Name: Ayoung Kang
ID: 933-610-350
kanga2@oregonstate.edu

Homework Assignment 3
CS 535 Deep Learning

2021-05-27

1 Question 1

Add a batch normalization layer after the first fully-connected layer(fc1) (8 points). Save the model after training(Checkout our tutorial on how to save your model). Be careful that batch normalization layer performs differently between training and evaluation process, make sure you understand how to convert your model between training mode and evaluation mode(you can find hints in my code). Observe the difference of final training/testing accuracy with/without batch normalization layer.

1.1 Code

A batch normalization layer was inserted after the first fully-connected layer (before non-linearity).

```
# Question1
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.batchnorm = nn.BatchNorm1d(512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = self.fc1(x)
        x = self.batchnorm(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

1.2 Accuracy/Loss

Figure 1 and Figure 2 show the train/test accuracy and loss without/with a batch normalization layer, respectively. Compared to the train/test accuracies without the batch normalization layer, having the batch normalization layer resulted in an increase in the train accuracy from 0.9652 to 0.9926, a 2.74% improvement, and an increase in the test accuracy from 0.7332 to 0.7813, a 4.81% improvement.

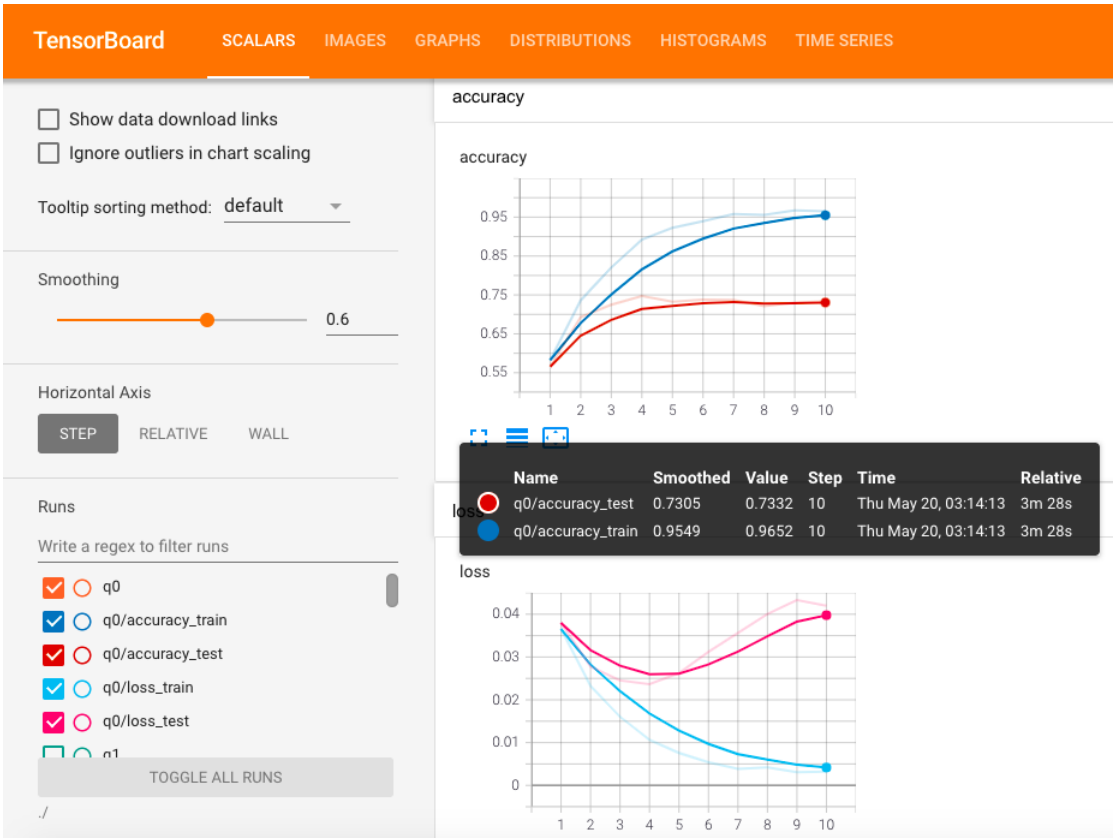

Figure 1: Train/test accuracy and loss without a batch normalization layer

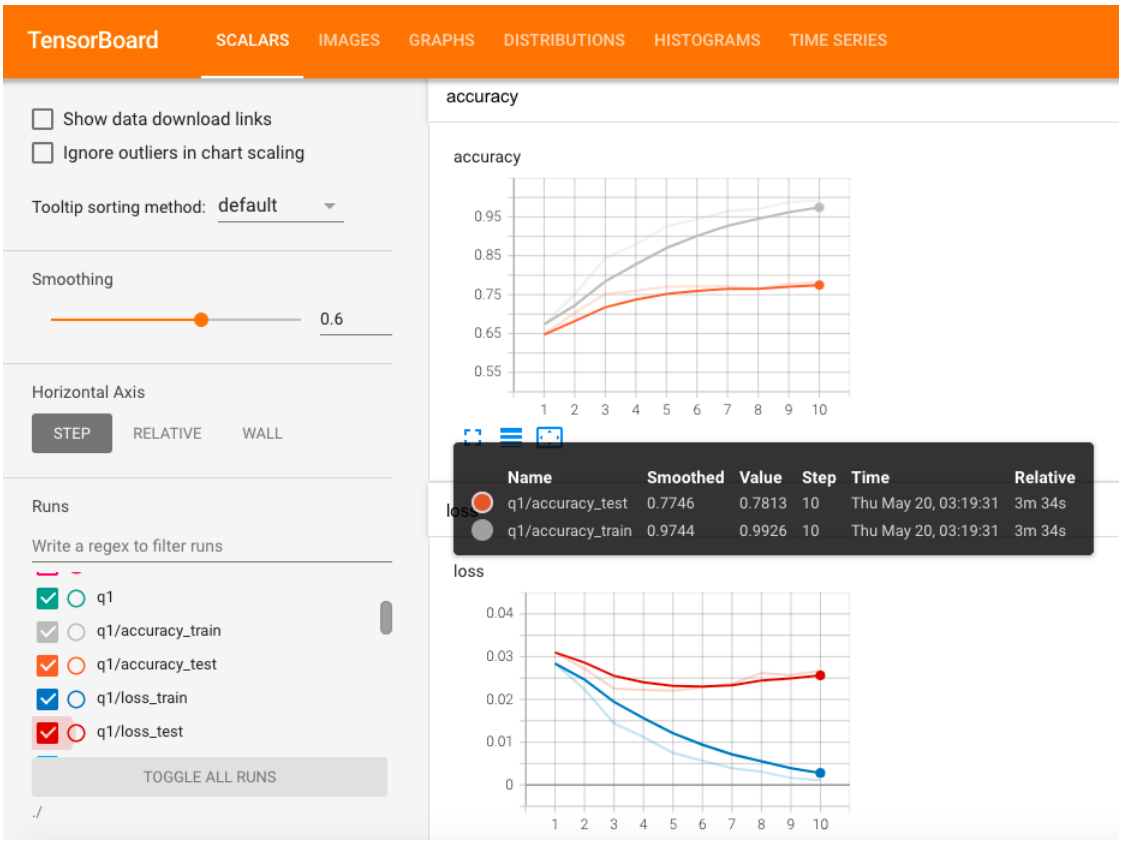

Figure 2: Train/test accuracy and loss with a batch normalization layer

2 Question 2

Modify our model by adding another fully connected layer with 512 nodes at the second-to-last layer (before the fc2 layer) (8 points). Apply the model weights you saved at step 1 to initialize to the new model(only up to fc2 layer since after that all layers are newly created) before training. Train and save the model (Hint: check the end of the assignment description to see how to partially restore weights from a pretrained weights file).

2.1 Code

Another fully connected layer was added before the fc2 layer. The model weights saved at step 1 was applied to this model.

```
# Question 2
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.batchnorm = nn.BatchNorm1d(512)
        self.fc_additional = nn.Linear(512, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = self.fc1(x)
        x = self.batchnorm(x)
        x = F.relu(x)
        x = F.relu(self.fc_additional(x))
        x = self.fc2(x)
        return x

if __name__ == "__main__":
    EXPT = input("Experiment>> ")
    print('Building model...')
    net = Net().cuda()
    net.train() # Why would I do this?

    pretrained_dict = torch.load('mytraining_q1.pth')
    model_dict = net.state_dict()
    temp = {key: val for key, val in pretrained_dict.items() if key in model_dict}
    model_dict.update(temp)
    net.load_state_dict(model_dict)
```

2.2 Accuracy/Loss

Figure 3 shows the train/test accuracy and loss with another fully connected layer. Since we applied the model weights we saved in question 1, the train/test accuracies are relatively high from epoch 1. It seems that having another fully connected layer does not have a huge impact on the network performance.

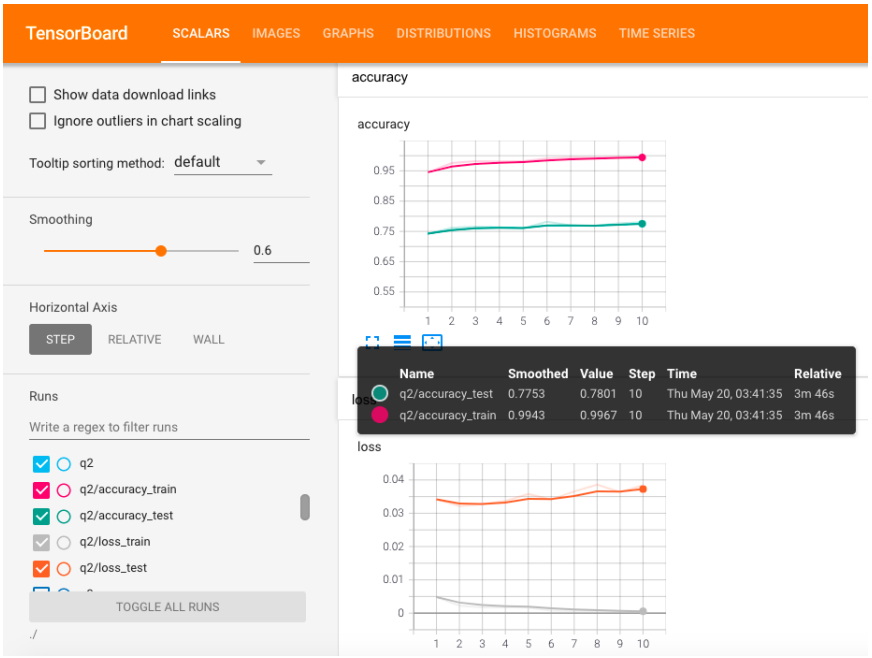

Figure 3: Train/test accuracy and loss with another fully connected layer

3 Question 3

Try to use an adaptive schedule to tune the learning rate, you can choose from RMSprop, Adagrad and Adam (Hint: you don't need to implement any of these, look at Pytorch documentation please) (8 points).

3.1 Code

We used the model in Question1 and torch.optim.Adam() as an optimizer.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

3.2 Accuracy/Loss

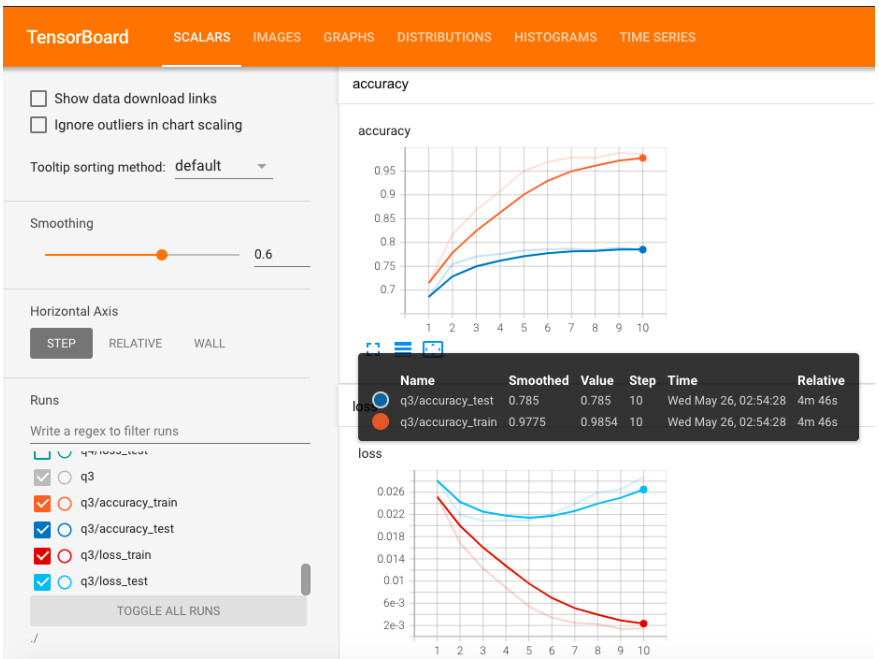

Figure 4: Train/test accuracy and loss when using Adam

As can be seen in Figure 5, when using Adam optimizer, our network converges faster because Adam uses momentum and adaptive learning rates.

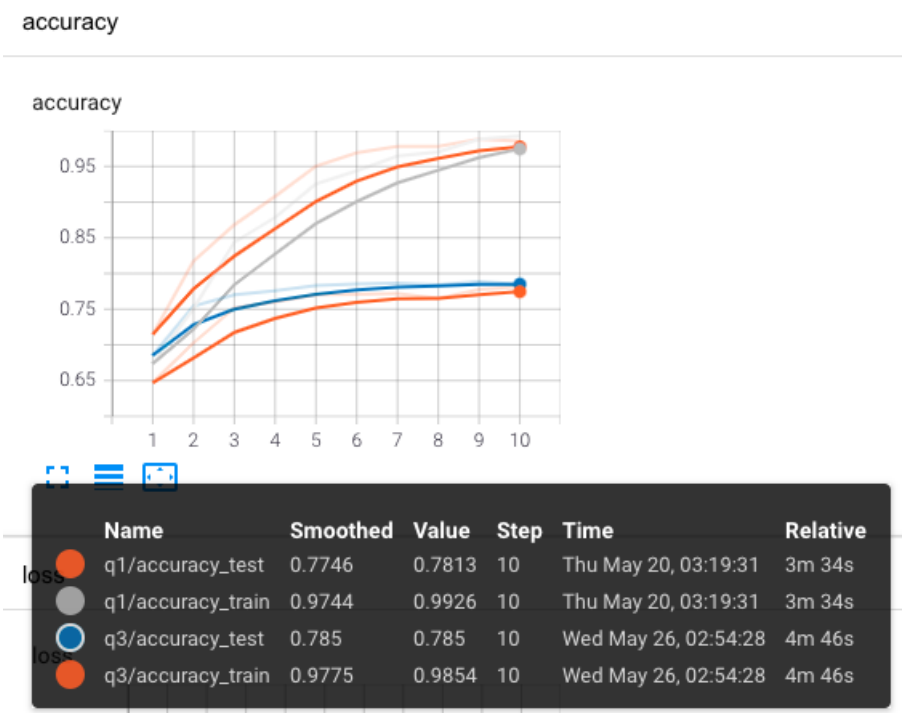

Figure 5: Train/test accuracy comparison

4 Question 4

Try to tune your network in another way (e.g. add/remove a layer, change the activation function, add/remove regularizer, change the number of hidden units, more batch normalization layers) not described in the previous four. You can start from random initialization or previous results as you wish (8 points).

4.1 Code

1. Two more convolutional layers followed by a pooling layer were added.
2. A batch normalization layer was inserted after every conv layer.

```
# Question 4
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(64)
        self.conv5 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn5 = nn.BatchNorm2d(128)
        self.conv6 = nn.Conv2d(128, 128, 3, padding=1)
        self.bn6 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.batchnorm = nn.BatchNorm1d(512)
```

```
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(self.bn1(x))
        x = self.conv2(x)
        x = F.relu(self.bn2(x))
        x = self.pool(x)
        x = self.conv3(x)
        x = F.relu(self.bn3(x))
        x = self.conv4(x)
        x = F.relu(self.bn4(x))
        x = self.pool(x)
        x = self.conv5(x)
        x = F.relu(self.bn5(x))
        x = self.conv6(x)
        x = F.relu(self.bn6(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = self.fc1(x)
        x = F.relu(self.batchnorm(x))
        x = self.fc2(x)
        return x
```

4.2 Accuracy/Loss

With the model above, the test accuracy increased to 0.843 as can be seen in Figure 6.

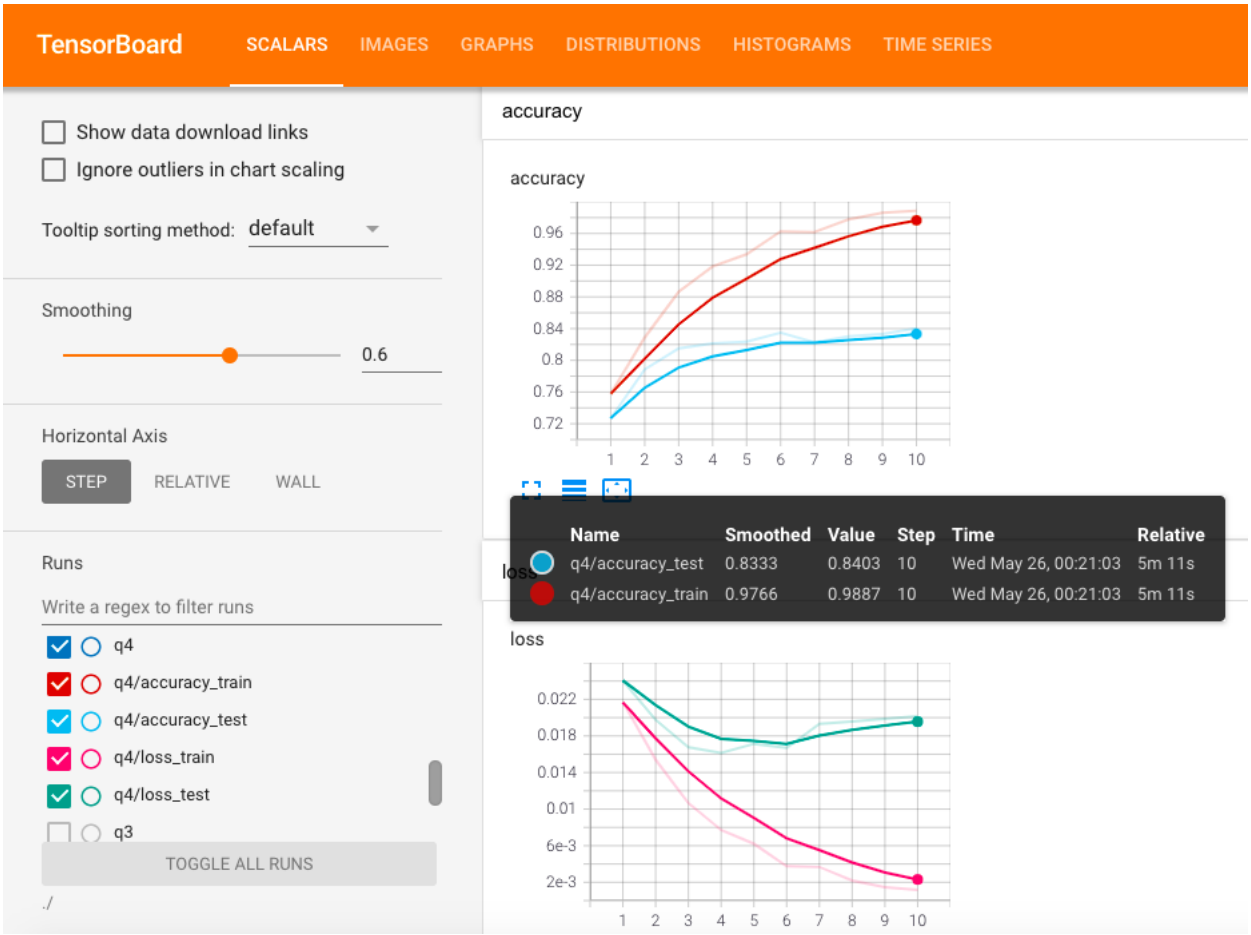

Figure 6: Train/test accuracy and loss

5 Question 5

Try to use the visualization toolkit, tensorboard, for tracking and visualizing your training process and include them in your report: show the loss and accuracy, visualize the model graph, and display images or other tensors as they change over time. (8 points).

5.1 Code

```
def show_img(img):
    # unnormalize the images
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    return npimg # return the unnormalized images

dataiter = iter(trainloader)
images, labels = dataiter.next()

img_grid = torchvision.utils.make_grid(images)
img_grid = show_img(img_grid)
writer.add_image('CIFAR10', img_grid, 0)
writer.add_graph(net, images)
net = net.cuda()

writer.add_scalars("accuracy", {
            "train": train_acc,
            "test": test_acc
        }, epoch + 1)
writer.add_scalars("loss", {
        "train": train_loss,
        "test": test_loss
    }, epoch + 1)
writer.add_histogram("conv1.bias", net.conv1.bias, epoch + 1)
writer.add_histogram("conv1.weight", net.conv1.weight, epoch + 1)
writer.add_histogram("conv2.bias", net.conv2.bias, epoch + 1)
writer.add_histogram("conv2.weight", net.conv2.weight, epoch + 1)
```

5.2 Accuracy/Loss

See Figure 1 - Figure 6, please.

5.3 Model Graph

The model graph for the setting in Question 4.

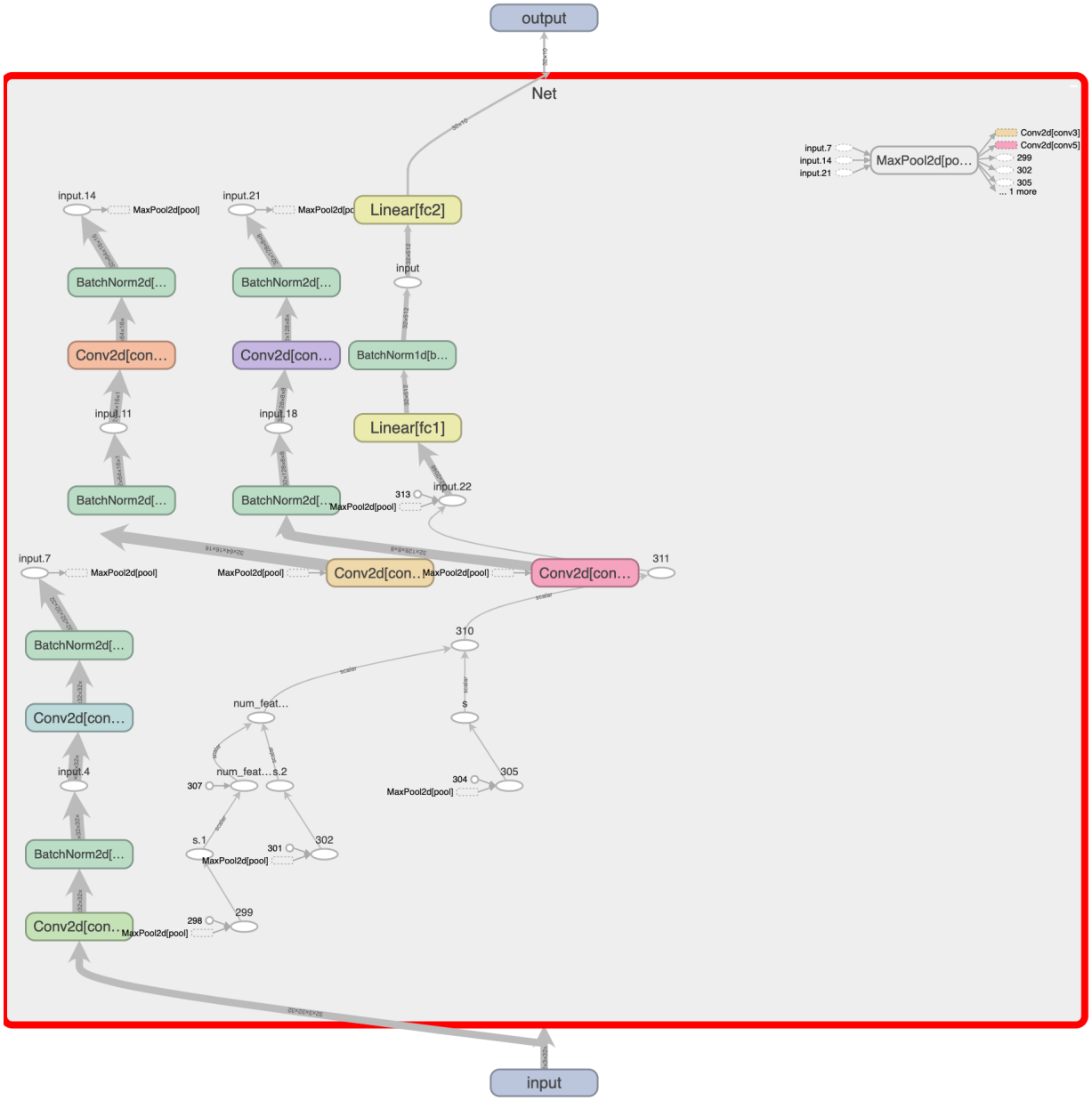

Figure 7: Model Graph

5.4 Images

We unnormalized the images before adding them to TensorBoard to properly visualize it.

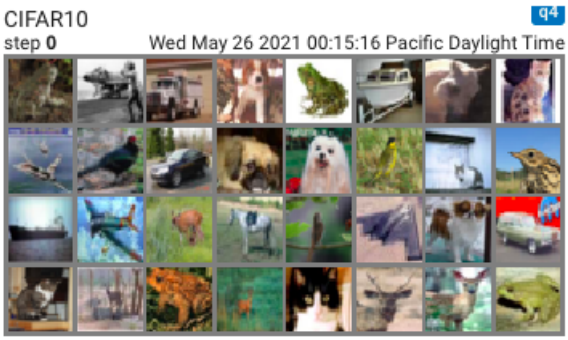

Figure 8: Images

5.5 Tensors

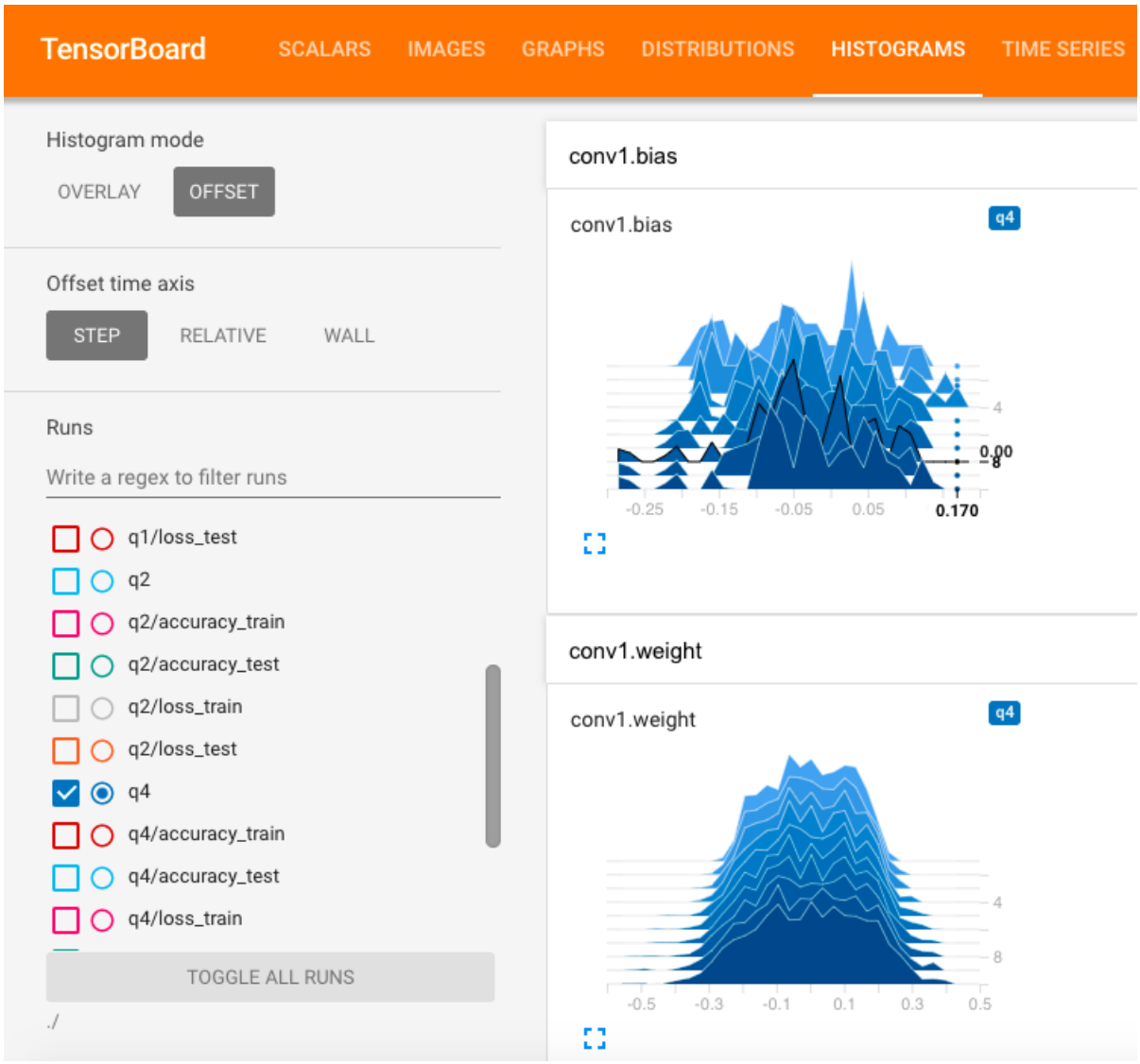

Figure 9: Tensors