
```

where the input arguments are:

- `tag` (string) is the name of the experiment. Tensorboard can show multiple logs from *different* experiments with the *same* tag on the same figure. This allows easy comparison.
- `scalar_value` (a scalar value) is the value to be saved and monitored, e.g., the accuracy or loss value.
- `step` is the training step when `scalar_value` is generated.

In the `train` function below, we monitor the training loss as the training commences. We shall save the `train_loss` to `tensorboard` (line 54) every 100 batch cycles (line 47).

In [59]:

```python
import time

def train (model, writer, lr=0.001, num_epochs=5, loop_per_val=200):

    # set the optimizer
    optimizer = optim.SGD(net.parameters(), lr=lr)

    # set the criterion
    criterion = nn.CrossEntropyLoss()

    # set to training mode
    net.train()

    # train the model
    for e in range(num_epochs):

        running_loss = 0
        running_count = 0
```

```
        for i, (inputs, labels) in enumerate(trainloader):

            # transfer input to GPU
            if torch.cuda.is_available():
                inputs = inputs.cuda()
                labels = labels.cuda()

            # set grad to zero
            net.zero_grad()

            # forward propagation
            outputs = net(inputs)

            # compute loss
            loss = criterion(outputs, labels)

            # backward propagation
            loss.backward()

            # update model
            optimizer.step()

            # compute running loss and validation
            running_loss += loss.item()
            running_count +=1

            # display the averaged loss value every 100 cycles
            if i % loop_per_val == loop_per_val - 1:
                train_loss = running_loss / loop_per_val
                running_loss = 0.
                running_count = 0
                print(f'[Epoch {e+1:2d}/{num_epochs:d} Iter {e*len(trainloader)+i:5d}]: train_loss = {train_loss:.4f}')

                # Update tensorboard
                writer.add_scalar("Training loss", train_loss, e*len(trainloader)+i)

    return model
```

Next, create a summary writer object. We shall name our `log_dir` based on the learning rate (`lr`) settings since we will be comparing different `lr` settings later.

In [60]:

```
writer = SummaryWriter('runs/cifar_lr0.001')
```

In [61]:

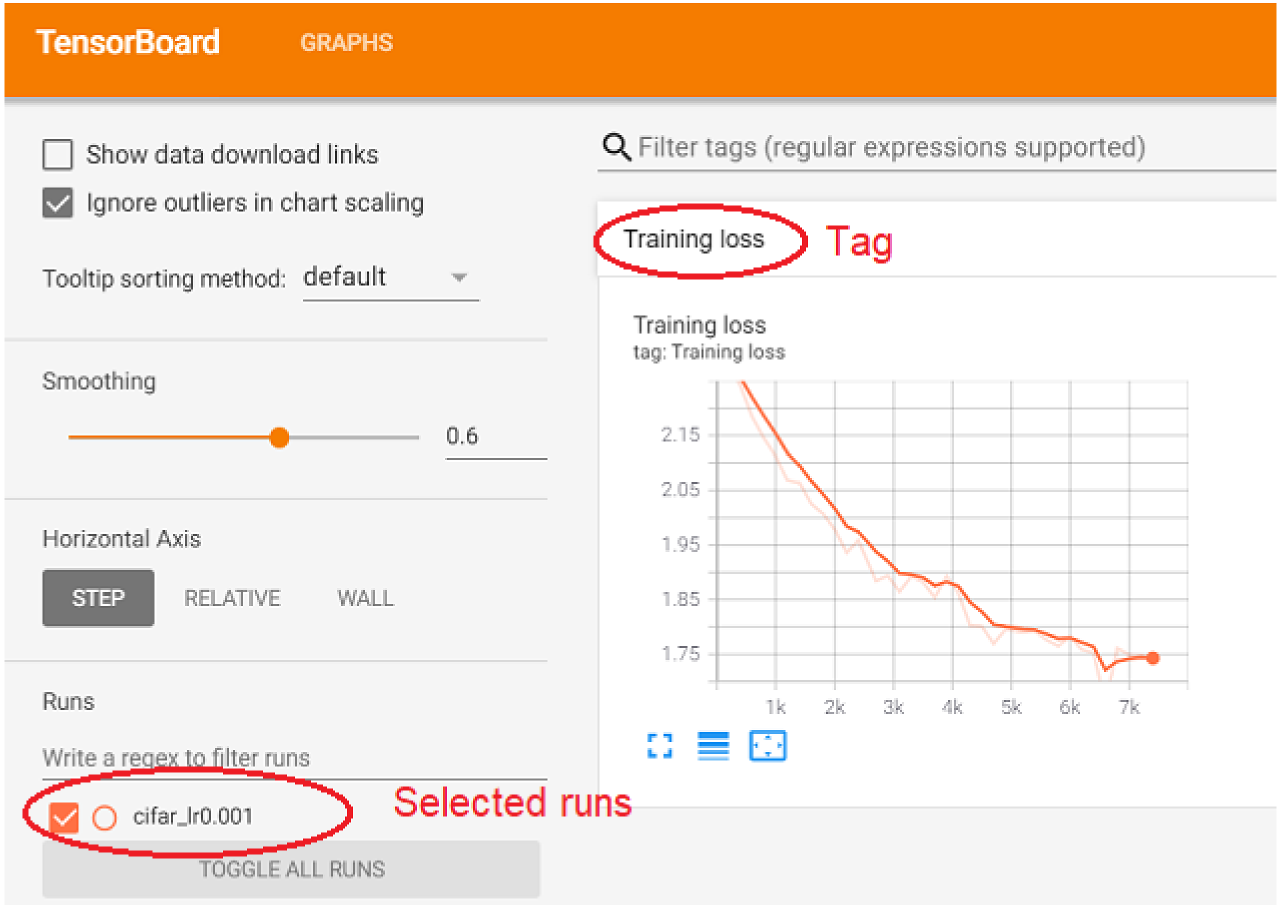
```
net = build_model()
net = train (net, writer, lr = 0.001, num_epochs=3, loop_per_val=200)
```

```
[Epoch  1/3 Iter   199]: train_loss = 2.2629
[Epoch  1/3 Iter   399]: train_loss = 2.2043
[Epoch  1/3 Iter   599]: train_loss = 2.1585
[Epoch  1/3 Iter   799]: train_loss = 2.1023
[Epoch  1/3 Iter   999]: train_loss = 2.0967
[Epoch  1/3 Iter  1199]: train_loss = 2.0120
[Epoch  1/3 Iter  1399]: train_loss = 2.0342
[Epoch  1/3 Iter  1599]: train_loss = 2.0096
[Epoch  1/3 Iter  1799]: train_loss = 1.9712
[Epoch  1/3 Iter  1999]: train_loss = 1.9607
[Epoch  1/3 Iter  2199]: train_loss = 1.9945
[Epoch  1/3 Iter  2399]: train_loss = 1.9643
[Epoch  2/3 Iter  2699]: train_loss = 1.8742
[Epoch  2/3 Iter  2899]: train_loss = 1.8986
[Epoch  2/3 Iter  3099]: train_loss = 1.8525
[Epoch  2/3 Iter  3299]: train_loss = 1.9143
[Epoch  2/3 Iter  3499]: train_loss = 1.8484
[Epoch  2/3 Iter  3699]: train_loss = 1.8375
[Epoch  2/3 Iter  3899]: train_loss = 1.8288
[Epoch  2/3 Iter  4099]: train_loss = 1.7753
[Epoch  2/3 Iter  4299]: train_loss = 1.7768
[Epoch  2/3 Iter  4499]: train_loss = 1.8118
[Epoch  2/3 Iter  4699]: train_loss = 1.8389
[Epoch  2/3 Iter  4899]: train_loss = 1.8338
[Epoch  3/3 Iter  5199]: train_loss = 1.7953
[Epoch  3/3 Iter  5399]: train_loss = 1.7889
[Epoch  3/3 Iter  5599]: train_loss = 1.7601
[Epoch  3/3 Iter  5799]: train_loss = 1.7343
[Epoch  3/3 Iter  5999]: train_loss = 1.7597
[Epoch  3/3 Iter  6199]: train_loss = 1.7446
[Epoch  3/3 Iter  6399]: train_loss = 1.7255
[Epoch  3/3 Iter  6599]: train_loss = 1.7353
[Epoch  3/3 Iter  6799]: train_loss = 1.7630
[Epoch  3/3 Iter  6999]: train_loss = 1.7387
[Epoch  3/3 Iter  7199]: train_loss = 1.7192
[Epoch  3/3 Iter  7399]: train_loss = 1.7349
```

In [62]:

```
writer.close()
```

Scroll up to the Tensorboard cell above and check the plot. It should look like the following figure.



4. Comparing a Scalar Across Different Runs

In the following, we shall use a higher learning rate (0.01). Tensorboard will visualize all experiments with the same **tag** in the same plot. This allows you to compare the results from different experiments in the same figure. In this case, the tag name is `Training loss` .

In [63]:

```
net = build_model()
writer = SummaryWriter('runs/cifar_lr0.01')
```

```
net = train (net, writer, lr = 0.01, num_epochs=3, loop_per_val=200)
writer.close()
```

```
[Epoch  1/3 Iter   199]: train_loss = 2.1447
[Epoch  1/3 Iter   399]: train_loss = 2.0125
[Epoch  1/3 Iter   599]: train_loss = 1.8855
[Epoch  1/3 Iter   799]: train_loss = 1.8521
[Epoch  1/3 Iter   999]: train_loss = 1.8467
[Epoch  1/3 Iter  1199]: train_loss = 1.8424
[Epoch  1/3 Iter  1399]: train_loss = 1.8056
[Epoch  1/3 Iter  1599]: train_loss = 1.7517
[Epoch  1/3 Iter  1799]: train_loss = 1.7759
[Epoch  1/3 Iter  1999]: train_loss = 1.6565
[Epoch  1/3 Iter  2199]: train_loss = 1.7712
[Epoch  1/3 Iter  2399]: train_loss = 1.7342
[Epoch  2/3 Iter  2699]: train_loss = 1.5594
[Epoch  2/3 Iter  2899]: train_loss = 1.7206
[Epoch  2/3 Iter  3099]: train_loss = 1.6957
[Epoch  2/3 Iter  3299]: train_loss = 1.5883
[Epoch  2/3 Iter  3499]: train_loss = 1.5971
[Epoch  2/3 Iter  3699]: train_loss = 1.6264
[Epoch  2/3 Iter  3899]: train_loss = 1.6380
[Epoch  2/3 Iter  4099]: train_loss = 1.5705
[Epoch  2/3 Iter  4299]: train_loss = 1.5917
[Epoch  2/3 Iter  4499]: train_loss = 1.6448
[Epoch  2/3 Iter  4699]: train_loss = 1.6445
[Epoch  2/3 Iter  4899]: train_loss = 1.6004
[Epoch  3/3 Iter  5199]: train_loss = 1.4905
[Epoch  3/3 Iter  5399]: train_loss = 1.4861
[Epoch  3/3 Iter  5599]: train_loss = 1.5028
[Epoch  3/3 Iter  5799]: train_loss = 1.5535
[Epoch  3/3 Iter  5999]: train_loss = 1.5411
[Epoch  3/3 Iter  6199]: train_loss = 1.4869
[Epoch  3/3 Iter  6399]: train_loss = 1.5098
[Epoch  3/3 Iter  6599]: train_loss = 1.5594
[Epoch  3/3 Iter  6799]: train_loss = 1.5518
[Epoch  3/3 Iter  6999]: train_loss = 1.5819
[Epoch  3/3 Iter  7199]: train_loss = 1.5703
[Epoch  3/3 Iter  7399]: train_loss = 1.5660
```

Scroll up to the Tensorboard cell to observe the result. You should see something similar to the figure below. Tensorboard clearly allows you to compare the monitored value for different runs with different settings easily. To load the data automatically, enable "reload data" and set the reload period to the minimum settings of 30.

Settings

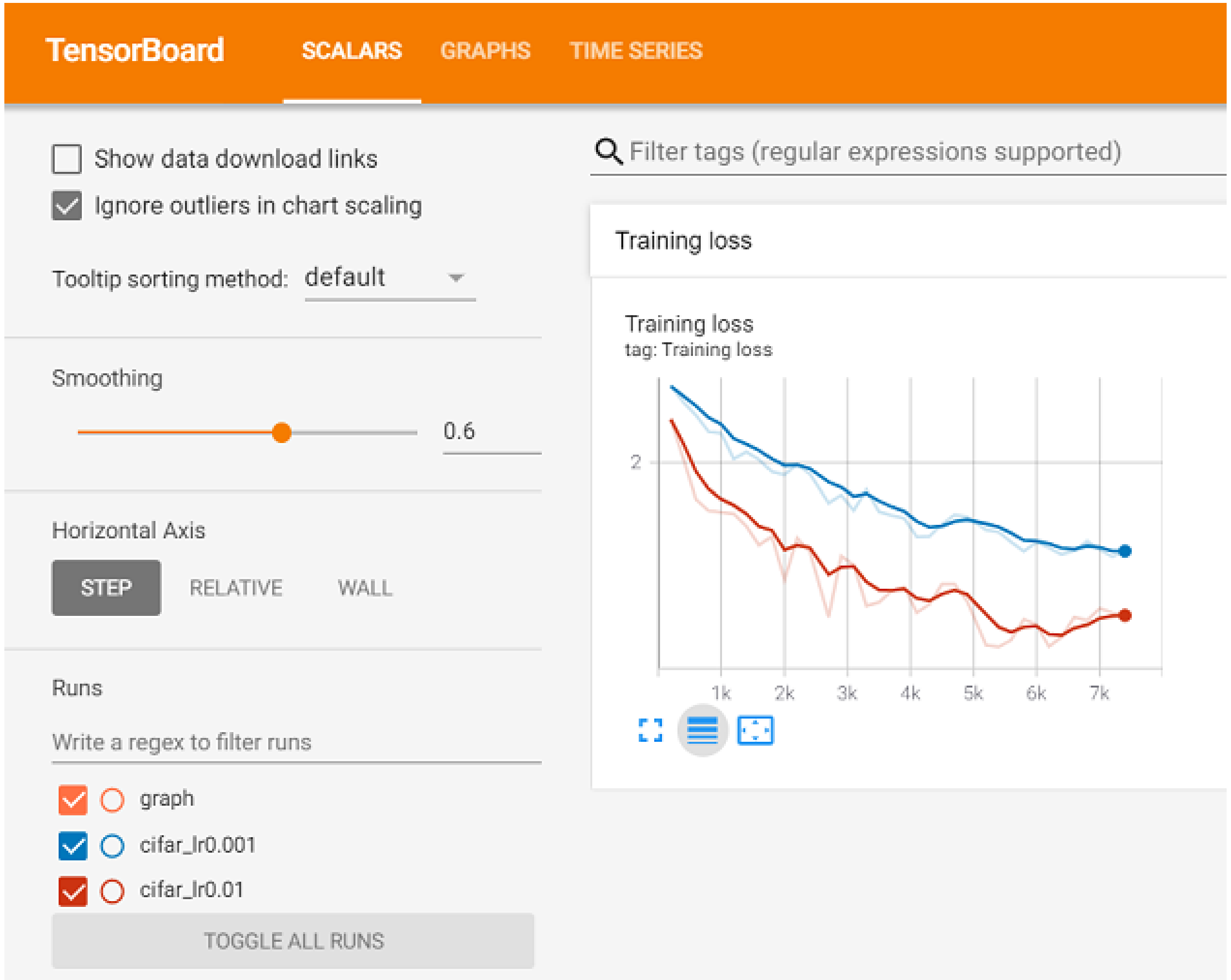
- [x] Reload data

Reload Period

30

Pagination Limit

12



5. Grouping Plots

Tensorboard allows you to log multiple information for one experiment. In the following, we shall log 3 different plots:

1. Train loss
2. Test loss

3. Test error

To avoid cluttering the UI, we can **group** the plots by naming them hierarchically. For example, `Loss/train` and `Loss/test` will be grouped together, while `Error/test` will be grouped separately in the TensorBoard interface.

First, let's create our evaluation function to compute the error of a given dataset.

In [73]:

```
def evaluate(model, dataloader):
    num_wrong = 0
    test_loss = 0
    total = 0

    criterion = nn.CrossEntropyLoss()

    with torch.no_grad():
        for X, Y in dataloader:
            # transfer input to GPU
            if torch.cuda.is_available():
                X = X.cuda()
                Y = Y.cuda()

            Yhat = model(X)

            # compute test loss
            test_loss += criterion(Yhat, Y).item()

            # compute error
            _, predicted = Yhat.max(axis=1)
            num_wrong += (predicted != Y).sum().item()

            total += len(Y)

    return num_wrong/total, test_loss/total
```

The following code computes the (1) training loss, (2) testing loss and (3) testing error every 100 (batch) iteration. Each time it computes the loss, it will then update tensorboard by calling the command `writer.add_scaler` (lines 56-58) for every values that we want to monitor.

In [74]:

```
def train2 (model, writer, lr=0.001, num_epochs=5, loop_per_val=200):

    # set the optimizer
```

```
    optimizer = optim.SGD(net.parameters(), lr=lr)

    # set the criterion
    criterion = nn.CrossEntropyLoss()

    # set to training mode
    net.train()

    # train the model
    for e in range(num_epochs):

        running_loss = 0
        running_count = 0

        for i, (inputs, labels) in enumerate(trainloader):

            # transfer input to GPU
            if torch.cuda.is_available():
                inputs = inputs.cuda()
                labels = labels.cuda()

            # set grad to zero
            net.zero_grad()

            # forward propagation
            outputs = net(inputs)

            # compute loss
            loss = criterion(outputs, labels)

            # backward propagation
            loss.backward()

            # update model
            optimizer.step()

            # compute running loss and validation
            running_loss += loss.item()
            running_count +=1

            # display the averaged loss value every 100 cycles
            if i % loop_per_val == loop_per_val - 1:
                train_loss = running_loss / loop_per_val
                running_loss = 0.
```

```
                running_count = 0

                test_error, test_loss = evaluate(net, testloader)

                print(f'[Epoch {e+1:2d}/{num_epochs:d} Iter {e*len(trainloader)+i}]: train_loss = {train_loss:.4f}, test_loss = {t

                # Update tensorboard
                writer.add_scalar("Group1_Loss/training loss", train_loss, e*len(trainloader)+i)
                writer.add_scalar("Group1_Loss/testing loss", test_loss, e*len(trainloader)+i)
                writer.add_scalar("Group2_Error/testing error", test_error, e*len(trainloader)+i)

    return model
```

Now, let's build the model and log the training loss, test loss and test error.

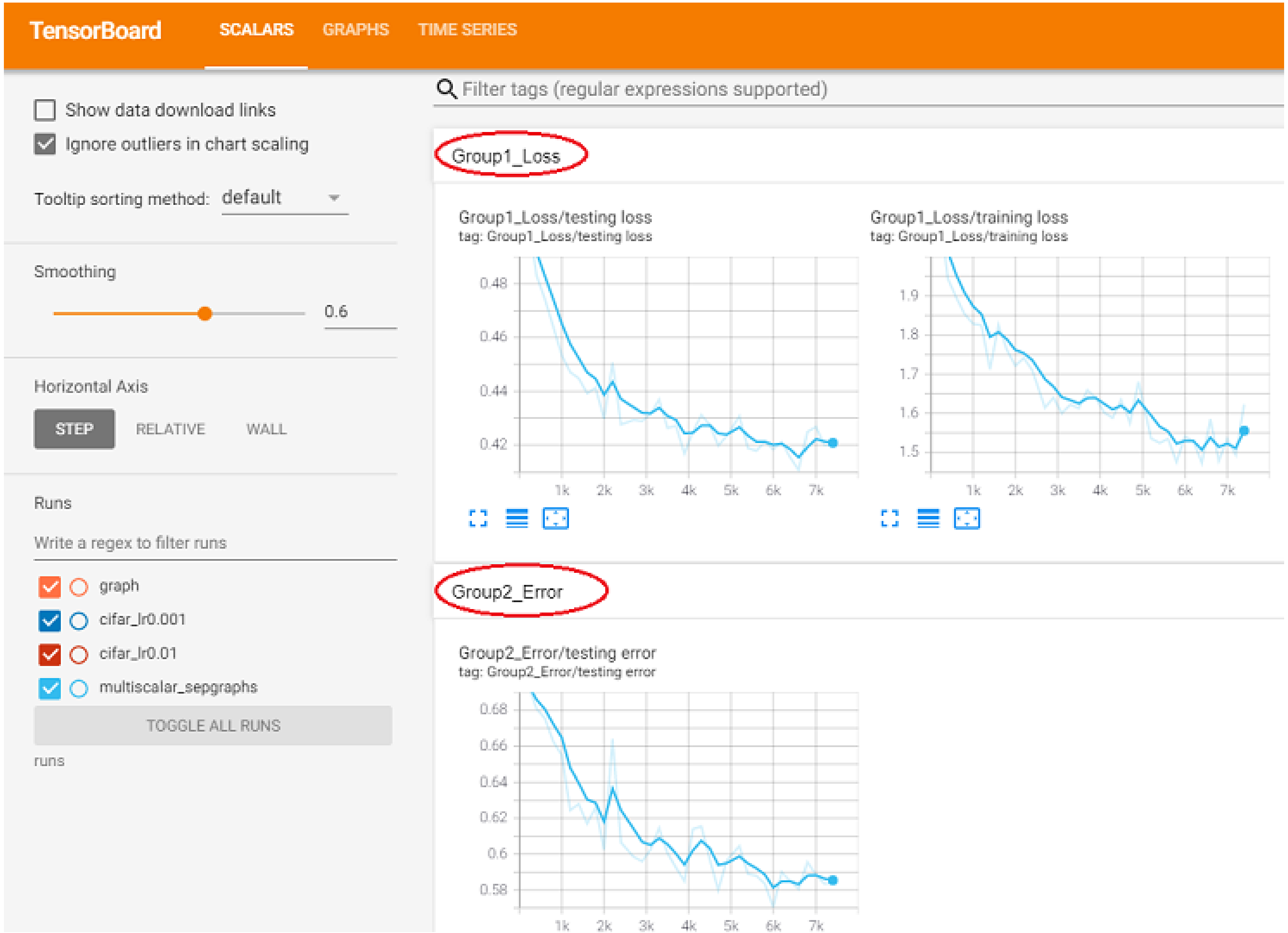
In [75]:

```
net = build_model()
writer = SummaryWriter('runs/multiscalar_sepgraphs')
net = train2 (net, writer, lr = 0.01, num_epochs=3, loop_per_val=200)
writer.close()
```

```
[Epoch  1/3 Iter   200/2500]: train_loss = 2.1300, test_loss = 0.5063 | test error: 0.6945
[Epoch  1/3 Iter   400/2500]: train_loss = 1.9450, test_loss = 0.4832 | test error: 0.6810
[Epoch  1/3 Iter   600/2500]: train_loss = 1.8970, test_loss = 0.4743 | test error: 0.6755
[Epoch  1/3 Iter   800/2500]: train_loss = 1.8536, test_loss = 0.4640 | test error: 0.6625
[Epoch  1/3 Iter  1000/2500]: train_loss = 1.8288, test_loss = 0.4534 | test error: 0.6550
[Epoch  1/3 Iter  1200/2500]: train_loss = 1.8248, test_loss = 0.4471 | test error: 0.6245
[Epoch  1/3 Iter  1400/2500]: train_loss = 1.7121, test_loss = 0.4449 | test error: 0.6280
[Epoch  1/3 Iter  1600/2500]: train_loss = 1.8256, test_loss = 0.4392 | test error: 0.6170
[Epoch  1/3 Iter  1800/2500]: train_loss = 1.7620, test_loss = 0.4413 | test error: 0.6260
[Epoch  1/3 Iter  2000/2500]: train_loss = 1.7216, test_loss = 0.4298 | test error: 0.6025
[Epoch  1/3 Iter  2200/2500]: train_loss = 1.7413, test_loss = 0.4507 | test error: 0.6640
[Epoch  1/3 Iter  2400/2500]: train_loss = 1.7100, test_loss = 0.4276 | test error: 0.6065
[Epoch  2/3 Iter   200/2500]: train_loss = 1.6143, test_loss = 0.4294 | test error: 0.5985
[Epoch  2/3 Iter   400/2500]: train_loss = 1.6400, test_loss = 0.4289 | test error: 0.5960
[Epoch  2/3 Iter   600/2500]: train_loss = 1.6002, test_loss = 0.4316 | test error: 0.6025
[Epoch  2/3 Iter   800/2500]: train_loss = 1.6224, test_loss = 0.4369 | test error: 0.6145
[Epoch  2/3 Iter  1000/2500]: train_loss = 1.6128, test_loss = 0.4263 | test error: 0.6005
[Epoch  2/3 Iter  1200/2500]: train_loss = 1.6591, test_loss = 0.4272 | test error: 0.5925
[Epoch  2/3 Iter  1400/2500]: train_loss = 1.6398, test_loss = 0.4168 | test error: 0.5850
[Epoch  2/3 Iter  1600/2500]: train_loss = 1.6019, test_loss = 0.4249 | test error: 0.6140
[Epoch  2/3 Iter  1800/2500]: train_loss = 1.5882, test_loss = 0.4312 | test error: 0.6155
[Epoch  2/3 Iter  2000/2500]: train_loss = 1.6342, test_loss = 0.4275 | test error: 0.5970
```

```
[Epoch  2/3 Iter  2200/2500]: train_loss = 1.5751, test_loss = 0.4198 | test error: 0.5800
[Epoch  2/3 Iter  2400/2500]: train_loss = 1.6807, test_loss = 0.4233 | test error: 0.5965
[Epoch  3/3 Iter   200/2500]: train_loss = 1.5356, test_loss = 0.4308 | test error: 0.6045
[Epoch  3/3 Iter   400/2500]: train_loss = 1.5242, test_loss = 0.4189 | test error: 0.5890
[Epoch  3/3 Iter   600/2500]: train_loss = 1.5350, test_loss = 0.4179 | test error: 0.5880
[Epoch  3/3 Iter   800/2500]: train_loss = 1.4753, test_loss = 0.4212 | test error: 0.5835
[Epoch  3/3 Iter  1000/2500]: train_loss = 1.5407, test_loss = 0.4184 | test error: 0.5705
[Epoch  3/3 Iter  1200/2500]: train_loss = 1.5286, test_loss = 0.4211 | test error: 0.5900
[Epoch  3/3 Iter  1400/2500]: train_loss = 1.4728, test_loss = 0.4154 | test error: 0.5850
[Epoch  3/3 Iter  1600/2500]: train_loss = 1.5848, test_loss = 0.4108 | test error: 0.5805
[Epoch  3/3 Iter  1800/2500]: train_loss = 1.4785, test_loss = 0.4249 | test error: 0.5955
[Epoch  3/3 Iter  2000/2500]: train_loss = 1.5344, test_loss = 0.4268 | test error: 0.5885
[Epoch  3/3 Iter  2200/2500]: train_loss = 1.4938, test_loss = 0.4200 | test error: 0.5835
[Epoch  3/3 Iter  2400/2500]: train_loss = 1.6227, test_loss = 0.4200 | test error: 0.5840
```

Scroll up to the Tensorboard cell to observe the result. You should see something similar to the figure below.



---

6. Plotting Multiple Plots in the Same Figure

You can also plot multiple scalar data in the same summary. To do that. You need to use the command `writer.addscalars` rather than `writer.addscalar` .

In [77]:
```
def train3 (model, writer, lr=0.001, num_epochs=5, loop_per_val=200):

    # set the optimizer
    optimizer = optim.SGD(net.parameters(), lr=lr)

    # set the criterion
    criterion = nn.CrossEntropyLoss()

    # set to training mode
    net.train()


    # train the model
    for e in range(num_epochs):

        running_loss = 0
        running_count = 0

        for i, (inputs, labels) in enumerate(trainloader):

            # transfer input to GPU
            if torch.cuda.is_available():
                inputs = inputs.cuda()
                labels = labels.cuda()

            # set grad to zero
            net.zero_grad()

            # forward propagation
            outputs = net(inputs)

            # compute loss
            loss = criterion(outputs, labels)

            # backward propagation
            loss.backward()

            # update model
            optimizer.step()

            # compute running loss and validation
            running_loss += loss.item()
            running_count +=1
```

```
            # display the averaged loss value every 100 cycles
            if i % loop_per_val == loop_per_val - 1:
                train_loss = running_loss / loop_per_val
                running_loss = 0.
                running_count = 0

                _, test_loss = evaluate(net, testloader)

                print(f'[Epoch {e+1:2d}/{num_epochs:d} Iter {e*len(trainloader)+i}]: train_loss = {train_loss:.4f}, test_loss = {t

                # Update tensorboard
                writer.add_scalars('combined', {'train_loss': train_loss,
                                                'test_loss': test_loss}, e*len(trainloader)+i)

    return model
```

Now, let's build the model and log the training loss and test loss.

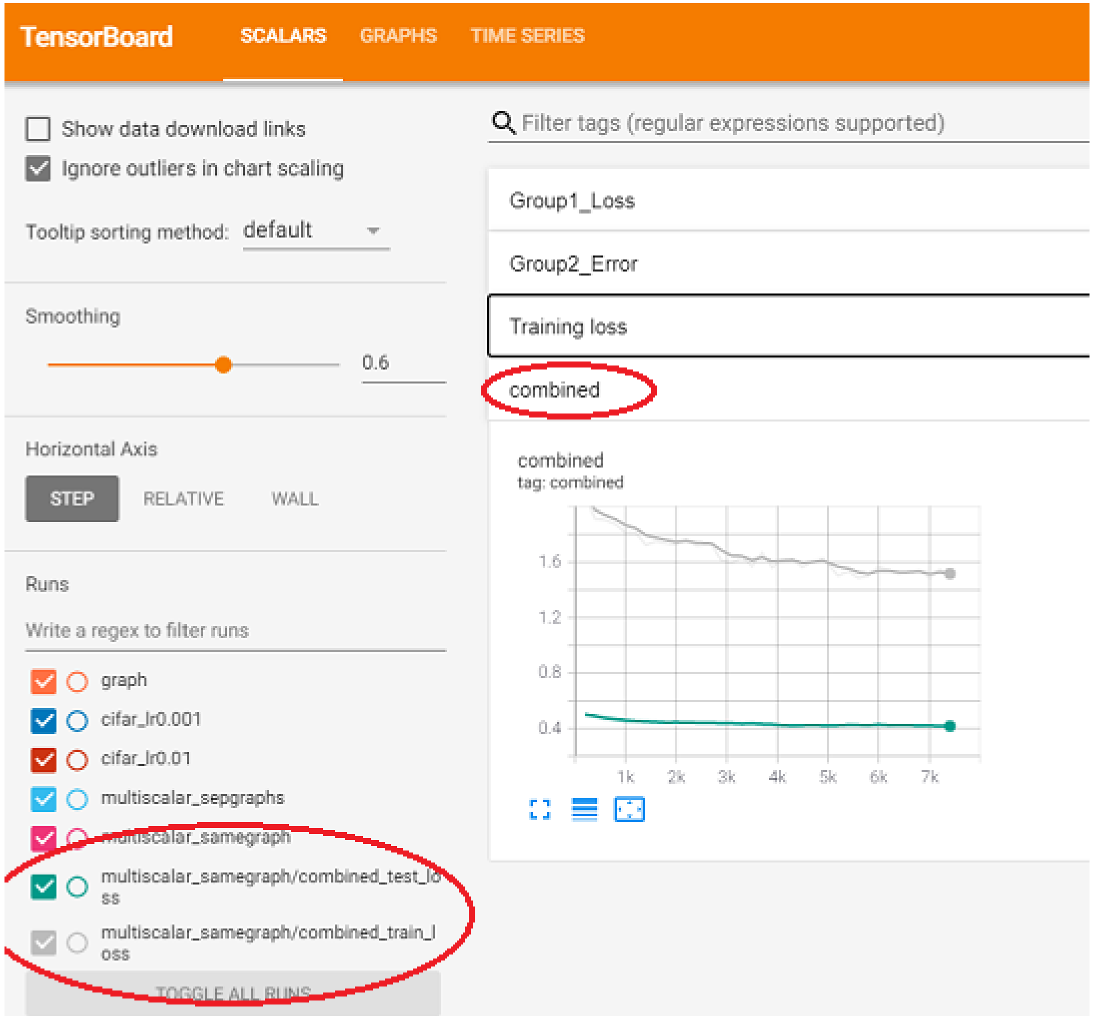
In [79]:

```
net = build_model()
writer = SummaryWriter('runs/multiscalar_samegraph')
net = train3 (net, writer, lr = 0.01, num_epochs=3, loop_per_val=200)
writer.close()
```

```
[Epoch  1/3 Iter   200/2500]: train_loss = 2.1144, test_loss = 0.5067
[Epoch  1/3 Iter   400/2500]: train_loss = 1.9488, test_loss = 0.4768
[Epoch  1/3 Iter   600/2500]: train_loss = 1.8901, test_loss = 0.4655
[Epoch  1/3 Iter   800/2500]: train_loss = 1.7990, test_loss = 0.4591
[Epoch  1/3 Iter  1000/2500]: train_loss = 1.7821, test_loss = 0.4588
[Epoch  1/3 Iter  1200/2500]: train_loss = 1.8275, test_loss = 0.4530
[Epoch  1/3 Iter  1400/2500]: train_loss = 1.7208, test_loss = 0.4498
[Epoch  1/3 Iter  1600/2500]: train_loss = 1.7355, test_loss = 0.4377
[Epoch  1/3 Iter  1800/2500]: train_loss = 1.7926, test_loss = 0.4365
[Epoch  1/3 Iter  2000/2500]: train_loss = 1.7570, test_loss = 0.4312
[Epoch  1/3 Iter  2200/2500]: train_loss = 1.7719, test_loss = 0.4440
[Epoch  1/3 Iter  2400/2500]: train_loss = 1.7608, test_loss = 0.4365
[Epoch  2/3 Iter   200/2500]: train_loss = 1.6473, test_loss = 0.4352
[Epoch  2/3 Iter   400/2500]: train_loss = 1.6725, test_loss = 0.4281
[Epoch  2/3 Iter   600/2500]: train_loss = 1.5417, test_loss = 0.4406
[Epoch  2/3 Iter   800/2500]: train_loss = 1.6487, test_loss = 0.4380
[Epoch  2/3 Iter  1000/2500]: train_loss = 1.6937, test_loss = 0.4336
[Epoch  2/3 Iter  1200/2500]: train_loss = 1.6178, test_loss = 0.4268
[Epoch  2/3 Iter  1400/2500]: train_loss = 1.5901, test_loss = 0.4260
```

```
[Epoch  2/3 Iter  1600/2500]: train_loss = 1.6707, test_loss = 0.4315
[Epoch  2/3 Iter  1800/2500]: train_loss = 1.6844, test_loss = 0.4273
[Epoch  2/3 Iter  2000/2500]: train_loss = 1.5781, test_loss = 0.4281
[Epoch  2/3 Iter  2200/2500]: train_loss = 1.5894, test_loss = 0.4180
[Epoch  2/3 Iter  2400/2500]: train_loss = 1.6271, test_loss = 0.4222
[Epoch  3/3 Iter   200/2500]: train_loss = 1.5054, test_loss = 0.4287
[Epoch  3/3 Iter   400/2500]: train_loss = 1.4911, test_loss = 0.4176
[Epoch  3/3 Iter   600/2500]: train_loss = 1.5010, test_loss = 0.4134
[Epoch  3/3 Iter   800/2500]: train_loss = 1.5056, test_loss = 0.4175
[Epoch  3/3 Iter  1000/2500]: train_loss = 1.5114, test_loss = 0.4230
[Epoch  3/3 Iter  1200/2500]: train_loss = 1.5527, test_loss = 0.4416
[Epoch  3/3 Iter  1400/2500]: train_loss = 1.5537, test_loss = 0.4209
[Epoch  3/3 Iter  1600/2500]: train_loss = 1.5315, test_loss = 0.4139
[Epoch  3/3 Iter  1800/2500]: train_loss = 1.5627, test_loss = 0.4330
[Epoch  3/3 Iter  2000/2500]: train_loss = 1.5188, test_loss = 0.4167
[Epoch  3/3 Iter  2200/2500]: train_loss = 1.5444, test_loss = 0.4262
[Epoch  3/3 Iter  2400/2500]: train_loss = 1.5945, test_loss = 0.4136
```

Scroll up to the Tensorboard cell to observe the result. You should see something similar to the figure below.



---

8. Other uses of Tensorboard

Besides, you can explore the following output to tensorboard:

1. Displaying images via the `writer.add_image` method
2. Displaying audio via the `writer.add_audio` method

3. Displaying video via the `writer.add_video` method
4. Displaying histogram via the `writer.add_histogram` method
5. Disualize the lower dimensional representation of higher dimensional data through the `add_embedding` method