Homework 10 - CIFAR10 Image Classification with PyTorch

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947

Enter your authorization code:
··········
Mounted at /content/drive/
```

About

The goal of the homework is to train a convolutional neural network on the standard CIFAR10 image classfication dataset.

When solving machine learning tasks using neural networks, one typically starts with a simple network architecture and then improves the network by adding new layers, retraining, adjusting parameters, retraining, etc. We attempt to illustrate this process below with several architecture improvements.

Dev Environment

Working on Google Colab

You may choose to work locally or on Google Colaboratory. You have access to free compute through this service. Colab is recommended since it will be setup correctly and will have access to GPU resources.

1. Visit https://colab.research.google.com/drive
2. Navigate to the **`Upload`** tab, and upload your `HW10.ipynb`
3. Now on the top right corner, under the `Comment` and `Share` options, you should see a `Connect` option. Once you are connected, you will have access to a VM with 12GB RAM, 50 GB disk space and a single GPU. The dropdown menu will allow you to connect to a local runtime as well.

Notes:

- **If you do not have a working setup for Python 3, this is your best bet. It will also save you from heavy installations like `tensorflow` if you don't want to deal with those.**
- ***There is a downside*. You can only use this instance for a single 12-hour stretch, after which your data will be deleted, and you would have redownload all your datasets, any libraries not already on the VM, and regenerate your logs.**

Installing PyTorch and Dependencies

The instructions for installing and setting up PyTorch can be found at https://pytorch.org/get-started/locally/. Make sure you follow the instructions for your machine. For any of the remaining libraries used in this assignment:

- We have provided a `hw8_requirements.txt` file on the homework web page.
- Download this file, and in the same directory you can run `pip3 install -r hw8_requirements.txt` Check that PyTorch installed correctly by running the following:

```
import torch
torch.rand(5, 3)
```

```
tensor([[0.1704, 0.0847, 0.1817],
        [0.8969, 0.9047, 0.7559],
        [0.0716, 0.1528, 0.7472],
        [0.0340, 0.8298, 0.8449],
        [0.6746, 0.7327, 0.4504]])
```

Part 0 Imports and Basic Setup (5 Points)

First, import the required libraries as follows. The libraries we will use will be the same as those in HW8.

```
import numpy as np
import torch
from torch import nn
from torch import optim

import matplotlib.pyplot as plt
```

GPU Support

Training of large network can take a long time. PyTorch supports GPU with just a small amount of effort.

When creating our networks, we will call `net.to(device)` to tell the network to train on the GPU, if one is available. Note, if the network utilizes the GPU, it is important that any tensors we use with it (such as the data) also reside on the CPU. Thus, a call like `images = images.to(device)` is necessary with any data we want to use with the GPU.

Note: If you can't get access to a GPU, don't worry to much. Since we use very small networks, the difference between CPU and GPU isn't large and in some cases GPU will actually be slower.

```
import torch.cuda as cuda

# Use a GPU, i.e. cuda:0 device if it available.
device = torch.device("cuda:0" if cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

Training Code

```
import time
```

```
class Flatten(nn.Module):
  """NN Module that flattens the incoming tensor."""
  def forward(self, input):
    return input.view(input.size(0), -1)

def train(model, train_loader, test_loader, loss_func, opt, num_epochs=10):
  all_training_loss = np.zeros((0,2))
  all_training_acc = np.zeros((0,2))
  all_test_loss = np.zeros((0,2))
  all_test_acc = np.zeros((0,2))

  training_step = 0
  training_loss, training_acc = 2.0, 0.0
  print_every = 1000

  start = time.clock()

  for i in range(num_epochs):
    epoch_start = time.clock()

    model.train()
    for images, labels in train_loader:
      images, labels = images.to(device), labels.to(device)
      opt.zero_grad()

      preds = model(images)
      loss = loss_func(preds, labels)
      loss.backward()
      opt.step()

      training_loss += loss.item()
      training_acc += (torch.argmax(preds, dim=1)==labels).float().mean()

      if training_step % print_every == 0:
        training_loss /= print_every
        training_acc /= print_every

        all_training_loss = np.concatenate((all_training_loss, [[training_step, training_los
        all_training_acc = np.concatenate((all_training_acc, [[training_step, training_acc]

        print('  Epoch %d @ step %d: Train Loss: %3f, Train Accuracy: %3f' % (
            i, training_step, training_loss, training_acc))
        training_loss, training_acc = 0.0, 0.0

      training_step+=1

    model.eval()
    with torch.no_grad():
      validation_loss, validation_acc = 0.0, 0.0
      count = 0
      for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        validation_loss+=loss_func(output,labels)
        validation_acc+=(torch.argmax(output, dim=1) == labels).float().mean()
        count += 1
      validation_loss/=count
      validation_acc/=count

      all_test_loss = np.concatenate((all_test_loss, [[training_step, validation_loss]]))
      all_test_acc = np.concatenate((all_test_acc, [[training_step, validation_acc]]))

      epoch_time = time.clock() - epoch_start

      print('Epoch %d Test Loss: %3f, Test Accuracy: %3f, time: %.1fs' % (
          i, validation_loss, validation_acc, epoch_time))

  total_time = time.clock() - start
  print('Final Test Loss: %3f, Test Accuracy: %3f, Total time: %.1fs' % (
      validation_loss, validation_acc, total_time))

  return {'loss': { 'train': all_training_loss, 'test': all_test_loss },
          'accuracy': { 'train': all_training_acc, 'test': all_test_acc }}

def plot_graphs(model_name, metrics):
  for metric, values in metrics.items():
    for name, v in values.items():
      plt.plot(v[:,0], v[:,1], label=name)
    plt.title(f'{metric} for {model_name}')
    plt.legend()
    plt.xlabel("Training Steps")
    plt.ylabel(metric)
    plt.show()
```

Load the **CIFA-10** dataset and define the transformations. You may also want to print its structure, size, as well as sample a few images to get a sense of how to design the network.

```
!mkdir hw10_data
```

```
# Download the data.
from torchvision import datasets, transforms

transformations = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
train_set = datasets.CIFAR10(root='hw10_data/', download=True, transform=transformations)
test_set = datasets.CIFAR10(root='hw10_data/', download=True, train=False, transform=transfo
```

```
0it [00:00, ?it/s]Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
 99%|██████████▉| 169500672/170498071 [00:12<00:00, 13168614.81it/s]Files already dow
```

Use `DataLoader` to create a loader for the training set and a loader for the testing set. You can use a `batch_size` of 8 to start, and change it if you wish.

```
from torch.utils.data import DataLoader

batch_size = 8
train_loader = torch.utils.data.DataLoader(train_set, batch_size, shuffle=True, num_workers=
test_loader = torch.utils.data.DataLoader(test_set, batch_size, shuffle=True, num_workers=2)

input_shape = np.array(train_set[0][0]).shape
input_dim = input_shape[1]*input_shape[2]*input_shape[0]
```

```
training_epochs = 5
```

Part 1 CIFAR10 with Fully Connected Neural Netowrk (25 Points)

As a warm-up, let's begin by training a two-layer fully connected neural network model on **CIFAR-10** dataset. You may go back to check HW8 for some basics.

We will give you this code to use as a baseline to compare against your CNN models.

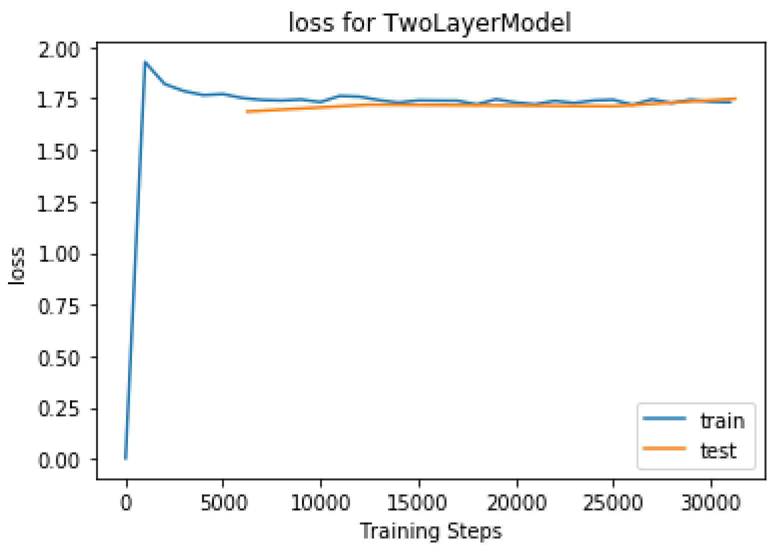
```
class TwoLayerModel(nn.Module):
  def __init__(self):
    super(TwoLayerModel, self).__init__()
    self.net = nn.Sequential(
      Flatten(),
      nn.Linear(input_dim, 64),
      nn.ReLU(),
      nn.Linear(64, 10))

  def forward(self, x):
    return self.net(x)

model = TwoLayerModel().to(device)

loss = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001, weight_decay=0.01)

# Training epoch should be about 15-20 sec each on GPU.
metrics = train(model, train_loader, test_loader, loss, optimizer, training_epochs)
```

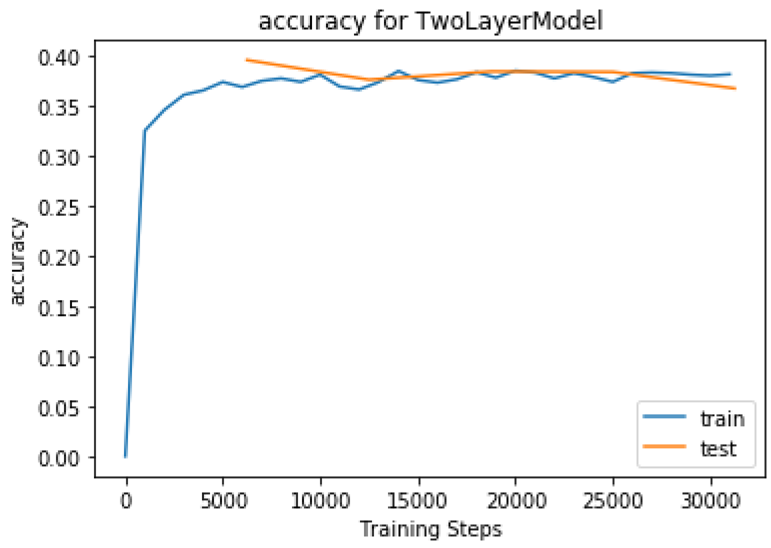
```
  Epoch 0 @ step 0: Train Loss: 0.004275, Train Accuracy: 0.000125
  Epoch 0 @ step 1000: Train Loss: 1.926468, Train Accuracy: 0.325750
  Epoch 0 @ step 2000: Train Loss: 1.820062, Train Accuracy: 0.346500
  Epoch 0 @ step 3000: Train Loss: 1.785277, Train Accuracy: 0.361625
  Epoch 0 @ step 4000: Train Loss: 1.765938, Train Accuracy: 0.366125
  Epoch 0 @ step 5000: Train Loss: 1.770911, Train Accuracy: 0.374500
  Epoch 0 @ step 6000: Train Loss: 1.751399, Train Accuracy: 0.369500
Epoch 0 Test Loss: 1.686041, Test Accuracy: 0.396500, time: 17.1s
  Epoch 1 @ step 7000: Train Loss: 1.742550, Train Accuracy: 0.375625
  Epoch 1 @ step 8000: Train Loss: 1.739604, Train Accuracy: 0.378125
  Epoch 1 @ step 9000: Train Loss: 1.744400, Train Accuracy: 0.374625
  Epoch 1 @ step 10000: Train Loss: 1.731967, Train Accuracy: 0.381875
  Epoch 1 @ step 11000: Train Loss: 1.761922, Train Accuracy: 0.370000
  Epoch 1 @ step 12000: Train Loss: 1.758104, Train Accuracy: 0.367000
Epoch 1 Test Loss: 1.719496, Test Accuracy: 0.376800, time: 17.2s
  Epoch 2 @ step 13000: Train Loss: 1.741736, Train Accuracy: 0.374375
  Epoch 2 @ step 14000: Train Loss: 1.729267, Train Accuracy: 0.385500
  Epoch 2 @ step 15000: Train Loss: 1.740460, Train Accuracy: 0.376500
  Epoch 2 @ step 16000: Train Loss: 1.739178, Train Accuracy: 0.373875
  Epoch 2 @ step 17000: Train Loss: 1.738806, Train Accuracy: 0.377250
  Epoch 2 @ step 18000: Train Loss: 1.720599, Train Accuracy: 0.384250
Epoch 2 Test Loss: 1.716003, Test Accuracy: 0.385100, time: 18.1s
  Epoch 3 @ step 19000: Train Loss: 1.745650, Train Accuracy: 0.378875
  Epoch 3 @ step 20000: Train Loss: 1.729838, Train Accuracy: 0.385875
  Epoch 3 @ step 21000: Train Loss: 1.720939, Train Accuracy: 0.384000
  Epoch 3 @ step 22000: Train Loss: 1.737750, Train Accuracy: 0.378125
  Epoch 3 @ step 23000: Train Loss: 1.727496, Train Accuracy: 0.383375
  Epoch 3 @ step 24000: Train Loss: 1.739845, Train Accuracy: 0.379500
Epoch 3 Test Loss: 1.711893, Test Accuracy: 0.384600, time: 17.1s
  Epoch 4 @ step 25000: Train Loss: 1.743235, Train Accuracy: 0.374625
  Epoch 4 @ step 26000: Train Loss: 1.718542, Train Accuracy: 0.383375
  Epoch 4 @ step 27000: Train Loss: 1.745367, Train Accuracy: 0.384000
  Epoch 4 @ step 28000: Train Loss: 1.727503, Train Accuracy: 0.383375
  Epoch 4 @ step 29000: Train Loss: 1.743228, Train Accuracy: 0.381875
  Epoch 4 @ step 30000: Train Loss: 1.734285, Train Accuracy: 0.380875
  Epoch 4 @ step 31000: Train Loss: 1.731516, Train Accuracy: 0.382250
Epoch 4 Test Loss: 1.747959, Test Accuracy: 0.368100, time: 17.1s
Final Test Loss: 1.747959, Test Accuracy: 0.368100, Total time: 86.6s
```

Plot the model results

Normally we would want to use Tensorboard for looking at metrics. However, if colab reset while we are working, we might lose our logs and therefore our metrics. Let's just plot some graphs that will survive across colab instances.

```
plot_graphs("TwoLayerModel", metrics)
```





Part 2 Convolutional Neural Network (CNN) (35 Points)

Now, let's design a convolution neural netwrok!

Build a simple CNN model, inserting 2 CNN layers in from of our 2 layer fully connect model from above:

1. A convolution with 3x3 filter, 16 output channels, stride = 1, padding=1
2. A ReLU activation
3. A Max-Pooling layer with 2x2 window
4. A convolution, 3x3 filter, 16 output channels, stride = 1, padding=1
5. A ReLU activation
6. Flatten layer

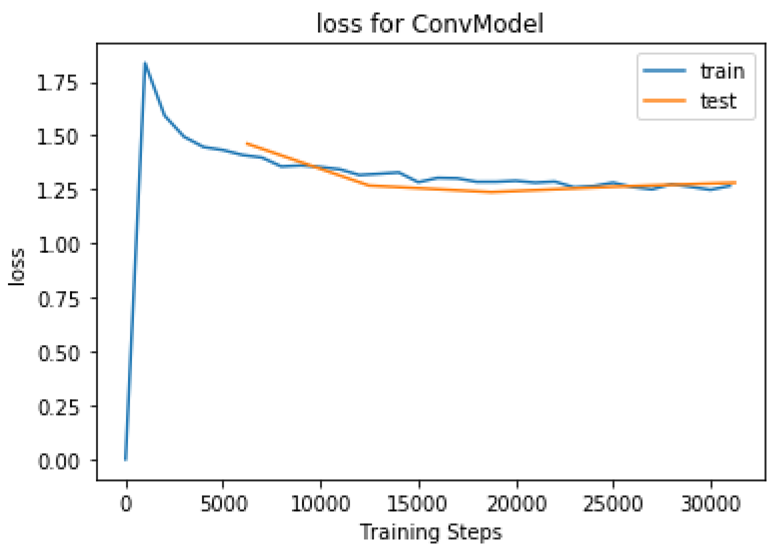
7. Fully connected linear layer with output size 64
8. ReLU
9. Fully connected linear layer, with output size 10

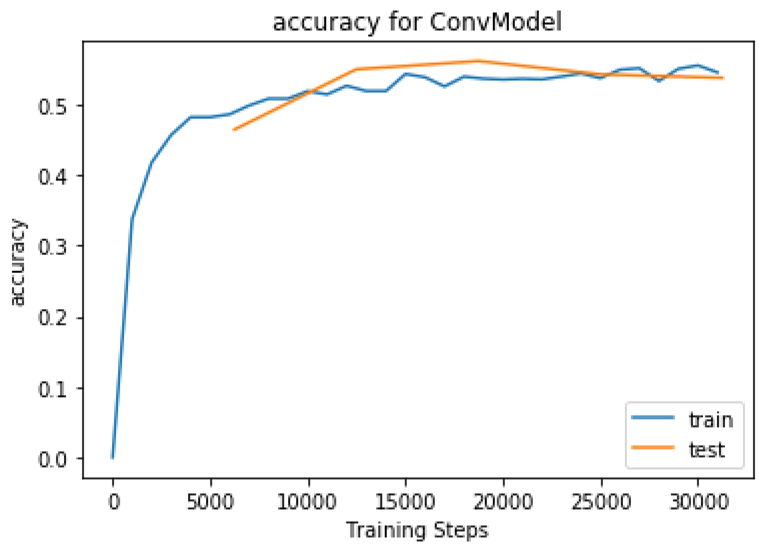
You will have to figure out the input sizes of the first fully connnected layer based on the previous layer sizes. Note that you also need to fill those in the report section (see report section in the notebook for details)

```
class ConvModel(nn.Module):
  # Your Code Here
  def __init__(self):
    super(ConvModel, self).__init__()
    self.conv = nn.Sequential(
        nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        Flatten(),
        nn.Linear(4096, 64),
        nn.ReLU(),
        nn.Linear(64, 10))

  def forward(self, x):
    return self.conv(x)

model = ConvModel().to(device)

loss = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001, weight_decay=0.01)

metrics = train(model, train_loader, test_loader, loss, optimizer, training_epochs)
```

```
    Epoch 0 @ step 0: Train Loss: 0.004307, Train Accuracy: 0.000000
    Epoch 0 @ step 1000: Train Loss: 1.835803, Train Accuracy: 0.337250
    Epoch 0 @ step 2000: Train Loss: 1.592801, Train Accuracy: 0.417500
    Epoch 0 @ step 3000: Train Loss: 1.494804, Train Accuracy: 0.456625
    Epoch 0 @ step 4000: Train Loss: 1.447257, Train Accuracy: 0.481875
    Epoch 0 @ step 5000: Train Loss: 1.433165, Train Accuracy: 0.482000
    Epoch 0 @ step 6000: Train Loss: 1.410358, Train Accuracy: 0.486000
  Epoch 0 Test Loss: 1.462240, Test Accuracy: 0.464400, time: 24.5s
    Epoch 1 @ step 7000: Train Loss: 1.398430, Train Accuracy: 0.498500
    Epoch 1 @ step 8000: Train Loss: 1.357032, Train Accuracy: 0.508250
    Epoch 1 @ step 9000: Train Loss: 1.361926, Train Accuracy: 0.508250
    Epoch 1 @ step 10000: Train Loss: 1.354716, Train Accuracy: 0.518875
```

```
plot_graphs("ConvModel", metrics)
```





Do you notice the improvement over the accuracy compared to that in Part 1?

Yes.

Part 3 Open Design Competition (35 Points + 10 bonus points)

Try to beat the previous models by adding additional layers, changing parameters, etc. You should add at least one layer.

Possible changes include:

- Dropout
- Batch Normalization
- More layers
- Residual Connections (harder)
- Change layer size
- Pooling layers, stride
- Different optimizer
- Train for longer

Once you have a model you think is great, evaluate it against our hidden test data (see hidden_loader above) and upload the results to the leader board on gradescope. **The top 3 scorers will get a bonus 10 points.**

You can steal model structures found on the internet if you want. The only constraint is that **you must train the model from scratch**.

```
from torch.utils.data import DataLoader

batch_size = 8
train_loader = torch.utils.data.DataLoader(train_set, batch_size, shuffle=True, num_workers=
test_loader = torch.utils.data.DataLoader(test_set, batch_size, shuffle=True, num_workers=2)

input_shape = np.array(train_set[0][0]).shape
input_dim = input_shape[1]*input_shape[2]*input_shape[0]
```

```
# You Awesome Super Best model code here
class AwesomeModel(nn.Module):
  # Your Code Here
  def __init__(self):
    super(AwesomeModel, self).__init__()
    self.conv = nn.Sequential(
        nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=32),
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=32),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        nn.Dropout(p=0.2),

        nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=64),
        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=64),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        nn.Dropout(p=0.3),

        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=64),
        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.BatchNorm2d(num_features=64),
        nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
        nn.Dropout(p=0.4),

        Flatten(),
        nn.Linear(1024, 100),
        nn.ReLU(),
        nn.Linear(100, 10))

  def forward(self, x):
    return self.conv(x)


## Load Saved Model
# PATH = 'drive/My Drive/CS 498 AML/HW10/model2_checkpoint_3Conv2D_3.pth.tar'
# model = AwesomeModel()
# model.load_state_dict(torch.load(PATH, map_location="cuda:0"))
# model.to(device)

# Initialize Model
model = AwesomeModel().to(device)

# Training
loss = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
metrics = train(model, train_loader, test_loader, loss, optimizer, 60)
# torch.save(model.state_dict(), 'drive/My Drive/CS 498 AML/HW10/model2_checkpoint_3Conv2D_5
  Epoch 0 @ step 0: Train Loss: 0.004334, Train Accuracy: 0.000125
  Epoch 0 @ step 1000: Train Loss: 1.713327, Train Accuracy: 0.369750
  Epoch 0 @ step 2000: Train Loss: 1.406628, Train Accuracy: 0.498375
  Epoch 0 @ step 3000: Train Loss: 1.259663, Train Accuracy: 0.546500
  Epoch 0 @ step 4000: Train Loss: 1.168738, Train Accuracy: 0.583750
  Epoch 0 @ step 5000: Train Loss: 1.091209, Train Accuracy: 0.613125
  Epoch 0 @ step 6000: Train Loss: 1.026775, Train Accuracy: 0.633375
Epoch 0 Test Loss: 0.899229, Test Accuracy: 0.685400, time: 53.2s
  Epoch 1 @ step 7000: Train Loss: 0.984515, Train Accuracy: 0.655375
  Epoch 1 @ step 8000: Train Loss: 0.958142, Train Accuracy: 0.662750
  Epoch 1 @ step 9000: Train Loss: 0.917372, Train Accuracy: 0.680375
  Epoch 1 @ step 10000: Train Loss: 0.899818, Train Accuracy: 0.689375
  Epoch 1 @ step 11000: Train Loss: 0.860245, Train Accuracy: 0.699250
  Epoch 1 @ step 12000: Train Loss: 0.861270, Train Accuracy: 0.700250
Epoch 1 Test Loss: 0.774932, Test Accuracy: 0.731500, time: 53.2s
  Epoch 2 @ step 13000: Train Loss: 0.834324, Train Accuracy: 0.710250
  Epoch 2 @ step 14000: Train Loss: 0.784732, Train Accuracy: 0.719875
  Epoch 2 @ step 15000: Train Loss: 0.797737, Train Accuracy: 0.717375
  Epoch 2 @ step 16000: Train Loss: 0.792894, Train Accuracy: 0.725000
  Epoch 2 @ step 17000: Train Loss: 0.794747, Train Accuracy: 0.722750
  Epoch 2 @ step 18000: Train Loss: 0.753367, Train Accuracy: 0.738250
Epoch 2 Test Loss: 0.661410, Test Accuracy: 0.770600, time: 54.3s
  Epoch 3 @ step 19000: Train Loss: 0.754940, Train Accuracy: 0.735750
  Epoch 3 @ step 20000: Train Loss: 0.694611, Train Accuracy: 0.762000
  Epoch 3 @ step 21000: Train Loss: 0.716741, Train Accuracy: 0.745000
  Epoch 3 @ step 22000: Train Loss: 0.735537, Train Accuracy: 0.742125
  Epoch 3 @ step 23000: Train Loss: 0.696336, Train Accuracy: 0.760125
  Epoch 3 @ step 24000: Train Loss: 0.709621, Train Accuracy: 0.755000
Epoch 3 Test Loss: 0.635224, Test Accuracy: 0.781400, time: 54.8s
  Epoch 4 @ step 25000: Train Loss: 0.679930, Train Accuracy: 0.759000
  Epoch 4 @ step 26000: Train Loss: 0.649383, Train Accuracy: 0.775000
  Epoch 4 @ step 27000: Train Loss: 0.658962, Train Accuracy: 0.771625
  Epoch 4 @ step 28000: Train Loss: 0.644270, Train Accuracy: 0.775125
  Epoch 4 @ step 29000: Train Loss: 0.646067, Train Accuracy: 0.770250
  Epoch 4 @ step 30000: Train Loss: 0.672734, Train Accuracy: 0.769625
  Epoch 4 @ step 31000: Train Loss: 0.637147, Train Accuracy: 0.777625
Epoch 4 Test Loss: 0.583313, Test Accuracy: 0.798700, time: 56.8s
  Epoch 5 @ step 32000: Train Loss: 0.609356, Train Accuracy: 0.783500
  Epoch 5 @ step 33000: Train Loss: 0.608653, Train Accuracy: 0.789375
  Epoch 5 @ step 34000: Train Loss: 0.610808, Train Accuracy: 0.787000
  Epoch 5 @ step 35000: Train Loss: 0.607500, Train Accuracy: 0.785750
  Epoch 5 @ step 36000: Train Loss: 0.608746, Train Accuracy: 0.788625
  Epoch 5 @ step 37000: Train Loss: 0.610913, Train Accuracy: 0.787875
Epoch 5 Test Loss: 0.558663, Test Accuracy: 0.807300, time: 55.0s
  Epoch 6 @ step 38000: Train Loss: 0.577606, Train Accuracy: 0.807000
  Epoch 6 @ step 39000: Train Loss: 0.574980, Train Accuracy: 0.802000
  Epoch 6 @ step 40000: Train Loss: 0.563049, Train Accuracy: 0.804875
  Epoch 6 @ step 41000: Train Loss: 0.569357, Train Accuracy: 0.809375
  Epoch 6 @ step 42000: Train Loss: 0.589778, Train Accuracy: 0.795500
  Epoch 6 @ step 43000: Train Loss: 0.585804, Train Accuracy: 0.798375
Epoch 6 Test Loss: 0.554358, Test Accuracy: 0.815800, time: 55.0s
  Epoch 7 @ step 44000: Train Loss: 0.574155, Train Accuracy: 0.801500
  Epoch 7 @ step 45000: Train Loss: 0.547207, Train Accuracy: 0.811000
  Epoch 7 @ step 46000: Train Loss: 0.538718, Train Accuracy: 0.812625
  Epoch 7 @ step 47000: Train Loss: 0.544380, Train Accuracy: 0.812250
  Epoch 7 @ step 48000: Train Loss: 0.538671, Train Accuracy: 0.810625
  Epoch 7 @ step 49000: Train Loss: 0.551511, Train Accuracy: 0.811375
Epoch 7 Test Loss: 0.542635, Test Accuracy: 0.816800, time: 54.7s
  Epoch 8 @ step 50000: Train Loss: 0.544221, Train Accuracy: 0.811750
  Epoch 8 @ step 51000: Train Loss: 0.499475, Train Accuracy: 0.820875
  Epoch 8 @ step 52000: Train Loss: 0.503819, Train Accuracy: 0.824875
  Epoch 8 @ step 53000: Train Loss: 0.543293, Train Accuracy: 0.810750
  Epoch 8 @ step 54000: Train Loss: 0.532195, Train Accuracy: 0.813500
  Epoch 8 @ step 55000: Train Loss: 0.530132, Train Accuracy: 0.812000
  Epoch 8 @ step 56000: Train Loss: 0.535771, Train Accuracy: 0.811500
Epoch 8 Test Loss: 0.532264, Test Accuracy: 0.824000, time: 53.7s
  Epoch 9 @ step 57000: Train Loss: 0.485003, Train Accuracy: 0.832250
  Epoch 9 @ step 58000: Train Loss: 0.501837, Train Accuracy: 0.823875
  Epoch 9 @ step 59000: Train Loss: 0.513059, Train Accuracy: 0.824125
  Epoch 9 @ step 60000: Train Loss: 0.510197, Train Accuracy: 0.823500
  Epoch 9 @ step 61000: Train Loss: 0.502278, Train Accuracy: 0.825750
  Epoch 9 @ step 62000: Train Loss: 0.511031, Train Accuracy: 0.819875
Epoch 9 Test Loss: 0.525686, Test Accuracy: 0.824200, time: 56.0s
  Epoch 10 @ step 63000: Train Loss: 0.490878, Train Accuracy: 0.830000
  Epoch 10 @ step 64000: Train Loss: 0.462843, Train Accuracy: 0.838875
  Epoch 10 @ step 65000: Train Loss: 0.465455, Train Accuracy: 0.839750
  Epoch 10 @ step 66000: Train Loss: 0.471662, Train Accuracy: 0.834500
  Epoch 10 @ step 67000: Train Loss: 0.484863, Train Accuracy: 0.828000
  Epoch 10 @ step 68000: Train Loss: 0.494808, Train Accuracy: 0.830125
Epoch 10 Test Loss: 0.506358, Test Accuracy: 0.831500, time: 54.3s
  Epoch 11 @ step 69000: Train Loss: 0.488225, Train Accuracy: 0.827375
  Epoch 11 @ step 70000: Train Loss: 0.458943, Train Accuracy: 0.841875
  Epoch 11 @ step 71000: Train Loss: 0.454306, Train Accuracy: 0.843125
  Epoch 11 @ step 72000: Train Loss: 0.459643, Train Accuracy: 0.842000
  Epoch 11 @ step 73000: Train Loss: 0.497179, Train Accuracy: 0.827125
  Epoch 11 @ step 74000: Train Loss: 0.478431, Train Accuracy: 0.831750
Epoch 11 Test Loss: 0.508246, Test Accuracy: 0.829500, time: 53.1s
  Epoch 12 @ step 75000: Train Loss: 0.467120, Train Accuracy: 0.838125
  Epoch 12 @ step 76000: Train Loss: 0.434584, Train Accuracy: 0.847125
  Epoch 12 @ step 77000: Train Loss: 0.446001, Train Accuracy: 0.845500
  Epoch 12 @ step 78000: Train Loss: 0.453545, Train Accuracy: 0.837250
  Epoch 12 @ step 79000: Train Loss: 0.464854, Train Accuracy: 0.839625
  Epoch 12 @ step 80000: Train Loss: 0.475968, Train Accuracy: 0.828500
  Epoch 12 @ step 81000: Train Loss: 0.469164, Train Accuracy: 0.836875
Epoch 12 Test Loss: 0.501376, Test Accuracy: 0.834800, time: 55.4s
  Epoch 13 @ step 82000: Train Loss: 0.425438, Train Accuracy: 0.848000
  Epoch 13 @ step 83000: Train Loss: 0.426051, Train Accuracy: 0.851500
  Epoch 13 @ step 84000: Train Loss: 0.447021, Train Accuracy: 0.842750
  Epoch 13 @ step 85000: Train Loss: 0.443222, Train Accuracy: 0.843500
  Epoch 13 @ step 86000: Train Loss: 0.456045, Train Accuracy: 0.841125
  Epoch 13 @ step 87000: Train Loss: 0.443751, Train Accuracy: 0.845625
Epoch 13 Test Loss: 0.495647, Test Accuracy: 0.834400, time: 55.6s
  Epoch 14 @ step 88000: Train Loss: 0.452720, Train Accuracy: 0.843750
  Epoch 14 @ step 89000: Train Loss: 0.420991, Train Accuracy: 0.850750
  Epoch 14 @ step 90000: Train Loss: 0.427785, Train Accuracy: 0.846875
  Epoch 14 @ step 91000: Train Loss: 0.426401, Train Accuracy: 0.849375
  Epoch 14 @ step 92000: Train Loss: 0.433093, Train Accuracy: 0.846250
  Epoch 14 @ step 93000: Train Loss: 0.442122, Train Accuracy: 0.847000
Epoch 14 Test Loss: 0.474697, Test Accuracy: 0.842400, time: 56.2s
  Epoch 15 @ step 94000: Train Loss: 0.427886, Train Accuracy: 0.848375
  Epoch 15 @ step 95000: Train Loss: 0.405291, Train Accuracy: 0.860500
  Epoch 15 @ step 96000: Train Loss: 0.414968, Train Accuracy: 0.855125
  Epoch 15 @ step 97000: Train Loss: 0.445331, Train Accuracy: 0.841000
  Epoch 15 @ step 98000: Train Loss: 0.425737, Train Accuracy: 0.856250
  Epoch 15 @ step 99000: Train Loss: 0.419690, Train Accuracy: 0.853250
Epoch 15 Test Loss: 0.487122, Test Accuracy: 0.840300, time: 53.8s
  Epoch 16 @ step 100000: Train Loss: 0.431039, Train Accuracy: 0.850125
  Epoch 16 @ step 101000: Train Loss: 0.403503, Train Accuracy: 0.856000
  Epoch 16 @ step 102000: Train Loss: 0.408011, Train Accuracy: 0.861125
  Epoch 16 @ step 103000: Train Loss: 0.393327, Train Accuracy: 0.862875
  Epoch 16 @ step 104000: Train Loss: 0.433078, Train Accuracy: 0.844500
  Epoch 16 @ step 105000: Train Loss: 0.421070, Train Accuracy: 0.853375
  Epoch 16 @ step 106000: Train Loss: 0.428317, Train Accuracy: 0.850250
Epoch 16 Test Loss: 0.487058, Test Accuracy: 0.840500, time: 54.9s
  Epoch 17 @ step 107000: Train Loss: 0.379871, Train Accuracy: 0.867000
  Epoch 17 @ step 108000: Train Loss: 0.392878, Train Accuracy: 0.861750
  Epoch 17 @ step 109000: Train Loss: 0.390035, Train Accuracy: 0.859875
  Epoch 17 @ step 110000: Train Loss: 0.407702, Train Accuracy: 0.858000
  Epoch 17 @ step 111000: Train Loss: 0.422684, Train Accuracy: 0.849250
  Epoch 17 @ step 112000: Train Loss: 0.422359, Train Accuracy: 0.854250
Epoch 17 Test Loss: 0.481470, Test Accuracy: 0.841200, time: 55.1s
  Epoch 18 @ step 113000: Train Loss: 0.403070, Train Accuracy: 0.855125
  Epoch 18 @ step 114000: Train Loss: 0.388888, Train Accuracy: 0.864875
  Epoch 18 @ step 115000: Train Loss: 0.377959, Train Accuracy: 0.867625
  Epoch 18 @ step 116000: Train Loss: 0.389065, Train Accuracy: 0.864500
  Epoch 18 @ step 117000: Train Loss: 0.405707, Train Accuracy: 0.860375
  Epoch 18 @ step 118000: Train Loss: 0.392889, Train Accuracy: 0.864625
Epoch 18 Test Loss: 0.470457, Test Accuracy: 0.845000, time: 54.9s
  Epoch 19 @ step 119000: Train Loss: 0.397001, Train Accuracy: 0.861875
  Epoch 19 @ step 120000: Train Loss: 0.381155, Train Accuracy: 0.866500
  Epoch 19 @ step 121000: Train Loss: 0.388859, Train Accuracy: 0.861750
  Epoch 19 @ step 122000: Train Loss: 0.390138, Train Accuracy: 0.863000
  Epoch 19 @ step 123000: Train Loss: 0.386216, Train Accuracy: 0.863625
  Epoch 19 @ step 124000: Train Loss: 0.393735, Train Accuracy: 0.861500
Epoch 19 Test Loss: 0.494135, Test Accuracy: 0.839700, time: 55.6s
  Epoch 20 @ step 125000: Train Loss: 0.407460, Train Accuracy: 0.856750
  Epoch 20 @ step 126000: Train Loss: 0.362731, Train Accuracy: 0.875125
  Epoch 20 @ step 127000: Train Loss: 0.361697, Train Accuracy: 0.873375
  Epoch 20 @ step 128000: Train Loss: 0.382774, Train Accuracy: 0.863250
  Epoch 20 @ step 129000: Train Loss: 0.398023, Train Accuracy: 0.861000
  Epoch 20 @ step 130000: Train Loss: 0.388216, Train Accuracy: 0.865500
  Epoch 20 @ step 131000: Train Loss: 0.376252, Train Accuracy: 0.869625
Epoch 20 Test Loss: 0.493843, Test Accuracy: 0.846100, time: 54.2s
  Epoch 21 @ step 132000: Train Loss: 0.362002, Train Accuracy: 0.873000
  Epoch 21 @ step 133000: Train Loss: 0.350869, Train Accuracy: 0.877000
  Epoch 21 @ step 134000: Train Loss: 0.355748, Train Accuracy: 0.874375
  Epoch 21 @ step 135000: Train Loss: 0.381720, Train Accuracy: 0.863250
  Epoch 21 @ step 136000: Train Loss: 0.397291, Train Accuracy: 0.861375
  Epoch 21 @ step 137000: Train Loss: 0.381972, Train Accuracy: 0.862875
Epoch 21 Test Loss: 0.488169, Test Accuracy: 0.843800, time: 54.9s
  Epoch 22 @ step 138000: Train Loss: 0.378047, Train Accuracy: 0.870625
  Epoch 22 @ step 139000: Train Loss: 0.344177, Train Accuracy: 0.877625
  Epoch 22 @ step 140000: Train Loss: 0.369695, Train Accuracy: 0.868000
  Epoch 22 @ step 141000: Train Loss: 0.368723, Train Accuracy: 0.874375
  Epoch 22 @ step 142000: Train Loss: 0.386172, Train Accuracy: 0.866125
  Epoch 22 @ step 143000: Train Loss: 0.380448, Train Accuracy: 0.865875
Epoch 22 Test Loss: 0.496585, Test Accuracy: 0.839000, time: 54.9s
  Epoch 23 @ step 144000: Train Loss: 0.370740, Train Accuracy: 0.870000
  Epoch 23 @ step 145000: Train Loss: 0.343578, Train Accuracy: 0.880375
  Epoch 23 @ step 146000: Train Loss: 0.364496, Train Accuracy: 0.871375
  Epoch 23 @ step 147000: Train Loss: 0.359572, Train Accuracy: 0.872750
  Epoch 23 @ step 148000: Train Loss: 0.365739, Train Accuracy: 0.872125
  Epoch 23 @ step 149000: Train Loss: 0.362566, Train Accuracy: 0.873750
```

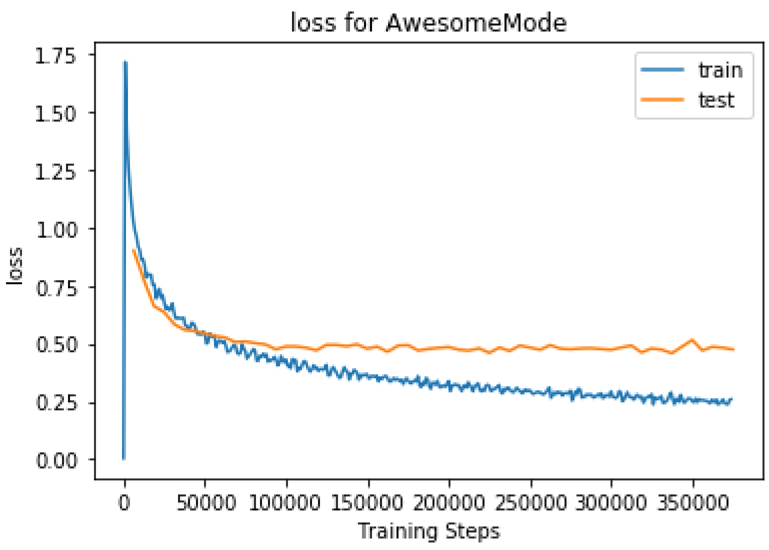
```
    Epoch 23 @ step 149000: Train Loss: 0.362566, Train Accuracy: 0.873750
  Epoch 23 Test Loss: 0.477903, Test Accuracy: 0.843300, time: 55.0s
    Epoch 24 @ step 150000: Train Loss: 0.367852, Train Accuracy: 0.871750
    Epoch 24 @ step 151000: Train Loss: 0.349545, Train Accuracy: 0.879500
    Epoch 24 @ step 152000: Train Loss: 0.351709, Train Accuracy: 0.881375
    Epoch 24 @ step 153000: Train Loss: 0.359961, Train Accuracy: 0.877500
    Epoch 24 @ step 154000: Train Loss: 0.357802, Train Accuracy: 0.870250
    Epoch 24 @ step 155000: Train Loss: 0.355754, Train Accuracy: 0.872750
    Epoch 24 @ step 156000: Train Loss: 0.361233, Train Accuracy: 0.873250
  Epoch 24 Test Loss: 0.486259, Test Accuracy: 0.849000, time: 54.8s
    Epoch 25 @ step 157000: Train Loss: 0.336491, Train Accuracy: 0.881250
    Epoch 25 @ step 158000: Train Loss: 0.351254, Train Accuracy: 0.876000
    Epoch 25 @ step 159000: Train Loss: 0.346393, Train Accuracy: 0.878750
    Epoch 25 @ step 160000: Train Loss: 0.354028, Train Accuracy: 0.875875
    Epoch 25 @ step 161000: Train Loss: 0.359429, Train Accuracy: 0.876000
    Epoch 25 @ step 162000: Train Loss: 0.357729, Train Accuracy: 0.875375
  Epoch 25 Test Loss: 0.464316, Test Accuracy: 0.851200, time: 55.4s
    Epoch 26 @ step 163000: Train Loss: 0.342760, Train Accuracy: 0.878250
    Epoch 26 @ step 164000: Train Loss: 0.334985, Train Accuracy: 0.883125
    Epoch 26 @ step 165000: Train Loss: 0.341234, Train Accuracy: 0.876500
    Epoch 26 @ step 166000: Train Loss: 0.350258, Train Accuracy: 0.876250
    Epoch 26 @ step 167000: Train Loss: 0.342441, Train Accuracy: 0.878125
    Epoch 26 @ step 168000: Train Loss: 0.347664, Train Accuracy: 0.879000
  Epoch 26 Test Loss: 0.490856, Test Accuracy: 0.849000, time: 55.3s
    Epoch 27 @ step 169000: Train Loss: 0.344107, Train Accuracy: 0.876000
    Epoch 27 @ step 170000: Train Loss: 0.336869, Train Accuracy: 0.881250
    Epoch 27 @ step 171000: Train Loss: 0.343200, Train Accuracy: 0.879375
    Epoch 27 @ step 172000: Train Loss: 0.324869, Train Accuracy: 0.886375
    Epoch 27 @ step 173000: Train Loss: 0.340365, Train Accuracy: 0.879875
    Epoch 27 @ step 174000: Train Loss: 0.350231, Train Accuracy: 0.876875
  Epoch 27 Test Loss: 0.492623, Test Accuracy: 0.843200, time: 55.8s
    Epoch 28 @ step 175000: Train Loss: 0.338920, Train Accuracy: 0.878875
    Epoch 28 @ step 176000: Train Loss: 0.330427, Train Accuracy: 0.881250
    Epoch 28 @ step 177000: Train Loss: 0.327353, Train Accuracy: 0.888500
    Epoch 28 @ step 178000: Train Loss: 0.337003, Train Accuracy: 0.882500
    Epoch 28 @ step 179000: Train Loss: 0.338647, Train Accuracy: 0.881250
    Epoch 28 @ step 180000: Train Loss: 0.336361, Train Accuracy: 0.879625
    Epoch 28 @ step 181000: Train Loss: 0.358618, Train Accuracy: 0.874125
  Epoch 28 Test Loss: 0.469586, Test Accuracy: 0.850900, time: 54.0s
    Epoch 29 @ step 182000: Train Loss: 0.320330, Train Accuracy: 0.887750
    Epoch 29 @ step 183000: Train Loss: 0.311581, Train Accuracy: 0.889875
    Epoch 29 @ step 184000: Train Loss: 0.331039, Train Accuracy: 0.882500
    Epoch 29 @ step 185000: Train Loss: 0.323084, Train Accuracy: 0.885000
    Epoch 29 @ step 186000: Train Loss: 0.350673, Train Accuracy: 0.874625
    Epoch 29 @ step 187000: Train Loss: 0.345256, Train Accuracy: 0.878000
  Epoch 29 Test Loss: 0.476527, Test Accuracy: 0.850700, time: 54.4s
    Epoch 30 @ step 188000: Train Loss: 0.316353, Train Accuracy: 0.886250
    Epoch 30 @ step 189000: Train Loss: 0.304603, Train Accuracy: 0.893000
    Epoch 30 @ step 190000: Train Loss: 0.331490, Train Accuracy: 0.886500
    Epoch 30 @ step 191000: Train Loss: 0.326794, Train Accuracy: 0.887625
    Epoch 30 @ step 192000: Train Loss: 0.322893, Train Accuracy: 0.880875
    Epoch 30 @ step 193000: Train Loss: 0.338039, Train Accuracy: 0.885250
  Epoch 30 Test Loss: 0.481072, Test Accuracy: 0.850500, time: 55.7s
    Epoch 31 @ step 194000: Train Loss: 0.316768, Train Accuracy: 0.886750
    Epoch 31 @ step 195000: Train Loss: 0.310021, Train Accuracy: 0.890875
    Epoch 31 @ step 196000: Train Loss: 0.317788, Train Accuracy: 0.885500
    Epoch 31 @ step 197000: Train Loss: 0.317042, Train Accuracy: 0.888875
    Epoch 31 @ step 198000: Train Loss: 0.313734, Train Accuracy: 0.890000
    Epoch 31 @ step 199000: Train Loss: 0.327570, Train Accuracy: 0.880875
  Epoch 31 Test Loss: 0.484593, Test Accuracy: 0.848300, time: 54.9s
    Epoch 32 @ step 200000: Train Loss: 0.337217, Train Accuracy: 0.883625
    Epoch 32 @ step 201000: Train Loss: 0.294987, Train Accuracy: 0.895375
    Epoch 32 @ step 202000: Train Loss: 0.318131, Train Accuracy: 0.887125
    Epoch 32 @ step 203000: Train Loss: 0.319510, Train Accuracy: 0.888125
    Epoch 32 @ step 204000: Train Loss: 0.323540, Train Accuracy: 0.887250
    Epoch 32 @ step 205000: Train Loss: 0.318758, Train Accuracy: 0.889000
    Epoch 32 @ step 206000: Train Loss: 0.320881, Train Accuracy: 0.887250
  Epoch 32 Test Loss: 0.472997, Test Accuracy: 0.851100, time: 54.8s
    Epoch 33 @ step 207000: Train Loss: 0.306895, Train Accuracy: 0.892875
    Epoch 33 @ step 208000: Train Loss: 0.294207, Train Accuracy: 0.895250
    Epoch 33 @ step 209000: Train Loss: 0.315471, Train Accuracy: 0.888750
    Epoch 33 @ step 210000: Train Loss: 0.319666, Train Accuracy: 0.887750
    Epoch 33 @ step 211000: Train Loss: 0.309490, Train Accuracy: 0.888000
    Epoch 33 @ step 212000: Train Loss: 0.332121, Train Accuracy: 0.886250
  Epoch 33 Test Loss: 0.469470, Test Accuracy: 0.853200, time: 54.4s
    Epoch 34 @ step 213000: Train Loss: 0.317870, Train Accuracy: 0.891000
    Epoch 34 @ step 214000: Train Loss: 0.311049, Train Accuracy: 0.892500
    Epoch 34 @ step 215000: Train Loss: 0.293312, Train Accuracy: 0.894875
    Epoch 34 @ step 216000: Train Loss: 0.317206, Train Accuracy: 0.891750
    Epoch 34 @ step 217000: Train Loss: 0.319374, Train Accuracy: 0.888000
    Epoch 34 @ step 218000: Train Loss: 0.328124, Train Accuracy: 0.882375
  Epoch 34 Test Loss: 0.479017, Test Accuracy: 0.851000, time: 54.7s
    Epoch 35 @ step 219000: Train Loss: 0.302521, Train Accuracy: 0.890000
    Epoch 35 @ step 220000: Train Loss: 0.292856, Train Accuracy: 0.893750
    Epoch 35 @ step 221000: Train Loss: 0.303815, Train Accuracy: 0.894250
    Epoch 35 @ step 222000: Train Loss: 0.295076, Train Accuracy: 0.897250
    Epoch 35 @ step 223000: Train Loss: 0.317962, Train Accuracy: 0.883750
    Epoch 35 @ step 224000: Train Loss: 0.321266, Train Accuracy: 0.887000
  Epoch 35 Test Loss: 0.459516, Test Accuracy: 0.855800, time: 56.8s
    Epoch 36 @ step 225000: Train Loss: 0.315433, Train Accuracy: 0.889500
    Epoch 36 @ step 226000: Train Loss: 0.293715, Train Accuracy: 0.899500
    Epoch 36 @ step 227000: Train Loss: 0.301728, Train Accuracy: 0.896125
    Epoch 36 @ step 228000: Train Loss: 0.301288, Train Accuracy: 0.893250
    Epoch 36 @ step 229000: Train Loss: 0.309587, Train Accuracy: 0.889750
    Epoch 36 @ step 230000: Train Loss: 0.310866, Train Accuracy: 0.890125
    Epoch 36 @ step 231000: Train Loss: 0.320202, Train Accuracy: 0.888000
  Epoch 36 Test Loss: 0.482444, Test Accuracy: 0.853000, time: 54.3s
    Epoch 37 @ step 232000: Train Loss: 0.292192, Train Accuracy: 0.896375
    Epoch 37 @ step 233000: Train Loss: 0.283908, Train Accuracy: 0.900125
    Epoch 37 @ step 234000: Train Loss: 0.303858, Train Accuracy: 0.893125
    Epoch 37 @ step 235000: Train Loss: 0.297409, Train Accuracy: 0.896125
    Epoch 37 @ step 236000: Train Loss: 0.309124, Train Accuracy: 0.889500
    Epoch 37 @ step 237000: Train Loss: 0.307495, Train Accuracy: 0.891500
  Epoch 37 Test Loss: 0.467832, Test Accuracy: 0.856100, time: 55.6s
    Epoch 38 @ step 238000: Train Loss: 0.297125, Train Accuracy: 0.896125
    Epoch 38 @ step 239000: Train Loss: 0.291415, Train Accuracy: 0.894625
    Epoch 38 @ step 240000: Train Loss: 0.289322, Train Accuracy: 0.898500
    Epoch 38 @ step 241000: Train Loss: 0.300978, Train Accuracy: 0.890875
    Epoch 38 @ step 242000: Train Loss: 0.308603, Train Accuracy: 0.889250
    Epoch 38 @ step 243000: Train Loss: 0.299057, Train Accuracy: 0.894375
  Epoch 38 Test Loss: 0.491137, Test Accuracy: 0.852000, time: 55.3s
    Epoch 39 @ step 244000: Train Loss: 0.295273, Train Accuracy: 0.898250
    Epoch 39 @ step 245000: Train Loss: 0.284956, Train Accuracy: 0.901125
    Epoch 39 @ step 246000: Train Loss: 0.288418, Train Accuracy: 0.899625
    Epoch 39 @ step 247000: Train Loss: 0.301888, Train Accuracy: 0.894875
    Epoch 39 @ step 248000: Train Loss: 0.293935, Train Accuracy: 0.897375
    Epoch 39 @ step 249000: Train Loss: 0.296863, Train Accuracy: 0.895625
  Epoch 39 Test Loss: 0.482436, Test Accuracy: 0.855800, time: 55.1s
    Epoch 40 @ step 250000: Train Loss: 0.297397, Train Accuracy: 0.894625
    Epoch 40 @ step 251000: Train Loss: 0.287969, Train Accuracy: 0.900000
    Epoch 40 @ step 252000: Train Loss: 0.281901, Train Accuracy: 0.903000
    Epoch 40 @ step 253000: Train Loss: 0.285818, Train Accuracy: 0.898375
    Epoch 40 @ step 254000: Train Loss: 0.288951, Train Accuracy: 0.899750
    Epoch 40 @ step 255000: Train Loss: 0.291944, Train Accuracy: 0.897375
    Epoch 40 @ step 256000: Train Loss: 0.287039, Train Accuracy: 0.900000
  Epoch 40 Test Loss: 0.472843, Test Accuracy: 0.856500, time: 55.5s
    Epoch 41 @ step 257000: Train Loss: 0.287286, Train Accuracy: 0.898750
    Epoch 41 @ step 258000: Train Loss: 0.279866, Train Accuracy: 0.899750
    Epoch 41 @ step 259000: Train Loss: 0.282702, Train Accuracy: 0.900875
    Epoch 41 @ step 260000: Train Loss: 0.289842, Train Accuracy: 0.900875
    Epoch 41 @ step 261000: Train Loss: 0.301200, Train Accuracy: 0.892625
    Epoch 41 @ step 262000: Train Loss: 0.309610, Train Accuracy: 0.887875
  Epoch 41 Test Loss: 0.492876, Test Accuracy: 0.856400, time: 53.5s
    Epoch 42 @ step 263000: Train Loss: 0.265596, Train Accuracy: 0.906000
    Epoch 42 @ step 264000: Train Loss: 0.276816, Train Accuracy: 0.898125
    Epoch 42 @ step 265000: Train Loss: 0.288711, Train Accuracy: 0.899500
    Epoch 42 @ step 266000: Train Loss: 0.281175, Train Accuracy: 0.899500
    Epoch 42 @ step 267000: Train Loss: 0.291046, Train Accuracy: 0.898625
    Epoch 42 @ step 268000: Train Loss: 0.286734, Train Accuracy: 0.900000
  Epoch 42 Test Loss: 0.477845, Test Accuracy: 0.858200, time: 55.4s
    Epoch 43 @ step 269000: Train Loss: 0.288991, Train Accuracy: 0.897625
    Epoch 43 @ step 270000: Train Loss: 0.274468, Train Accuracy: 0.905500
    Epoch 43 @ step 271000: Train Loss: 0.284812, Train Accuracy: 0.897875
    Epoch 43 @ step 272000: Train Loss: 0.285915, Train Accuracy: 0.898375
    Epoch 43 @ step 273000: Train Loss: 0.283111, Train Accuracy: 0.899625
    Epoch 43 @ step 274000: Train Loss: 0.288690, Train Accuracy: 0.899625
  Epoch 43 Test Loss: 0.474904, Test Accuracy: 0.850000, time: 54.8s
    Epoch 44 @ step 275000: Train Loss: 0.295955, Train Accuracy: 0.896375
    Epoch 44 @ step 276000: Train Loss: 0.253586, Train Accuracy: 0.911000
    Epoch 44 @ step 277000: Train Loss: 0.270240, Train Accuracy: 0.906500
    Epoch 44 @ step 278000: Train Loss: 0.292690, Train Accuracy: 0.896125
    Epoch 44 @ step 279000: Train Loss: 0.281919, Train Accuracy: 0.901250
    Epoch 44 @ step 280000: Train Loss: 0.305267, Train Accuracy: 0.892500
    Epoch 44 @ step 281000: Train Loss: 0.299543, Train Accuracy: 0.896000
  Epoch 44 Test Loss: 0.478482, Test Accuracy: 0.856200, time: 53.8s
    Epoch 45 @ step 282000: Train Loss: 0.266841, Train Accuracy: 0.904250
    Epoch 45 @ step 283000: Train Loss: 0.268757, Train Accuracy: 0.904625
    Epoch 45 @ step 284000: Train Loss: 0.271928, Train Accuracy: 0.906000
    Epoch 45 @ step 285000: Train Loss: 0.280467, Train Accuracy: 0.903500
    Epoch 45 @ step 286000: Train Loss: 0.275330, Train Accuracy: 0.904875
    Epoch 45 @ step 287000: Train Loss: 0.279846, Train Accuracy: 0.899375
  Epoch 45 Test Loss: 0.479540, Test Accuracy: 0.855300, time: 55.9s
    Epoch 46 @ step 288000: Train Loss: 0.277691, Train Accuracy: 0.904250
    Epoch 46 @ step 289000: Train Loss: 0.267158, Train Accuracy: 0.903875
    Epoch 46 @ step 290000: Train Loss: 0.268854, Train Accuracy: 0.907875
    Epoch 46 @ step 291000: Train Loss: 0.274175, Train Accuracy: 0.902250
    Epoch 46 @ step 292000: Train Loss: 0.279567, Train Accuracy: 0.898875
    Epoch 46 @ step 293000: Train Loss: 0.273709, Train Accuracy: 0.903375
  Epoch 46 Test Loss: 0.476162, Test Accuracy: 0.859800, time: 55.0s
    Epoch 47 @ step 294000: Train Loss: 0.284545, Train Accuracy: 0.899750
    Epoch 47 @ step 295000: Train Loss: 0.263776, Train Accuracy: 0.908375
    Epoch 47 @ step 296000: Train Loss: 0.269430, Train Accuracy: 0.905750
    Epoch 47 @ step 297000: Train Loss: 0.277759, Train Accuracy: 0.904375
    Epoch 47 @ step 298000: Train Loss: 0.276979, Train Accuracy: 0.903500
    Epoch 47 @ step 298000: Train Loss: 0.276979, Train Accuracy: 0.903500
    Epoch 47 @ step 299000: Train Loss: 0.270801, Train Accuracy: 0.902125
  Epoch 47 Test Loss: 0.472568, Test Accuracy: 0.854800, time: 55.5s
    Epoch 48 @ step 300000: Train Loss: 0.293332, Train Accuracy: 0.892125
    Epoch 48 @ step 301000: Train Loss: 0.270018, Train Accuracy: 0.902875
    Epoch 48 @ step 302000: Train Loss: 0.267800, Train Accuracy: 0.906625
    Epoch 48 @ step 303000: Train Loss: 0.264126, Train Accuracy: 0.907500
    Epoch 48 @ step 304000: Train Loss: 0.271735, Train Accuracy: 0.905000
    Epoch 48 @ step 305000: Train Loss: 0.294541, Train Accuracy: 0.896000
    Epoch 48 @ step 306000: Train Loss: 0.288568, Train Accuracy: 0.897750
  Epoch 48 Test Loss: 0.482305, Test Accuracy: 0.853400, time: 54.3s
    Epoch 49 @ step 307000: Train Loss: 0.250456, Train Accuracy: 0.909625
    Epoch 49 @ step 308000: Train Loss: 0.257814, Train Accuracy: 0.909750
    Epoch 49 @ step 309000: Train Loss: 0.273443, Train Accuracy: 0.902000
    Epoch 49 @ step 310000: Train Loss: 0.289521, Train Accuracy: 0.899875
    Epoch 49 @ step 311000: Train Loss: 0.271125, Train Accuracy: 0.903625
    Epoch 49 @ step 312000: Train Loss: 0.267659, Train Accuracy: 0.905125
  Epoch 49 Test Loss: 0.491337, Test Accuracy: 0.851900, time: 53.2s
    Epoch 50 @ step 313000: Train Loss: 0.269322, Train Accuracy: 0.905000
    Epoch 50 @ step 314000: Train Loss: 0.257147, Train Accuracy: 0.910125
    Epoch 50 @ step 315000: Train Loss: 0.267232, Train Accuracy: 0.903875
    Epoch 50 @ step 316000: Train Loss: 0.278843, Train Accuracy: 0.904000
    Epoch 50 @ step 317000: Train Loss: 0.272410, Train Accuracy: 0.904750
    Epoch 50 @ step 318000: Train Loss: 0.265433, Train Accuracy: 0.905000
  Epoch 50 Test Loss: 0.461281, Test Accuracy: 0.856800, time: 56.4s
    Epoch 51 @ step 319000: Train Loss: 0.271007, Train Accuracy: 0.906750
    Epoch 51 @ step 320000: Train Loss: 0.244221, Train Accuracy: 0.915500
    Epoch 51 @ step 321000: Train Loss: 0.261501, Train Accuracy: 0.906250
    Epoch 51 @ step 322000: Train Loss: 0.265826, Train Accuracy: 0.905625
    Epoch 51 @ step 323000: Train Loss: 0.271416, Train Accuracy: 0.907250
    Epoch 51 @ step 324000: Train Loss: 0.267698, Train Accuracy: 0.906875
  Epoch 51 Test Loss: 0.478503, Test Accuracy: 0.856200, time: 54.2s
    Epoch 52 @ step 325000: Train Loss: 0.280802, Train Accuracy: 0.904625
    Epoch 52 @ step 326000: Train Loss: 0.236664, Train Accuracy: 0.919375
    Epoch 52 @ step 327000: Train Loss: 0.266233, Train Accuracy: 0.904500
    Epoch 52 @ step 328000: Train Loss: 0.259377, Train Accuracy: 0.909625
    Epoch 52 @ step 329000: Train Loss: 0.261511, Train Accuracy: 0.910125
    Epoch 52 @ step 330000: Train Loss: 0.265638, Train Accuracy: 0.905125
    Epoch 52 @ step 331000: Train Loss: 0.272435, Train Accuracy: 0.904750
  Epoch 52 Test Loss: 0.473082, Test Accuracy: 0.859400, time: 54.1s
    Epoch 53 @ step 332000: Train Loss: 0.243633, Train Accuracy: 0.912625
    Epoch 53 @ step 333000: Train Loss: 0.244713, Train Accuracy: 0.913125
    Epoch 53 @ step 334000: Train Loss: 0.249255, Train Accuracy: 0.913500
    Epoch 53 @ step 335000: Train Loss: 0.260321, Train Accuracy: 0.910750
    Epoch 53 @ step 336000: Train Loss: 0.275853, Train Accuracy: 0.903250
    Epoch 53 @ step 337000: Train Loss: 0.287953, Train Accuracy: 0.901750
  Epoch 53 Test Loss: 0.457739, Test Accuracy: 0.860500, time: 52.2s
    Epoch 54 @ step 338000: Train Loss: 0.259021, Train Accuracy: 0.907875
    Epoch 54 @ step 339000: Train Loss: 0.237624, Train Accuracy: 0.918375
    Epoch 54 @ step 340000: Train Loss: 0.259761, Train Accuracy: 0.909375
    Epoch 54 @ step 341000: Train Loss: 0.260864, Train Accuracy: 0.907625
    Epoch 54 @ step 342000: Train Loss: 0.248091, Train Accuracy: 0.914875
    Epoch 54 @ step 343000: Train Loss: 0.277833, Train Accuracy: 0.902250
  Epoch 54 Test Loss: 0.486501, Test Accuracy: 0.859900, time: 54.0s
    Epoch 55 @ step 344000: Train Loss: 0.256852, Train Accuracy: 0.908500
    Epoch 55 @ step 345000: Train Loss: 0.244455, Train Accuracy: 0.912500
    Epoch 55 @ step 346000: Train Loss: 0.249632, Train Accuracy: 0.912875
```

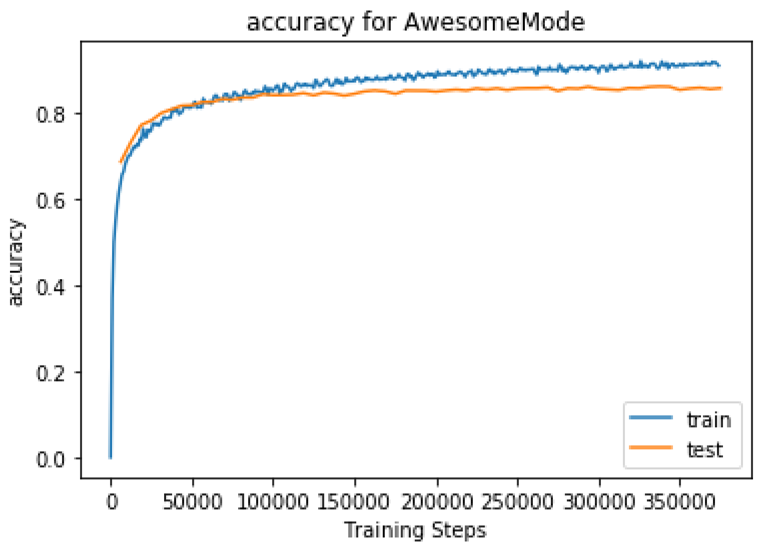
What changes did you make to improve your model?

We used BatchNorm2d after each Conv2d layer to normalize data. After two Conv2d-ReLU-BatchNorm2d series, MaxPool2d was added to extract significant information from convolutional data followed by a Dropout layer to decrease overfitting. The whole process was repeated two more times before Flatten() and 2 fully-connected Linear layers. We trained for 60 epochs to ensure maximum test accuracy. However, it seems like the model started overfitting and test accuracy stopped increasing just after 50,000 steps. The optimizer was also changed to Adam optimizer due to its general effectiveness in reducing loss in computer vision applications. We also experimented on adding another set of convolutional layers and Linear layer, but these approaches did not improve the test accuracy.

```
      Epoch 57 @ step 358000: Train Loss: 0.253021, Train Accuracy: 0.912125
```

```
plot_graphs("AwesomeMode", metrics)
```





After you get a nice model, download the test_file.zip and unzip it to get test_file.pt. In colab, you can explore your files from the left side bar. You can also download the files to your machine from there.

```
!wget http://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_file.zip
!unzip test_file.zip
```

```
--2019-04-28 02:03:41--  http://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/t
Resolving courses.engr.illinois.edu (courses.engr.illinois.edu)... 130.126.151.9
Connecting to courses.engr.illinois.edu (courses.engr.illinois.edu)|130.126.151.9|:80
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_file.zip |
--2019-04-28 02:03:41--  https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/
Connecting to courses.engr.illinois.edu (courses.engr.illinois.edu)|130.126.151.9|:44
HTTP request sent, awaiting response... 200 OK
Length: 3841776 (3.7M) [application/x-zip-compressed]
Saving to: ‘test_file.zip’

test_file.zip       100%[===================>]   3.66M  5.69MB/s    in 0.6s

2019-04-28 02:03:42 (5.69 MB/s) - ‘test_file.zip’ saved [3841776/3841776]

Archive:  test_file.zip
```

Then use your model to predict the label of the test images. Fill the remaining code below, where x has two dimensions (batch_size x one image size). Remember to reshpe x accordingly before feeding it into your model. The submission.txt should contain one predicted label (0~9) each line. Submit your submission.txt to the competition in gradscope.

```
import torch.utils.data as Data

test_file = 'test_file.pt'

tensor = torch.load(test_file)
torch_dataset = Data.TensorDataset(tensor)
test_loader = torch.utils.data.DataLoader(torch_dataset, batch_size, shuffle=False, num_work
all_lab = []
for ele in test_loader:
    x = ele[0]
    x = x.view(-1,3,32,32)
    model.eval()
    with torch.no_grad():
      x = x.to(device)
      output = model(x)
      ind = torch.argmax(output, dim=1)
      lab = ind.cpu().numpy()
      all_lab.append(lab)
all_lab = np.array(all_lab)
all_lab = all_lab.flatten()
np.savetxt('drive/My Drive/CS 498 AML/HW10/submission.txt', all_lab, fmt='%d', delimiter=','
```

HW10 Report

Part 0: Imports and Basic Setup (5 Points)

Part 1: Fully connected neural networks (25 Points)

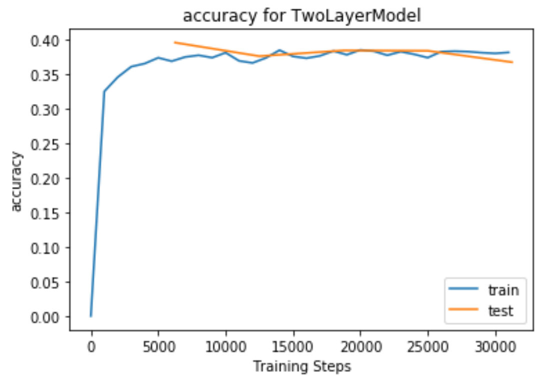
Test (on validation set) accuracy (5 Points): `0.368100`

Test loss (5 Points): `1.747959`

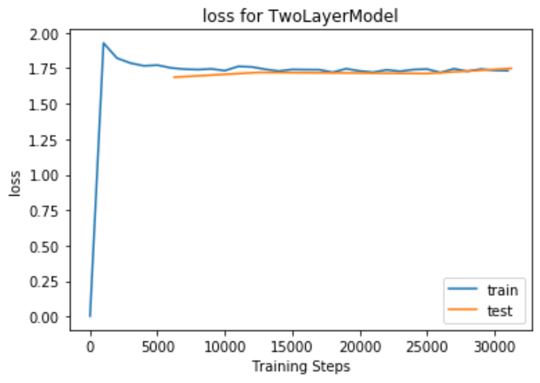
Training time (5 Points): `86.6s`

Plots:

Plot a graph of accuracy on validation set vs training steps (5 Points)



Plot a graph of loss on validation set vs training steps (5 Points)



Part 2: Convolution Network (Basic) (35 Points)

Tensor dimensions: A good way to debug your network for size mismatches is to print the dimension of output after every layer: (10 Points)

Output dimension after 1st conv layer: (8, 16, 32, 32)

Output dimension after 1st max pooling: (8, 16, 16, 16)

Output dimension after 2nd conv layer: (8, 16, 16, 16)

Output dimension after flatten layer: (8, 4096)

Output dimension after 1st fully connected layer: (8, 64)

Output dimension after 2nd fully connected layer: (8, 10)

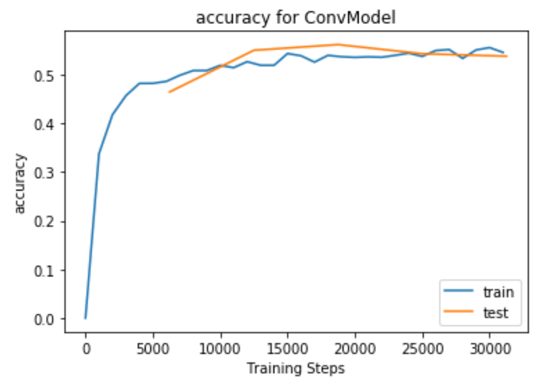
Test (on validation set) Accuracy (5 Points): `0.537800`

Test loss (5 Points): `1.281966`

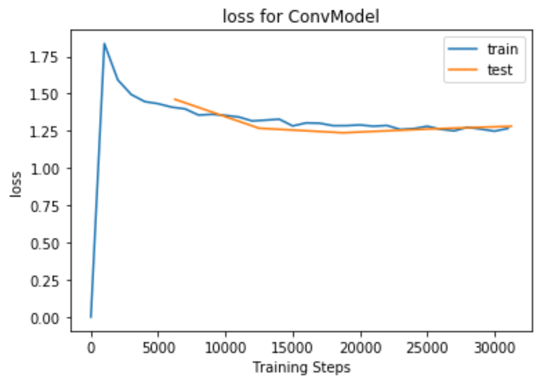
Training time (5 Points): `125.5s`

Plots:

Plot a graph of accuracy on validation set vs training steps (5 Points)



Plot a graph of loss on validation set vs training steps (5 Points)



Part 3: Convolution Network (Add one or more suggested changes) (35 Points)

Describe the additional changes implemented, your intuition for as to why it works, you may also describe other approaches you experimented with (10 Points):

We used BatchNorm2d after each Conv2d layer to normalize data. After two Conv2d-ReLU-BatchNorm2d series, MaxPool2d was added to extract significant information from convolutional data followed by a Dropout layer to decrease overfitting. The whole process was repeated two more times before Flatten() and 2 fully-connected Linear layers. We trained for 60 epochs to ensure maximum test accuracy. However, it seems like the model started overfitting and test accuracy stopped increasing just *after 50,000 steps. The optimizer was also changed to Adam optimizer due to its general effectiveness in reducing loss in computer vision applications. We also experimented on adding another set of convolutional layers and Linear layer, but these approaches did not improve the test accuracy.*

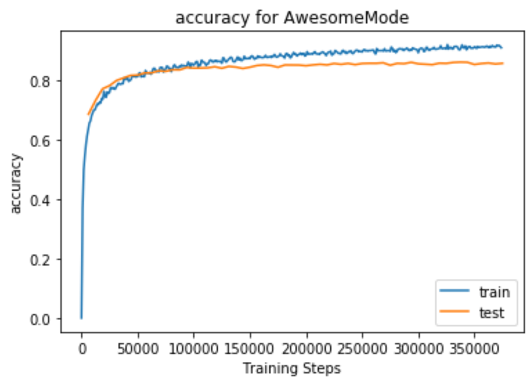
Test (on validation set) Accuracy (5 Points): `0.856400`

Test loss (5 Points): `0.474526`

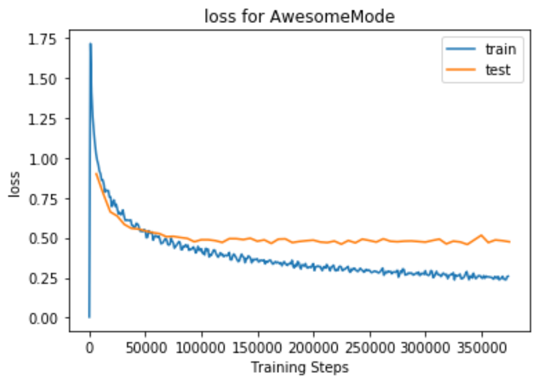
Training time (5 Points): `3282.2s`

Plots:

Plot a graph of accuracy on validation set vs training steps (5 Points)



Plot a graph of loss on validation set vs training steps (5 Points)


loss for AwesomeMode
loss
Training Steps
train
test