

Sam Chiang, Risha Madineni, Roy Wang, Kurtis Jia

## ▼ Assignment 4: Deep Learning

This assignment will give you practice using the library PyTorch, which is a popular library for constructing deep learning models. Deep Learning is a very powerful tool because it is able to learn very complicated functions. Deep Learning has revolutionized fields like image and speech recognition.

The specific task we are trying to solve in this assignment is image classification. We will be using a very common dataset called CIFAR-10 that has 60,000 images separated into 10 classes. The classes are

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

In this assignment, you will practice:

- Reading documentation for a modern machine learning library
- Writing PyTorch code.
- Evaluating neural network models.

**Unless otherwise noted, every answer you submit should have code that clearly shows the answer in the output.** Answers submitted that do not have associated code that shows the answer may not be accepted for credit.

**Make sure to restart the kernel and run all cells** (especially before turning it in) to make sure your code runs correctly.

---

## Part 1 - Background Reading

Before starting this assignment, you should familiarize yourself with PyTorch. There are two options for this. 1) Work through the tutorial linked on Canvas with the assignment. It is interactive and uses Colab. 2) Work through [their tutorial](#) and preferably follow along with a Colab notebook. You should only have to fully read the sections:

- What is PyTorch?

- Autograd: Automatic Differentiation
  - Just skim this section, it's a bit advanced but good to be aware of.
- Neural Networks
- Training a Classifier


You will probably also want to look at the [documentation](#) to see what the specific values they are passing in during the tutorial.

## Assignment

**Before you start, make sure you have enabled GPU for your Colab Runtime.** In the Runtime drop down above, click on "Change runtime type" and select "GPU" in the Hardware Accelerator field.

First we import all of the modules we will use in this assignment.

```
import math
import numpy as np
import matplotlib.pyplot as plt          # For plotting
import seaborn as sns                   # For styling plots
import torch                            # Overall PyTorch import
import torch.nn as nn                  # For specifying neural networks
import torch.nn.functional as F        # For common functions
import torch.optim as optim            # For optimizing model parameters
import torchvision.datasets as datasets # To download data
import torchvision.transforms as transforms # For pre-processing data
import torchvision.utils as utils
sns.set()
```

```
 /usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the fun
import pandas.util.testing as tm
```

Then we define some constants that will be used throughout the program.

```
INPUT_SIZE    = 3 * 32 * 32    # An image has 32 x 32 pixels each with Red/Green/Blue values.
NUM_CLASSES   = 10             # The number of output classes. In this case, from 0 to 9
NUM_EPOCHS    = 20             # The number of times we loop over the whole dataset during training
BATCH_SIZE    = 100            # The size of input data took for one iteration of an epoch
LEARNING_RATE = 1e-3           # The speed of convergence
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

The first step is to download the dataset using PyTorch. The code below produces a train set and a test set. Some notes on PyTorch terminology:

- Most of the time we use the `train_loader` instead of the `trainset`. The reason being the loader gives us batches of examples from the entire `trainset`. Looping over the `train_loader` returns a subset of the examples on each iteration.
- PyTorch commonly talks about "tensors". For our purposes, a tensor is just a multi-dimensional array. For example, the `trainset` is a tensor with shape `(50000, 32, 32, 3)` because there are 50,000 images, each is 32x32 pixels with 3 color channels (Red/Green/Blue). You can index into these just like `numpy` arrays.
- We pre-process the data into tensors and then normalize them so all the pixels have mean 0.5 and standard deviation 0.5.
- Training the models on the full dataset can take a long time. We have left some commented out code to help you sample the training/test data to speed up development time. We encourage you to un-comment the code line while you are figuring out how to write the code so it takes less time to run as long as you re-comment the lines out before training your final models.

**IMPORTANT: Make sure you comment the sampling lines out and re-run your solution on the full dataset to get the correct results.** If you do not correctly train the models on the full dataset in your final submission, you will not get full credit.

```
# We do a small amount of standardization on the image.
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset = datasets.CIFAR10(root='./data', train=True, download=True,
                             transform=transform)

testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Uncomment the two lines below to speed up train time when debugging if your code works.
# It effectively takes a small chunk of the training and test set
# MAKE SURE TO RECOMMENT THE LINE OF CODE BEFORE SUBMITTING. Otherwise your results will be wrong.
#trainset, _ = torch.utils.data.random_split(trainset, [BATCH_SIZE * 10, len(trainset) - BATCH_SIZE * 10])
#testset, _ = torch.utils.data.random_split(testset, [BATCH_SIZE * 10, len(testset) - BATCH_SIZE * 10])

train_loader = torch.utils.data.DataLoader(trainset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)

# Uncomment line below to speed up train time when testing if your code works.
# MAKE SURE TO RECOMMENT THE LINE OF CODE BEFORE SUBMITTING. Otherwise your results will be wrong.
#test_loader = torch.utils.data.DataLoader(testset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
```



Files already downloaded and verified  
Files already downloaded and verified

## ▼ Part 2: Visualizing the Data (5pts)

In the following we will understand the dataset a bit better and visualize some of the images in CIFAR-10. In the following we extract a batch of images from the dataset and visualize them.

**Question:** Explain the shape of `images`. Which dimension corresponds to the number of images? Which to the color channels? And which to the shape of the image?

Answer: The dimensions and what these correspond to as below:

Number of Images: 100

Color Channels: 3

Shape of image: 32 x 32

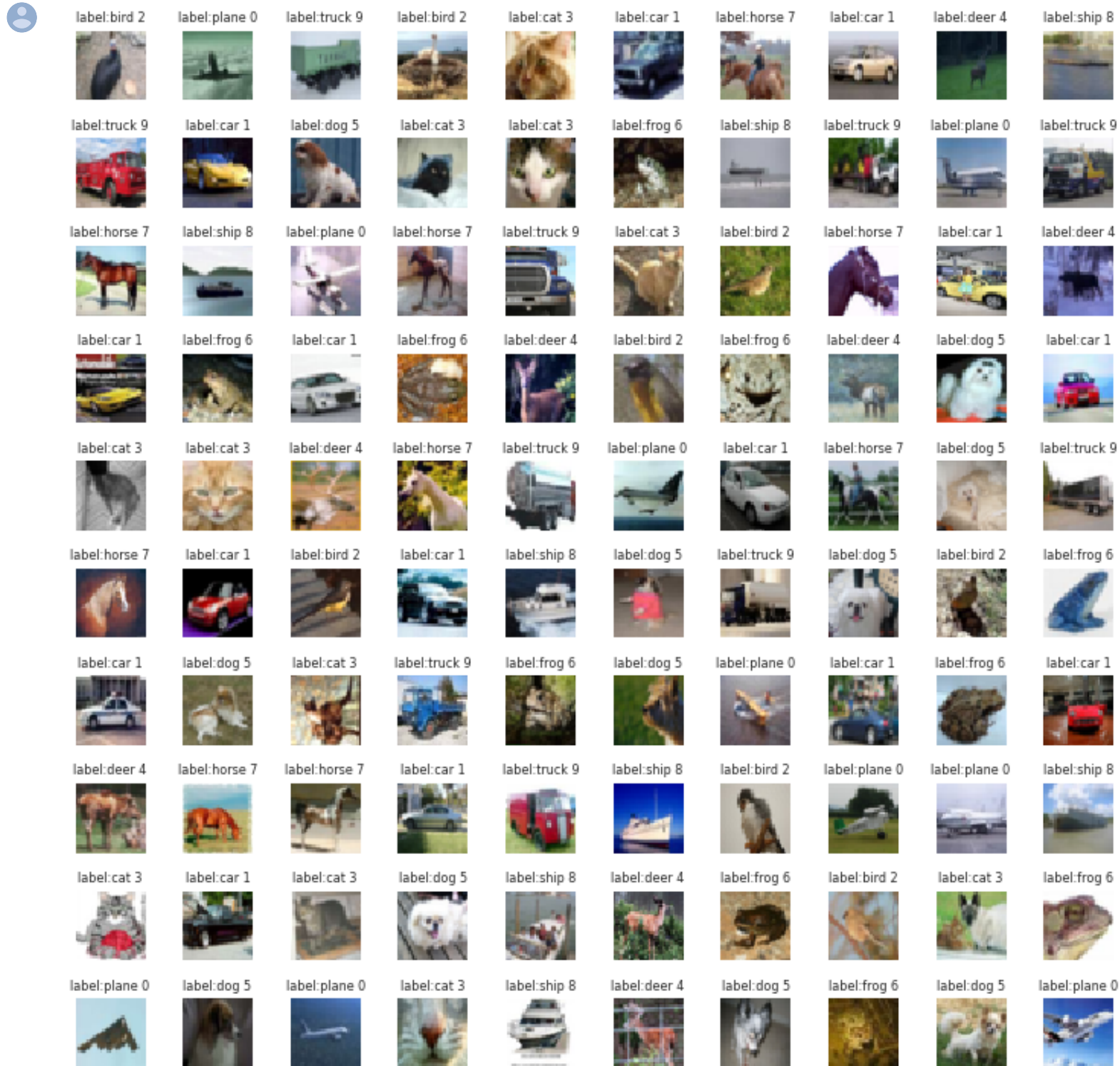
```
#@title
dataiter = iter(test_loader)
images_batch, labels_batch = dataiter.next()
print('size of images batch', images_batch.shape)
print('size of labels', labels_batch.shape)
```



```
size of images batch torch.Size([100, 3, 32, 32])
size of labels torch.Size([100])
```

Now we display a few of the images along with their labels.

```
def imshow(imgs, labels):
    imgs = imgs / 2 + 0.5      # unnormalize
    npimgs = imgs.numpy()[:100]
    fig, ax = plt.subplots(10,10, figsize=(10, 10))
    for i,img in enumerate(npimgs):
        ax[i//10, i%10].imshow(img.transpose(1,2,0), interpolation='nearest')
        ax[i//10, i%10].set_title('label:{} {}'.format(classes[labels[i]], labels[i]), fontsize=8)
        ax[i//10, i%10].set_axis_off()
    fig.tight_layout(h_pad=1, w_pad=1)
    plt.show()
imshow(images_batch, labels_batch)
```



## ▼ Part 3

---

In the cell below, we define the critical helper functions to train and visualize the results of the neural networks.

You should read and understand what they are doing since the next step will ask you to modify one of them.

### Task 1 (5pts)

Right now the `train` function works, but we want to add extra functionality so that it keeps track of how the train/test accuracies change as training progresses (if the `compute_accs` parameter is true). It would be very inefficient to evaluate the train/test accuracy at the end of each batch, so instead we should only evaluate it at the end of each training epoch.

Modify the code so that at the end of each epoch if `compute_accs` is true, it computes the training and test accuracies and puts those values at the end of `train_accs` and `test_accs` respectively.

```
def train(net, train_loader, test_loader,
          num_epochs=NUM_EPOCHS, learning_rate=LEARNING_RATE,
          compute_accs=False):
    """
    This function trains the given network on the training data for the given number of epochs.
    If compute_accs is true, evaluates the train and test accuracy of the network at the end of
    each epoch.

    Args:
        net: The neural network to train
        train_loader, test_loader: The pytorch dataset loaders for the trainst and testset
        num_epochs: The number of times to loop over the batches in train_loader
        learning_rate: The learning rate for the optimizer
        compute_accs: A bool flag for whether or not this function should compute the train and test
                      accuracies at the end of each epoch. This feature is useful for visualizing
                      how the model is learning, but slows down training time.

    Returns:
        The train and test accuracies if compute_accs is True, None otherwise
    """
    # First initialize the criterion (loss function) and the optimizer
    # (algorithm like gradient descent). Here we use a common loss function for multi-class
    # classification called the Cross Entropy Loss and the popular Adam algorithm for gradient descent.
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

    train_accs = []
    test_accs = []

    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(train_loader):  # Loop over each batch in train_loader
```

```

# If you are using a GPU, speed up computation by moving values to the GPU
if torch.cuda.is_available():
    net = net.cuda()
    images = images.cuda()
    labels = labels.cuda()

optimizer.zero_grad()          # Reset gradient for next computation
outputs = net(images)          # Forward pass: compute the output class given a image
loss = criterion(outputs, labels) # Compute loss: difference between the pred and true
loss.backward()                # Backward pass: compute the weight
optimizer.step()               # Optimizer: update the weights of hidden nodes

if (i + 1) % 100 == 0: # Print every 100 batches
    print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(train_loader)}], '
          f'Loss: {loss.item():.4f}')
#TODO: use the accuracy function below to compute the train_acc and test_acc
# and append them to train_accs and test_accs
if compute_accs:
    train_acc = accuracy(net, train_loader) #TODO
    test_acc = accuracy(net, test_loader) #TODO
    train_accs.append(train_acc)
    test_accs.append(test_acc)
    print(f'Epoch [{epoch + 1}/{num_epochs}], Train Accuracy {100 * train_acc:.2f}%, Test Accuracy {100 * test_a

if compute_accs:
    return train_accs, test_accs
else:
    return None

def accuracy(net, data_loader):
    """
    For a given data_loader, evaluate the model on the dataset and compute its classification
    accuracy.

    Args:
        net: The neural network to train
        data_loader: A dataset loader for some dataset.
    Returns:
        The classification accuracy of the model on this dataset.
    """
    correct = 0
    total = 0
    for images, labels in data_loader:
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

```

```

        outputs = net(images)                # Make predictions
        _, predicted = torch.max(outputs.data, 1) # Choose class with highest scores
        total += labels.size(0)              # Increment the total count
        correct += (predicted == labels).sum().item() # Increment the correct count

    return correct / total

def plot_history(histories):
    """
    Given a series of training/test accuracies from training, plots them to visualize learning.

    Args:
        histories: A list of dictionaries storing information about each model trained.
                   Each dictionary should have the keys:
                       * name: The model name
                       * train_accs: A list of train accuracies
                       * test_accs: A list of test accuracies.
    """
    plt.figure(figsize=(16,10))
    epochs = list(range(1, len(histories[0]['train_accs']) + 1))
    for model_history in histories:
        val = plt.plot(epochs, model_history['test_accs'],
                       '--', label=model_history['name'] + ' Test')
        plt.plot(epochs, model_history['train_accs'], color=val[0].get_color(),
                 label=model_history['name'] + ' Train')

    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.xlim([1,max(epochs)])

```

## ▼ Task 2 (20pts)

Now with all the helper code done, we are going to actually write the PyTorch code that specifies the neural network models. Each of the models described should be a different Python class with the specified name.

### NetA

The first neural network will be the simplest, in that it has no hidden layers. It should take the image and flatten it to a vector for the input, and then have 10 outputs, one for each class.

There should be no non-linearities for this network and is just a very simple linear classifier.

### NetB



The second neural network will be slightly more complicated in that it has a hidden layer with 300 nodes and adds a non-linearity between the layers. It should use the following operations in this order:

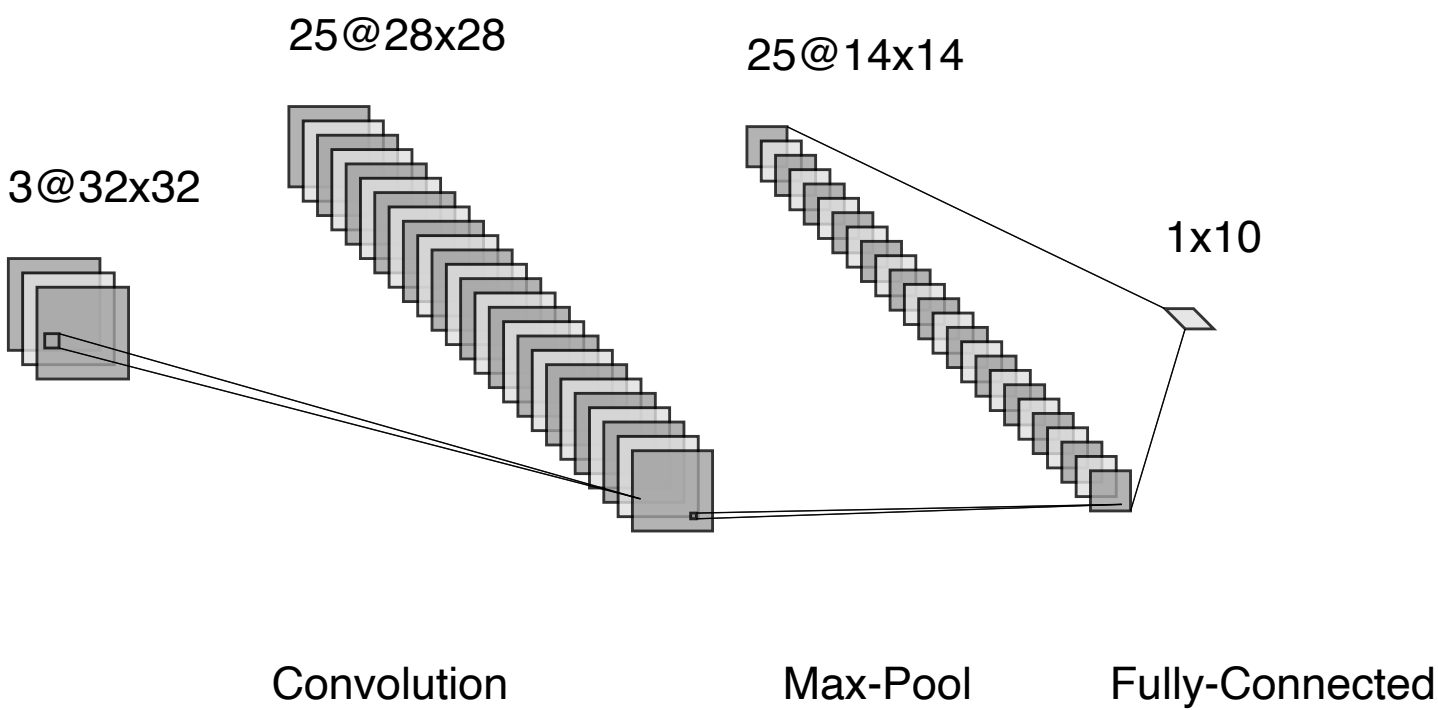
- Flatten the image to a vector for the input
- Use a fully-connected linear layer with 300 hidden-neurons
- Use the ReLU activation function
- Use a fully-connected linear layer to the 10 outputs.

## NetC

This third neural network will be a convolutional neural network. It should use the following operations in this order:

- Use a convolution layer with kernel-width 5 and depth 25
- Use the ReLU activation function
- Use a max-pool operation with kernel-width 2 and stride 2
- Flatten the image to a vector for the next step's input
- Use a fully-connected linear layer to the 10 outputs.

This architecture can be seen visually in the following diagram (the left-most object is the input image).



Notice that these diagrams use the notation

`channels @ height x width`

to describe the dimensions of the results at each step. You may find the code from the Lecture 7 in class useful.

```
class NetA(nn.Module):
    def __init__(self):
        super(NetA, self).__init__()
        #TODO
        self.linear = nn.Sequential(
            nn.Flatten(),
            nn.Linear(3*32*32, 10)
        ) # Initializing the first linear layer
    def forward(self, x):
        #TODO
        output = self.linear(x)

        return output

class NetB(nn.Module):
    def __init__(self):
        super(NetB, self).__init__()
        #TODO
        self.linear = nn.Sequential(
            nn.Flatten(),nn.Linear(INPUT_SIZE,300), nn.Linear(300,10)
        )

    def forward(self, x):
        #TODO
        output = F.relu(self.linear(x))
        return output

class NetC(nn.Module):
    def __init__(self,):
        super(NetC, self).__init__()
        #TODO
        self.conv = nn.Conv2d(3, 25, 5)
        self.pool = nn.MaxPool2d((2, 2))
        self.lin = nn.Linear(25 * 14 * 14 , 10)

    def forward(self, x):
        #TODO
        x = self.conv(x)
        x = F.relu(x)
        x = self.pool(x)
        x = x.view(-1, 25 * 14 * 14)
        x = self.lin(x)
        return x
```

Below is the cell that runs the main part of the code. It trains each of the 3 models you specified above for 10 epochs and then plots their train/test accuracies throughout the training process. When training all 3 models on the entire datasets, this should take about 15-20 minutes to run.

**You are welcome (and encouraged) to change the `nets` list during development so you don't have to train all 3 models each time, but make sure your final result trains. For debugging purposes, you can uncomment the lines above where the a smaller portion of the training and test set are loaded.**

```
nets = [NetA(), NetB(), NetC()]

%%time
nets = [NetA(), NetB(), NetC()]
histories = []

for net in nets:
    net_name = type(net).__name__
    print(f'==== Training {net_name} ====')
    train_history, test_history = train(net, train_loader, test_loader,
                                       num_epochs=NUM_EPOCHS,
                                       learning_rate=LEARNING_RATE,
                                       compute_accs=True)

    histories.append({
        'name': net_name,
        'train_accs': train_history,
        'test_accs': test_history
    })
plot_history(histories)
```



```
==== Training NetA ====
Epoch [1/20], Step [100/500], Loss: 1.9936
Epoch [1/20], Step [200/500], Loss: 1.6307
Epoch [1/20], Step [300/500], Loss: 1.6792
Epoch [1/20], Step [400/500], Loss: 1.7605
Epoch [1/20], Step [500/500], Loss: 2.0868
Epoch [1/20], Train Accuracy 39.63%, Test Accuracy 42.50%
Epoch [2/20], Step [100/500], Loss: 1.8044
Epoch [2/20], Step [200/500], Loss: 1.7690
Epoch [2/20], Step [300/500], Loss: 1.8710
Epoch [2/20], Step [400/500], Loss: 1.7601
Epoch [2/20], Step [500/500], Loss: 1.7717
Epoch [2/20], Train Accuracy 41.04%, Test Accuracy 40.90%
Epoch [3/20], Step [100/500], Loss: 1.7125
Epoch [3/20], Step [200/500], Loss: 1.6896
Epoch [3/20], Step [300/500], Loss: 1.7463
Epoch [3/20], Step [400/500], Loss: 1.9901
Epoch [3/20], Step [500/500], Loss: 1.7921
Epoch [3/20], Train Accuracy 42.43%, Test Accuracy 43.20%
```

```

Epoch [4/20], Step [100/500], Loss: 1.8135
Epoch [4/20], Step [200/500], Loss: 1.7973
Epoch [4/20], Step [300/500], Loss: 1.7299
Epoch [4/20], Step [400/500], Loss: 1.8329
Epoch [4/20], Step [500/500], Loss: 1.7775
Epoch [4/20], Train Accuracy 42.08%, Test Accuracy 40.90%
Epoch [5/20], Step [100/500], Loss: 1.6281
Epoch [5/20], Step [200/500], Loss: 1.7697
Epoch [5/20], Step [300/500], Loss: 1.7120
Epoch [5/20], Step [400/500], Loss: 1.6500
Epoch [5/20], Step [500/500], Loss: 1.6903
Epoch [5/20], Train Accuracy 42.97%, Test Accuracy 43.40%
Epoch [6/20], Step [100/500], Loss: 1.7293
Epoch [6/20], Step [200/500], Loss: 1.7621
Epoch [6/20], Step [300/500], Loss: 1.8276
Epoch [6/20], Step [400/500], Loss: 1.5356
Epoch [6/20], Step [500/500], Loss: 1.6349
Epoch [6/20], Train Accuracy 43.01%, Test Accuracy 42.60%
Epoch [7/20], Step [100/500], Loss: 1.6767
Epoch [7/20], Step [200/500], Loss: 1.9427
Epoch [7/20], Step [300/500], Loss: 1.6141
Epoch [7/20], Step [400/500], Loss: 1.9568
Epoch [7/20], Step [500/500], Loss: 1.8324
Epoch [7/20], Train Accuracy 41.91%, Test Accuracy 40.30%
Epoch [8/20], Step [100/500], Loss: 1.5881
Epoch [8/20], Step [200/500], Loss: 1.9704
Epoch [8/20], Step [300/500], Loss: 1.7030
Epoch [8/20], Step [400/500], Loss: 1.8181
Epoch [8/20], Step [500/500], Loss: 1.6808
Epoch [8/20], Train Accuracy 42.79%, Test Accuracy 40.90%
Epoch [9/20], Step [100/500], Loss: 1.6667
Epoch [9/20], Step [200/500], Loss: 1.6861
Epoch [9/20], Step [300/500], Loss: 1.5453
Epoch [9/20], Step [400/500], Loss: 1.8657
Epoch [9/20], Step [500/500], Loss: 1.7797
Epoch [9/20], Train Accuracy 42.25%, Test Accuracy 42.50%
Epoch [10/20], Step [100/500], Loss: 1.8866
Epoch [10/20], Step [200/500], Loss: 1.6095
Epoch [10/20], Step [300/500], Loss: 1.5536
Epoch [10/20], Step [400/500], Loss: 1.7859
Epoch [10/20], Step [500/500], Loss: 1.7320
Epoch [10/20], Train Accuracy 43.00%, Test Accuracy 40.80%
Epoch [11/20], Step [100/500], Loss: 1.8058
Epoch [11/20], Step [200/500], Loss: 1.7139
Epoch [11/20], Step [300/500], Loss: 1.8255
Epoch [11/20], Step [400/500], Loss: 1.8188
Epoch [11/20], Step [500/500], Loss: 1.7661
Epoch [11/20], Train Accuracy 43.12%, Test Accuracy 40.50%
Epoch [12/20], Step [100/500], Loss: 1.5063
Epoch [12/20], Step [200/500], Loss: 1.8401
Epoch [12/20], Step [300/500], Loss: 1.6620
Epoch [12/20], Step [400/500], Loss: 1.7461
Epoch [12/20], Step [500/500], Loss: 1.7048
Epoch [12/20], Train Accuracy 42.70%, Test Accuracy 39.40%
Epoch [13/20], Step [100/500], Loss: 1.5575

```

```
Epoch [13/20], Step [100/500], Loss: 1.7278
Epoch [13/20], Step [200/500], Loss: 1.7278
Epoch [13/20], Step [300/500], Loss: 1.8019
Epoch [13/20], Step [400/500], Loss: 1.6584
Epoch [13/20], Step [500/500], Loss: 1.8293
Epoch [13/20], Train Accuracy 42.14%, Test Accuracy 40.30%
Epoch [14/20], Step [100/500], Loss: 1.5405
Epoch [14/20], Step [200/500], Loss: 1.7477
Epoch [14/20], Step [300/500], Loss: 1.7454
Epoch [14/20], Step [400/500], Loss: 1.6691
Epoch [14/20], Step [500/500], Loss: 1.7811
Epoch [14/20], Train Accuracy 43.10%, Test Accuracy 43.10%
Epoch [15/20], Step [100/500], Loss: 1.6662
Epoch [15/20], Step [200/500], Loss: 1.8634
Epoch [15/20], Step [300/500], Loss: 1.6675
Epoch [15/20], Step [400/500], Loss: 1.6058
Epoch [15/20], Step [500/500], Loss: 1.7998
Epoch [15/20], Train Accuracy 43.22%, Test Accuracy 41.00%
Epoch [16/20], Step [100/500], Loss: 1.7268
Epoch [16/20], Step [200/500], Loss: 2.1180
Epoch [16/20], Step [300/500], Loss: 1.8707
Epoch [16/20], Step [400/500], Loss: 1.9386
Epoch [16/20], Step [500/500], Loss: 1.7644
Epoch [16/20], Train Accuracy 44.32%, Test Accuracy 41.30%
Epoch [17/20], Step [100/500], Loss: 1.5101
Epoch [17/20], Step [200/500], Loss: 1.5913
Epoch [17/20], Step [300/500], Loss: 1.5837
Epoch [17/20], Step [400/500], Loss: 1.6892
Epoch [17/20], Step [500/500], Loss: 1.8308
Epoch [17/20], Train Accuracy 43.76%, Test Accuracy 39.80%
Epoch [18/20], Step [100/500], Loss: 1.8511
Epoch [18/20], Step [200/500], Loss: 1.6878
Epoch [18/20], Step [300/500], Loss: 1.7751
Epoch [18/20], Step [400/500], Loss: 1.6291
Epoch [18/20], Step [500/500], Loss: 1.7701
Epoch [18/20], Train Accuracy 43.13%, Test Accuracy 40.90%
Epoch [19/20], Step [100/500], Loss: 1.6197
Epoch [19/20], Step [200/500], Loss: 1.6474
Epoch [19/20], Step [300/500], Loss: 1.7201
Epoch [19/20], Step [400/500], Loss: 1.8343
Epoch [19/20], Step [500/500], Loss: 1.7806
Epoch [19/20], Train Accuracy 44.15%, Test Accuracy 41.50%
Epoch [20/20], Step [100/500], Loss: 2.0059
Epoch [20/20], Step [200/500], Loss: 1.6610
Epoch [20/20], Step [300/500], Loss: 1.7096
Epoch [20/20], Step [400/500], Loss: 1.4718
Epoch [20/20], Step [500/500], Loss: 1.8440
Epoch [20/20], Train Accuracy 42.98%, Test Accuracy 41.00%
==== Training NetB ====
Epoch [1/20], Step [100/500], Loss: 1.8962
Epoch [1/20], Step [200/500], Loss: 1.8178
Epoch [1/20], Step [300/500], Loss: 1.7158
Epoch [1/20], Step [400/500], Loss: 1.7309
Epoch [1/20], Step [500/500], Loss: 1.6236
Epoch [1/20], Train Accuracy 39.37%, Test Accuracy 40.50%
```

```
Epoch [2/20], Step [100/500], Loss: 1.9437
Epoch [2/20], Step [200/500], Loss: 1.8144
Epoch [2/20], Step [300/500], Loss: 1.5986
Epoch [2/20], Step [400/500], Loss: 1.6821
Epoch [2/20], Step [500/500], Loss: 1.7569
Epoch [2/20], Train Accuracy 41.05%, Test Accuracy 41.90%
Epoch [3/20], Step [100/500], Loss: 1.8483
Epoch [3/20], Step [200/500], Loss: 1.5672
Epoch [3/20], Step [300/500], Loss: 1.8124
Epoch [3/20], Step [400/500], Loss: 1.5836
Epoch [3/20], Step [500/500], Loss: 1.7373
Epoch [3/20], Train Accuracy 42.00%, Test Accuracy 43.60%
Epoch [4/20], Step [100/500], Loss: 1.7000
Epoch [4/20], Step [200/500], Loss: 1.8101
Epoch [4/20], Step [300/500], Loss: 1.8935
Epoch [4/20], Step [400/500], Loss: 1.7168
Epoch [4/20], Step [500/500], Loss: 1.6900
Epoch [4/20], Train Accuracy 41.86%, Test Accuracy 42.40%
Epoch [5/20], Step [100/500], Loss: 2.0496
Epoch [5/20], Step [200/500], Loss: 1.7094
Epoch [5/20], Step [300/500], Loss: 1.5895
Epoch [5/20], Step [400/500], Loss: 1.7393
Epoch [5/20], Step [500/500], Loss: 1.8656
Epoch [5/20], Train Accuracy 42.19%, Test Accuracy 42.50%
Epoch [6/20], Step [100/500], Loss: 1.9867
Epoch [6/20], Step [200/500], Loss: 1.7200
Epoch [6/20], Step [300/500], Loss: 1.6971
Epoch [6/20], Step [400/500], Loss: 1.7605
Epoch [6/20], Step [500/500], Loss: 1.4874
Epoch [6/20], Train Accuracy 42.01%, Test Accuracy 42.10%
Epoch [7/20], Step [100/500], Loss: 1.7096
Epoch [7/20], Step [200/500], Loss: 1.5344
Epoch [7/20], Step [300/500], Loss: 1.6886
Epoch [7/20], Step [400/500], Loss: 1.6721
Epoch [7/20], Step [500/500], Loss: 1.6267
Epoch [7/20], Train Accuracy 42.20%, Test Accuracy 41.80%
Epoch [8/20], Step [100/500], Loss: 1.8163
Epoch [8/20], Step [200/500], Loss: 1.7930
Epoch [8/20], Step [300/500], Loss: 1.7751
Epoch [8/20], Step [400/500], Loss: 1.6766
Epoch [8/20], Step [500/500], Loss: 1.6433
Epoch [8/20], Train Accuracy 43.15%, Test Accuracy 42.80%
Epoch [9/20], Step [100/500], Loss: 1.8731
Epoch [9/20], Step [200/500], Loss: 1.8320
Epoch [9/20], Step [300/500], Loss: 1.6748
Epoch [9/20], Step [400/500], Loss: 1.4983
Epoch [9/20], Step [500/500], Loss: 1.8195
Epoch [9/20], Train Accuracy 42.18%, Test Accuracy 41.80%
Epoch [10/20], Step [100/500], Loss: 1.7053
Epoch [10/20], Step [200/500], Loss: 1.5073
Epoch [10/20], Step [300/500], Loss: 1.7903
Epoch [10/20], Step [400/500], Loss: 1.6240
Epoch [10/20], Step [500/500], Loss: 1.6866
Epoch [10/20], Train Accuracy 43.52%, Test Accuracy 43.70%
Epoch [11/20], Step [100/500], Loss: 1.7942
```

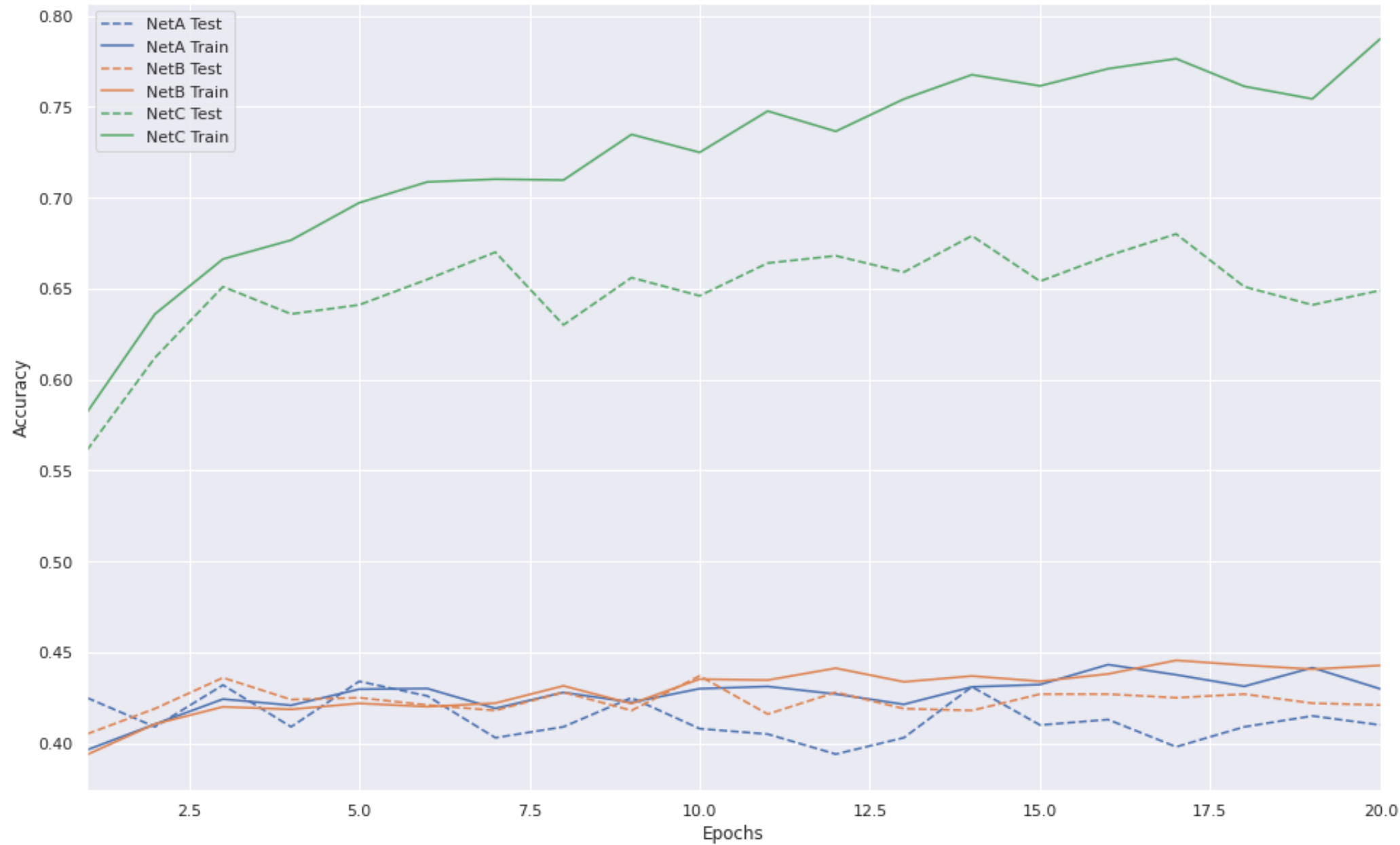
```
Epoch [11/20], Step [200/500], Loss: 1.7651
Epoch [11/20], Step [300/500], Loss: 1.7591
Epoch [11/20], Step [400/500], Loss: 1.5416
Epoch [11/20], Step [500/500], Loss: 1.6898
Epoch [11/20], Train Accuracy 43.47%, Test Accuracy 41.60%
Epoch [12/20], Step [100/500], Loss: 1.5503
Epoch [12/20], Step [200/500], Loss: 1.7787
Epoch [12/20], Step [300/500], Loss: 1.5132
Epoch [12/20], Step [400/500], Loss: 1.5762
Epoch [12/20], Step [500/500], Loss: 1.7292
Epoch [12/20], Train Accuracy 44.12%, Test Accuracy 42.80%
Epoch [13/20], Step [100/500], Loss: 1.6509
Epoch [13/20], Step [200/500], Loss: 1.6714
Epoch [13/20], Step [300/500], Loss: 1.7488
Epoch [13/20], Step [400/500], Loss: 1.7208
Epoch [13/20], Step [500/500], Loss: 1.5249
Epoch [13/20], Train Accuracy 43.37%, Test Accuracy 41.90%
Epoch [14/20], Step [100/500], Loss: 1.7558
Epoch [14/20], Step [200/500], Loss: 1.5704
Epoch [14/20], Step [300/500], Loss: 1.3954
Epoch [14/20], Step [400/500], Loss: 1.5954
Epoch [14/20], Step [500/500], Loss: 1.6389
Epoch [14/20], Train Accuracy 43.69%, Test Accuracy 41.80%
Epoch [15/20], Step [100/500], Loss: 1.6804
Epoch [15/20], Step [200/500], Loss: 1.5590
Epoch [15/20], Step [300/500], Loss: 1.6136
Epoch [15/20], Step [400/500], Loss: 1.6260
Epoch [15/20], Step [500/500], Loss: 1.7007
Epoch [15/20], Train Accuracy 43.40%, Test Accuracy 42.70%
Epoch [16/20], Step [100/500], Loss: 1.6841
Epoch [16/20], Step [200/500], Loss: 1.7432
Epoch [16/20], Step [300/500], Loss: 1.6926
Epoch [16/20], Step [400/500], Loss: 1.6887
Epoch [16/20], Step [500/500], Loss: 1.5338
Epoch [16/20], Train Accuracy 43.81%, Test Accuracy 42.70%
Epoch [17/20], Step [100/500], Loss: 1.8479
Epoch [17/20], Step [200/500], Loss: 1.6646
Epoch [17/20], Step [300/500], Loss: 1.6169
Epoch [17/20], Step [400/500], Loss: 1.7479
Epoch [17/20], Step [500/500], Loss: 1.4757
Epoch [17/20], Train Accuracy 44.56%, Test Accuracy 42.50%
Epoch [18/20], Step [100/500], Loss: 1.5398
Epoch [18/20], Step [200/500], Loss: 1.5387
Epoch [18/20], Step [300/500], Loss: 1.8801
Epoch [18/20], Step [400/500], Loss: 1.7344
Epoch [18/20], Step [500/500], Loss: 1.7206
Epoch [18/20], Train Accuracy 44.29%, Test Accuracy 42.70%
Epoch [19/20], Step [100/500], Loss: 1.6058
Epoch [19/20], Step [200/500], Loss: 1.3941
Epoch [19/20], Step [300/500], Loss: 1.8104
Epoch [19/20], Step [400/500], Loss: 1.7101
Epoch [19/20], Step [500/500], Loss: 1.8428
Epoch [19/20], Train Accuracy 44.07%, Test Accuracy 42.20%
Epoch [20/20], Step [100/500], Loss: 1.7999
Epoch [20/20], Step [200/500], Loss: 1.8284
```



```
Epoch [20/20], Step [200/500], Loss: 1.6334
Epoch [20/20], Step [300/500], Loss: 1.6205
Epoch [20/20], Step [400/500], Loss: 1.6804
Epoch [20/20], Step [500/500], Loss: 1.4267
Epoch [20/20], Train Accuracy 44.28%, Test Accuracy 42.10%
==== Training NetC ====
Epoch [1/20], Step [100/500], Loss: 1.5274
Epoch [1/20], Step [200/500], Loss: 1.6651
Epoch [1/20], Step [300/500], Loss: 1.2556
Epoch [1/20], Step [400/500], Loss: 1.1626
Epoch [1/20], Step [500/500], Loss: 1.0498
Epoch [1/20], Train Accuracy 58.18%, Test Accuracy 56.10%
Epoch [2/20], Step [100/500], Loss: 1.2223
Epoch [2/20], Step [200/500], Loss: 1.1021
Epoch [2/20], Step [300/500], Loss: 1.1554
Epoch [2/20], Step [400/500], Loss: 1.0635
Epoch [2/20], Step [500/500], Loss: 0.9084
Epoch [2/20], Train Accuracy 63.60%, Test Accuracy 61.20%
Epoch [3/20], Step [100/500], Loss: 1.0389
Epoch [3/20], Step [200/500], Loss: 0.9604
Epoch [3/20], Step [300/500], Loss: 1.1214
Epoch [3/20], Step [400/500], Loss: 1.0342
Epoch [3/20], Step [500/500], Loss: 0.9070
Epoch [3/20], Train Accuracy 66.62%, Test Accuracy 65.10%
Epoch [4/20], Step [100/500], Loss: 0.8620
Epoch [4/20], Step [200/500], Loss: 1.0663
Epoch [4/20], Step [300/500], Loss: 0.8946
Epoch [4/20], Step [400/500], Loss: 1.2772
Epoch [4/20], Step [500/500], Loss: 0.9082
Epoch [4/20], Train Accuracy 67.66%, Test Accuracy 63.60%
Epoch [5/20], Step [100/500], Loss: 0.8931
Epoch [5/20], Step [200/500], Loss: 0.9623
Epoch [5/20], Step [300/500], Loss: 1.0205
Epoch [5/20], Step [400/500], Loss: 0.8488
Epoch [5/20], Step [500/500], Loss: 0.9278
Epoch [5/20], Train Accuracy 69.72%, Test Accuracy 64.10%
Epoch [6/20], Step [100/500], Loss: 0.9254
Epoch [6/20], Step [200/500], Loss: 1.1201
Epoch [6/20], Step [300/500], Loss: 1.1102
Epoch [6/20], Step [400/500], Loss: 1.0574
Epoch [6/20], Step [500/500], Loss: 0.9108
Epoch [6/20], Train Accuracy 70.86%, Test Accuracy 65.50%
Epoch [7/20], Step [100/500], Loss: 0.8533
Epoch [7/20], Step [200/500], Loss: 0.7899
Epoch [7/20], Step [300/500], Loss: 0.8016
Epoch [7/20], Step [400/500], Loss: 1.0257
Epoch [7/20], Step [500/500], Loss: 0.8907
Epoch [7/20], Train Accuracy 71.01%, Test Accuracy 67.00%
Epoch [8/20], Step [100/500], Loss: 0.6978
Epoch [8/20], Step [200/500], Loss: 0.8374
Epoch [8/20], Step [300/500], Loss: 0.7614
Epoch [8/20], Step [400/500], Loss: 0.7919
Epoch [8/20], Step [500/500], Loss: 1.0430
Epoch [8/20], Train Accuracy 70.96%, Test Accuracy 63.00%
Epoch [9/20], Step [100/500], Loss: 0.8120
```

```
Epoch [9/20], Step [200/500], Loss: 0.7643
Epoch [9/20], Step [300/500], Loss: 0.8089
Epoch [9/20], Step [400/500], Loss: 0.9875
Epoch [9/20], Step [500/500], Loss: 0.6181
Epoch [9/20], Train Accuracy 73.47%, Test Accuracy 65.60%
Epoch [10/20], Step [100/500], Loss: 0.7798
Epoch [10/20], Step [200/500], Loss: 0.7921
Epoch [10/20], Step [300/500], Loss: 0.9767
Epoch [10/20], Step [400/500], Loss: 0.9214
Epoch [10/20], Step [500/500], Loss: 0.9027
Epoch [10/20], Train Accuracy 72.48%, Test Accuracy 64.60%
Epoch [11/20], Step [100/500], Loss: 0.7811
Epoch [11/20], Step [200/500], Loss: 0.8172
Epoch [11/20], Step [300/500], Loss: 0.8119
Epoch [11/20], Step [400/500], Loss: 0.8550
Epoch [11/20], Step [500/500], Loss: 0.6000
Epoch [11/20], Train Accuracy 74.76%, Test Accuracy 66.40%
Epoch [12/20], Step [100/500], Loss: 0.8494
Epoch [12/20], Step [200/500], Loss: 0.8968
Epoch [12/20], Step [300/500], Loss: 0.7003
Epoch [12/20], Step [400/500], Loss: 0.8307
Epoch [12/20], Step [500/500], Loss: 0.9645
Epoch [12/20], Train Accuracy 73.64%, Test Accuracy 66.80%
Epoch [13/20], Step [100/500], Loss: 0.8696
Epoch [13/20], Step [200/500], Loss: 0.6583
Epoch [13/20], Step [300/500], Loss: 0.9301
Epoch [13/20], Step [400/500], Loss: 0.8807
Epoch [13/20], Step [500/500], Loss: 0.6973
Epoch [13/20], Train Accuracy 75.42%, Test Accuracy 65.90%
Epoch [14/20], Step [100/500], Loss: 0.7014
Epoch [14/20], Step [200/500], Loss: 0.6253
Epoch [14/20], Step [300/500], Loss: 0.7588
Epoch [14/20], Step [400/500], Loss: 0.8602
Epoch [14/20], Step [500/500], Loss: 0.7305
Epoch [14/20], Train Accuracy 76.76%, Test Accuracy 67.90%
Epoch [15/20], Step [100/500], Loss: 0.8563
Epoch [15/20], Step [200/500], Loss: 0.7978
Epoch [15/20], Step [300/500], Loss: 0.8943
Epoch [15/20], Step [400/500], Loss: 0.7674
Epoch [15/20], Step [500/500], Loss: 0.5930
Epoch [15/20], Train Accuracy 76.14%, Test Accuracy 65.40%
Epoch [16/20], Step [100/500], Loss: 0.7880
Epoch [16/20], Step [200/500], Loss: 0.6419
Epoch [16/20], Step [300/500], Loss: 0.6731
Epoch [16/20], Step [400/500], Loss: 0.6176
Epoch [16/20], Step [500/500], Loss: 0.7084
Epoch [16/20], Train Accuracy 77.09%, Test Accuracy 66.80%
Epoch [17/20], Step [100/500], Loss: 0.7151
Epoch [17/20], Step [200/500], Loss: 0.8323
Epoch [17/20], Step [300/500], Loss: 0.7009
Epoch [17/20], Step [400/500], Loss: 1.0112
Epoch [17/20], Step [500/500], Loss: 0.6985
Epoch [17/20], Train Accuracy 77.64%, Test Accuracy 68.00%
Epoch [18/20], Step [100/500], Loss: 0.6235
Epoch [18/20], Step [200/500], Loss: 0.7720
```

```
Epoch [18/20], Step [300/500], Loss: 0.8435
Epoch [18/20], Step [400/500], Loss: 0.7095
Epoch [18/20], Step [500/500], Loss: 0.6479
Epoch [18/20], Train Accuracy 76.12%, Test Accuracy 65.10%
Epoch [19/20], Step [100/500], Loss: 0.5753
Epoch [19/20], Step [200/500], Loss: 0.5509
Epoch [19/20], Step [300/500], Loss: 0.7240
Epoch [19/20], Step [400/500], Loss: 0.7124
Epoch [19/20], Step [500/500], Loss: 0.8017
Epoch [19/20], Train Accuracy 75.43%, Test Accuracy 64.10%
Epoch [20/20], Step [100/500], Loss: 0.6915
Epoch [20/20], Step [200/500], Loss: 0.6948
Epoch [20/20], Step [300/500], Loss: 0.7954
Epoch [20/20], Step [400/500], Loss: 0.6481
Epoch [20/20], Step [500/500], Loss: 0.6851
Epoch [20/20], Train Accuracy 78.74%, Test Accuracy 64.90%
CPU times: user 3min 9s, sys: 54.1 s, total: 4min 4s
Wall time: 22min 33s
```



Double-click (or enter) to edit

**Question:** Which network had the highest Test accuracy after 20 epochs?

NetC had the highest test accuracy after 20 epochs (approx 65%).

## ▼ Part 4: Visualizing the outputs of layers (5 pts)

In this section we visualize the outputs of the first convolutional layer of NetC above. Recall that convolutions capture local data.

**Question:** What is the output dimension after the first layer? Why?

The output dimensions of first convolutional layer of NetC are 25@28\*28, here 25 is the depth (no. of channels) of the layer and 28x28 is the size.

As the kernel size is specified as 5, this convolution changes the dimensions as per calculations done below:

Input size = 32 (i.e. 32\*32)

Kernel size = 5

Stride = 1

Padding = 0

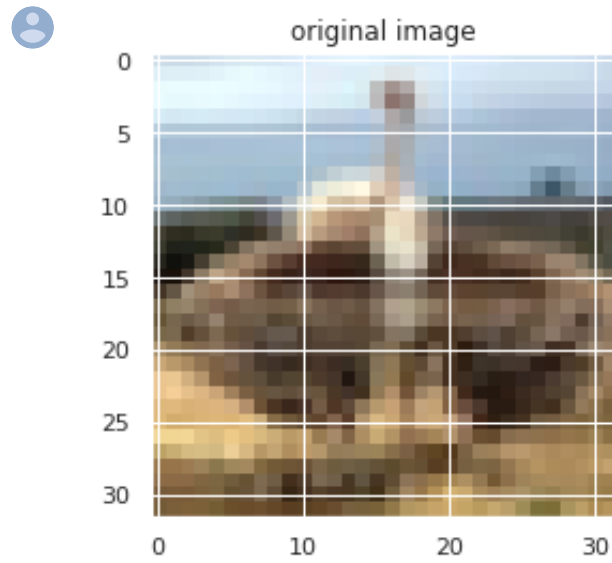
Output size = (Input size - Kernel width)/Stride + 1 = 27+1 = 28

```
import numpy as np
dataiter = iter(test_loader)
images, labels = dataiter.next()
# change this to choose different images
image_idx = 3
inpt = images[image_idx][np.newaxis,...]

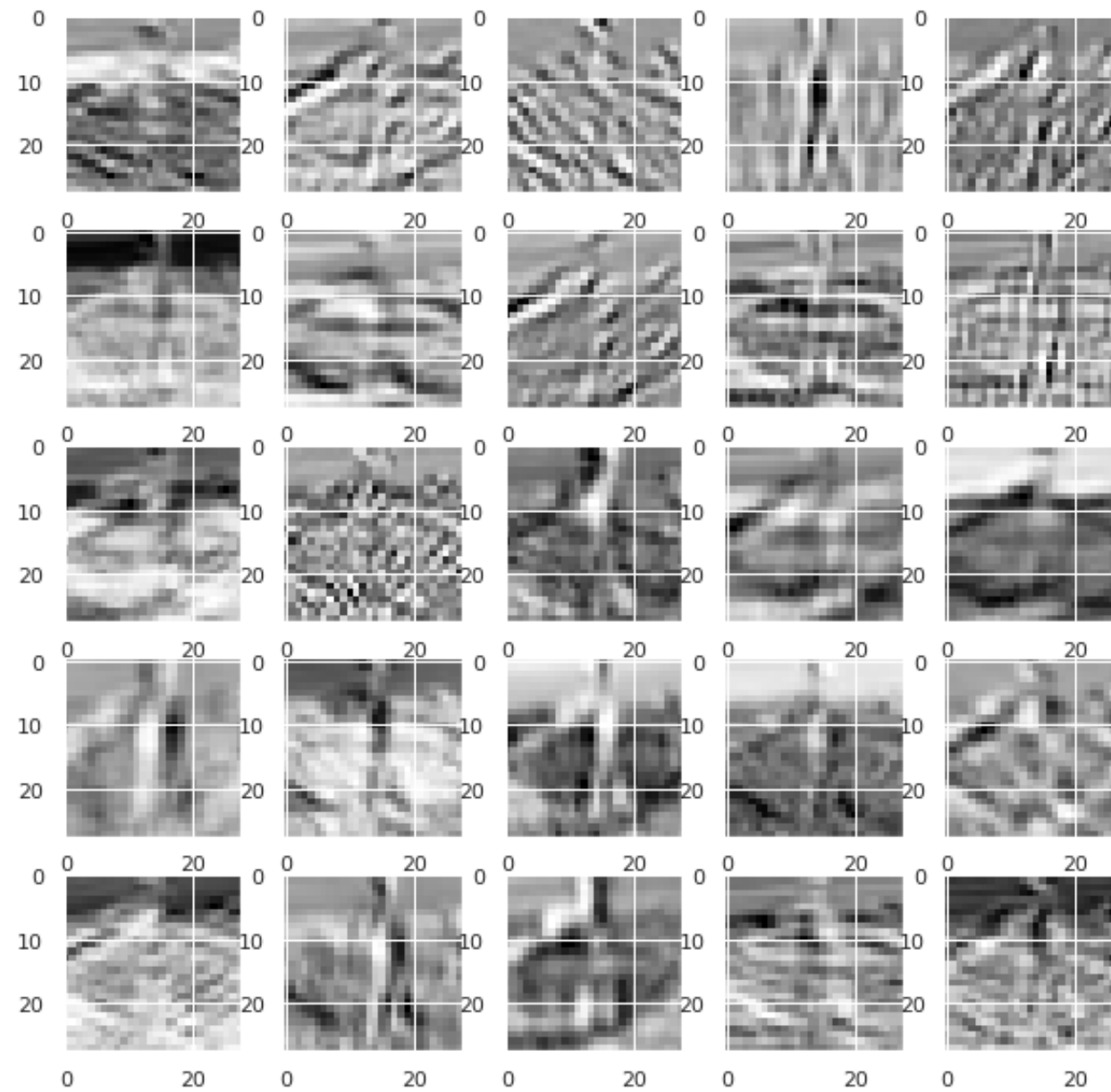
# un-normalize it
inpt = inpt/2+.5
plt.imshow(inpt[0].numpy().transpose(1,2,0), interpolation='nearest')
plt.title('original image')
plt.show()

# bring network C of the gpu to the cpu
nets[2].to('cpu')
# get the output of the first layer after passing the image in
output = nets[2].conv(inpt)
```

```
# Print its shape
print('output of conv layer shape', output.shape)
fig, ax = plt.subplots(5,5, figsize=(10,10))
for i in range(output.shape[1]):
    ax[i//5, i%5].imshow(output[0,i].detach().numpy(),interpolation='nearest', cmap=plt.get_cmap('Greys'))
plt.show()
```



```
output of conv layer shape torch.Size([1, 25, 28, 28])
```



## ▼ Part 5: Your Turn - NetD (20 pts)

Now it's your turn to design a neural network architecture! Your goal is to achieve 75% on the CIFAR10 test set that we loaded above. You can get their by

1. Changing hyperparameters such as batch size, the number of epochs, learning rate, optimizer, etc.
2. Adding additional convolutional and linear layers.
3. Using different training methodologies such as using [Batch Normalization](#) and Drop out (I really recommend looking into these :))

Create your own neural network architecture that has **at least 2 convolution layers** and **at least 2 fully connected layers**.

For your convenience, I have reproduced some of the code we used above below so that you can train independently of the rest. I removed the comments to make it a bit shorter. If you want to change hyperparameters on training or the optimizer, I suggest making a copy of the `train` function below.

*Some notes:*

- Every convolution operation should be followed by a pooling operation.
- Every linear layer and convolution layer should have an activation function. However, it is common to not have one on the very last layer to the 10 outputs so you do not need one there.
- Start with `NetC` and develop on top of it.
- It is not required, but it might help to make your code more modular by adding parameters to the constructor to specify numbers that contain details about the architecture. Then you can automate trying many different settings of these hyper-parameters. If you do add these parameters, modify the cell later that produces the training plots to pass in the appropriate parameter values.
- It's pretty tricky to get dimensions right in Torch. Use the `shape` command often to understand what the sizes are. For example, `print(x.shape)`. The first dimension is generally the batch size, followed by the number of channels, and then the image size. You can use the `.view` command to resize the tensor.
- Finally, it's good practice to keep the number of output channels and the dimensions of inner layers in powers of 2 (i.e. 4, 16, 32, 64, ...). This speeds up training time on the GPU.
- One last piece of advice: don't be impatient. Sometimes you won't get the result you want unless you train for 50 epochs or so.

In practice, you would create a separate validation set and build a model on that. However in this exercise we will cheat a bit to keep the complexity down. The number of hyperparameters to tune in the last exercise combined with the slow training times will hopefully give you a taste of how difficult it is to construct good performing networks. It should be emphasized that the networks we constructed are tiny; typical networks have dozens of layers, each with hyperparameters to tune.

**Grading:** I will be running your code - to ensure it gets the necessary accuracy. Your grade on this will be in terms of a percentage relative to 75%. I.e. if your final accuracy is 60%, then you will earn  $60/75 \times 20 = 16$

```
##time
BATCH_SIZE = 85
```

```
train_loader = torch.utils.data.DataLoader(trainset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
test_loader = torch.utils.data.DataLoader(testset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
```

```
class NetD(nn.Module):
    def __init__(self):
        super(NetD, self).__init__()
        #TODO
        self.conv_layer = nn.Sequential(

            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d((2, 2)),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d((2, 2)),
            nn.Dropout2d(p = 0.05))

        self.fc_layer = nn.Sequential(
            nn.Dropout(p = 0.05),
            nn.Linear(4096, 1024),
            nn.ReLU(),
            nn.Linear(1024, 100),
            nn.ReLU(),
            nn.Dropout(p = 0.05),
            nn.Linear(100, 10))

    def forward(self, x):
        #TODO
        x = self.conv_layer(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layer(x)
        return x
```

```
net = NetD()
histories = []
```

```
print(f'==== Training ====')
train_history, test_history = train(net, train_loader, test_loader,
                                    num_epochs=50,
                                    learning_rate=.001,
                                    compute_accs=True)
```

```
histories.append({
    'name': 'NetD',
    'train_accs': train_history,
    'test_accs': test_history
})
```



```
})  
plot_history(histories)
```

```
==== Training ====  
Epoch [1/50], Step [100/589], Loss: 1.4262  
Epoch [1/50], Step [200/589], Loss: 1.2879  
Epoch [1/50], Step [300/589], Loss: 1.1523  
Epoch [1/50], Step [400/589], Loss: 1.0765  
Epoch [1/50], Step [500/589], Loss: 1.1655  
Epoch [1/50], Train Accuracy 63.15%, Test Accuracy 61.26%  
Epoch [2/50], Step [100/589], Loss: 0.9974  
Epoch [2/50], Step [200/589], Loss: 0.8625  
Epoch [2/50], Step [300/589], Loss: 1.0882  
Epoch [2/50], Step [400/589], Loss: 0.9776  
Epoch [2/50], Step [500/589], Loss: 0.9885  
Epoch [2/50], Train Accuracy 71.08%, Test Accuracy 67.81%  
Epoch [3/50], Step [100/589], Loss: 0.9711  
Epoch [3/50], Step [200/589], Loss: 0.7214  
Epoch [3/50], Step [300/589], Loss: 0.9257  
Epoch [3/50], Step [400/589], Loss: 0.8288  
Epoch [3/50], Step [500/589], Loss: 0.5454  
Epoch [3/50], Train Accuracy 75.93%, Test Accuracy 70.72%  
Epoch [4/50], Step [100/589], Loss: 0.7370  
Epoch [4/50], Step [200/589], Loss: 0.6517  
Epoch [4/50], Step [300/589], Loss: 0.8317  
Epoch [4/50], Step [400/589], Loss: 0.6955  
Epoch [4/50], Step [500/589], Loss: 0.7056  
Epoch [4/50], Train Accuracy 80.08%, Test Accuracy 72.10%  
Epoch [5/50], Step [100/589], Loss: 0.4838  
Epoch [5/50], Step [200/589], Loss: 0.5948  
Epoch [5/50], Step [300/589], Loss: 0.5172  
Epoch [5/50], Step [400/589], Loss: 0.6006  
Epoch [5/50], Step [500/589], Loss: 0.5588  
Epoch [5/50], Train Accuracy 83.44%, Test Accuracy 73.62%  
Epoch [6/50], Step [100/589], Loss: 0.6023  
Epoch [6/50], Step [200/589], Loss: 0.4468  
Epoch [6/50], Step [300/589], Loss: 0.5241  
Epoch [6/50], Step [400/589], Loss: 0.4905  
Epoch [6/50], Step [500/589], Loss: 0.4944  
Epoch [6/50], Train Accuracy 88.09%, Test Accuracy 75.04%  
Epoch [7/50], Step [100/589], Loss: 0.2948  
Epoch [7/50], Step [200/589], Loss: 0.4431  
Epoch [7/50], Step [300/589], Loss: 0.4476  
Epoch [7/50], Step [400/589], Loss: 0.4130  
Epoch [7/50], Step [500/589], Loss: 0.3866  
Epoch [7/50], Train Accuracy 89.42%, Test Accuracy 74.55%  
Epoch [8/50], Step [100/589], Loss: 0.2987  
Epoch [8/50], Step [200/589], Loss: 0.2906  
Epoch [8/50], Step [300/589], Loss: 0.2331  
Epoch [8/50], Step [400/589], Loss: 0.4527  
Epoch [8/50], Step [500/589], Loss: 0.3357  
Epoch [8/50], Train Accuracy 91.12%, Test Accuracy 74.26%  
Epoch [9/50], Step [100/589], Loss: 0.0874  
Epoch [9/50], Step [200/589], Loss: 0.2501  
Epoch [9/50], Step [300/589], Loss: 0.2272
```

```
Epoch [9/50], Step [300/589], Loss: 0.3372
Epoch [9/50], Step [400/589], Loss: 0.2145
Epoch [9/50], Step [500/589], Loss: 0.2175
Epoch [9/50], Train Accuracy 92.59%, Test Accuracy 74.64%
Epoch [10/50], Step [100/589], Loss: 0.1999
Epoch [10/50], Step [200/589], Loss: 0.2101
Epoch [10/50], Step [300/589], Loss: 0.1919
Epoch [10/50], Step [400/589], Loss: 0.2601
Epoch [10/50], Step [500/589], Loss: 0.2268
Epoch [10/50], Train Accuracy 94.16%, Test Accuracy 74.02%
Epoch [11/50], Step [100/589], Loss: 0.0964
Epoch [11/50], Step [200/589], Loss: 0.1999
Epoch [11/50], Step [300/589], Loss: 0.1464
Epoch [11/50], Step [400/589], Loss: 0.1199
Epoch [11/50], Step [500/589], Loss: 0.1402
Epoch [11/50], Train Accuracy 95.85%, Test Accuracy 74.67%
Epoch [12/50], Step [100/589], Loss: 0.2615
Epoch [12/50], Step [200/589], Loss: 0.1301
Epoch [12/50], Step [300/589], Loss: 0.1588
Epoch [12/50], Step [400/589], Loss: 0.1171
Epoch [12/50], Step [500/589], Loss: 0.1158
Epoch [12/50], Train Accuracy 96.04%, Test Accuracy 74.14%
Epoch [13/50], Step [100/589], Loss: 0.1933
Epoch [13/50], Step [200/589], Loss: 0.1160
Epoch [13/50], Step [300/589], Loss: 0.1506
Epoch [13/50], Step [400/589], Loss: 0.1678
Epoch [13/50], Step [500/589], Loss: 0.1721
Epoch [13/50], Train Accuracy 97.26%, Test Accuracy 75.08%
Epoch [14/50], Step [100/589], Loss: 0.0508
Epoch [14/50], Step [200/589], Loss: 0.0683
Epoch [14/50], Step [300/589], Loss: 0.1254
Epoch [14/50], Step [400/589], Loss: 0.0549
Epoch [14/50], Step [500/589], Loss: 0.1142
Epoch [14/50], Train Accuracy 97.36%, Test Accuracy 75.04%
Epoch [15/50], Step [100/589], Loss: 0.1557
Epoch [15/50], Step [200/589], Loss: 0.0926
Epoch [15/50], Step [300/589], Loss: 0.1212
Epoch [15/50], Step [400/589], Loss: 0.0528
Epoch [15/50], Step [500/589], Loss: 0.1041
Epoch [15/50], Train Accuracy 97.72%, Test Accuracy 74.93%
Epoch [16/50], Step [100/589], Loss: 0.0714
Epoch [16/50], Step [200/589], Loss: 0.0833
Epoch [16/50], Step [300/589], Loss: 0.1278
Epoch [16/50], Step [400/589], Loss: 0.0342
Epoch [16/50], Step [500/589], Loss: 0.0723
Epoch [16/50], Train Accuracy 97.90%, Test Accuracy 74.40%
Epoch [17/50], Step [100/589], Loss: 0.0733
Epoch [17/50], Step [200/589], Loss: 0.0713
Epoch [17/50], Step [300/589], Loss: 0.1354
Epoch [17/50], Step [400/589], Loss: 0.0785
Epoch [17/50], Step [500/589], Loss: 0.0581
Epoch [17/50], Train Accuracy 98.16%, Test Accuracy 74.56%
Epoch [18/50], Step [100/589], Loss: 0.1630
Epoch [18/50], Step [200/589], Loss: 0.0888
Epoch [18/50], Step [300/589], Loss: 0.1027
```

```
Epoch [18/50], Step [400/589], Loss: 0.0928
Epoch [18/50], Step [500/589], Loss: 0.1252
Epoch [18/50], Train Accuracy 98.03%, Test Accuracy 74.61%
Epoch [19/50], Step [100/589], Loss: 0.0832
Epoch [19/50], Step [200/589], Loss: 0.0517
Epoch [19/50], Step [300/589], Loss: 0.0355
Epoch [19/50], Step [400/589], Loss: 0.0144
Epoch [19/50], Step [500/589], Loss: 0.1394
Epoch [19/50], Train Accuracy 97.91%, Test Accuracy 74.26%
Epoch [20/50], Step [100/589], Loss: 0.0200
Epoch [20/50], Step [200/589], Loss: 0.0216
Epoch [20/50], Step [300/589], Loss: 0.0251
Epoch [20/50], Step [400/589], Loss: 0.0188
Epoch [20/50], Step [500/589], Loss: 0.1047
Epoch [20/50], Train Accuracy 98.27%, Test Accuracy 74.71%
Epoch [21/50], Step [100/589], Loss: 0.0949
Epoch [21/50], Step [200/589], Loss: 0.1057
Epoch [21/50], Step [300/589], Loss: 0.0213
Epoch [21/50], Step [400/589], Loss: 0.0401
Epoch [21/50], Step [500/589], Loss: 0.0661
Epoch [21/50], Train Accuracy 98.68%, Test Accuracy 74.65%
Epoch [22/50], Step [100/589], Loss: 0.0393
Epoch [22/50], Step [200/589], Loss: 0.0320
Epoch [22/50], Step [300/589], Loss: 0.1101
Epoch [22/50], Step [400/589], Loss: 0.0478
Epoch [22/50], Step [500/589], Loss: 0.0965
Epoch [22/50], Train Accuracy 98.43%, Test Accuracy 74.07%
Epoch [23/50], Step [100/589], Loss: 0.0154
Epoch [23/50], Step [200/589], Loss: 0.0964
Epoch [23/50], Step [300/589], Loss: 0.0056
Epoch [23/50], Step [400/589], Loss: 0.0309
Epoch [23/50], Step [500/589], Loss: 0.0353
Epoch [23/50], Train Accuracy 98.53%, Test Accuracy 75.02%
Epoch [24/50], Step [100/589], Loss: 0.1645
Epoch [24/50], Step [200/589], Loss: 0.0877
Epoch [24/50], Step [300/589], Loss: 0.0151
Epoch [24/50], Step [400/589], Loss: 0.0658
Epoch [24/50], Step [500/589], Loss: 0.0949
Epoch [24/50], Train Accuracy 98.46%, Test Accuracy 74.30%
Epoch [25/50], Step [100/589], Loss: 0.0112
Epoch [25/50], Step [200/589], Loss: 0.0477
Epoch [25/50], Step [300/589], Loss: 0.1236
Epoch [25/50], Step [400/589], Loss: 0.1366
Epoch [25/50], Step [500/589], Loss: 0.0896
Epoch [25/50], Train Accuracy 98.73%, Test Accuracy 74.81%
Epoch [26/50], Step [100/589], Loss: 0.0644
Epoch [26/50], Step [200/589], Loss: 0.0157
Epoch [26/50], Step [300/589], Loss: 0.0278
Epoch [26/50], Step [400/589], Loss: 0.0456
Epoch [26/50], Step [500/589], Loss: 0.0129
Epoch [26/50], Train Accuracy 98.75%, Test Accuracy 74.41%
Epoch [27/50], Step [100/589], Loss: 0.0125
Epoch [27/50], Step [200/589], Loss: 0.0819
Epoch [27/50], Step [300/589], Loss: 0.1114
Epoch [27/50], Step [400/589], Loss: 0.0108
```

```
Epoch [27/50], Step [500/589], Loss: 0.0138
Epoch [27/50], Train Accuracy 98.79%, Test Accuracy 74.29%
Epoch [28/50], Step [100/589], Loss: 0.0215
Epoch [28/50], Step [200/589], Loss: 0.0732
Epoch [28/50], Step [300/589], Loss: 0.0454
Epoch [28/50], Step [400/589], Loss: 0.1269
Epoch [28/50], Step [500/589], Loss: 0.0335
Epoch [28/50], Train Accuracy 98.61%, Test Accuracy 74.68%
Epoch [29/50], Step [100/589], Loss: 0.0370
Epoch [29/50], Step [200/589], Loss: 0.0178
Epoch [29/50], Step [300/589], Loss: 0.0510
Epoch [29/50], Step [400/589], Loss: 0.0509
Epoch [29/50], Step [500/589], Loss: 0.0486
Epoch [29/50], Train Accuracy 98.94%, Test Accuracy 74.46%
Epoch [30/50], Step [100/589], Loss: 0.1263
Epoch [30/50], Step [200/589], Loss: 0.0691
Epoch [30/50], Step [300/589], Loss: 0.0481
Epoch [30/50], Step [400/589], Loss: 0.0307
Epoch [30/50], Step [500/589], Loss: 0.1110
Epoch [30/50], Train Accuracy 98.76%, Test Accuracy 74.37%
Epoch [31/50], Step [100/589], Loss: 0.0243
Epoch [31/50], Step [200/589], Loss: 0.0299
Epoch [31/50], Step [300/589], Loss: 0.1304
Epoch [31/50], Step [400/589], Loss: 0.0439
Epoch [31/50], Step [500/589], Loss: 0.0888
Epoch [31/50], Train Accuracy 98.55%, Test Accuracy 74.05%
Epoch [32/50], Step [100/589], Loss: 0.0523
Epoch [32/50], Step [200/589], Loss: 0.0214
Epoch [32/50], Step [300/589], Loss: 0.0210
Epoch [32/50], Step [400/589], Loss: 0.0080
Epoch [32/50], Step [500/589], Loss: 0.0183
Epoch [32/50], Train Accuracy 99.04%, Test Accuracy 74.41%
Epoch [33/50], Step [100/589], Loss: 0.0339
Epoch [33/50], Step [200/589], Loss: 0.0545
Epoch [33/50], Step [300/589], Loss: 0.0205
Epoch [33/50], Step [400/589], Loss: 0.0082
Epoch [33/50], Step [500/589], Loss: 0.1062
Epoch [33/50], Train Accuracy 98.63%, Test Accuracy 74.39%
Epoch [34/50], Step [100/589], Loss: 0.0247
Epoch [34/50], Step [200/589], Loss: 0.0180
Epoch [34/50], Step [300/589], Loss: 0.0113
Epoch [34/50], Step [400/589], Loss: 0.0028
Epoch [34/50], Step [500/589], Loss: 0.0580
Epoch [34/50], Train Accuracy 99.07%, Test Accuracy 74.52%
Epoch [35/50], Step [100/589], Loss: 0.0089
Epoch [35/50], Step [200/589], Loss: 0.0293
Epoch [35/50], Step [300/589], Loss: 0.0139
Epoch [35/50], Step [400/589], Loss: 0.0970
Epoch [35/50], Step [500/589], Loss: 0.0184
Epoch [35/50], Train Accuracy 99.10%, Test Accuracy 74.11%
Epoch [36/50], Step [100/589], Loss: 0.0060
Epoch [36/50], Step [200/589], Loss: 0.0215
Epoch [36/50], Step [300/589], Loss: 0.0227
Epoch [36/50], Step [400/589], Loss: 0.0263
```

```
Epoch [36/50], Step [500/589], Loss: 0.0478
Epoch [36/50], Train Accuracy 99.13%, Test Accuracy 74.34%
Epoch [37/50], Step [100/589], Loss: 0.0062
Epoch [37/50], Step [200/589], Loss: 0.0085
Epoch [37/50], Step [300/589], Loss: 0.0119
Epoch [37/50], Step [400/589], Loss: 0.0469
Epoch [37/50], Step [500/589], Loss: 0.0068
Epoch [37/50], Train Accuracy 99.08%, Test Accuracy 74.43%
Epoch [38/50], Step [100/589], Loss: 0.0110
Epoch [38/50], Step [200/589], Loss: 0.0059
Epoch [38/50], Step [300/589], Loss: 0.0274
Epoch [38/50], Step [400/589], Loss: 0.0017
Epoch [38/50], Step [500/589], Loss: 0.0039
Epoch [38/50], Train Accuracy 99.14%, Test Accuracy 74.28%
Epoch [39/50], Step [100/589], Loss: 0.0663
Epoch [39/50], Step [200/589], Loss: 0.0209
Epoch [39/50], Step [300/589], Loss: 0.0308
Epoch [39/50], Step [400/589], Loss: 0.0191
Epoch [39/50], Step [500/589], Loss: 0.0269
Epoch [39/50], Train Accuracy 99.07%, Test Accuracy 74.13%
Epoch [40/50], Step [100/589], Loss: 0.0762
Epoch [40/50], Step [200/589], Loss: 0.0562
Epoch [40/50], Step [300/589], Loss: 0.0143
Epoch [40/50], Step [400/589], Loss: 0.0357
Epoch [40/50], Step [500/589], Loss: 0.0519
Epoch [40/50], Train Accuracy 99.16%, Test Accuracy 74.62%
Epoch [41/50], Step [100/589], Loss: 0.0702
Epoch [41/50], Step [200/589], Loss: 0.0139
Epoch [41/50], Step [300/589], Loss: 0.0054
Epoch [41/50], Step [400/589], Loss: 0.0069
Epoch [41/50], Step [500/589], Loss: 0.0580
Epoch [41/50], Train Accuracy 99.12%, Test Accuracy 74.54%
Epoch [42/50], Step [100/589], Loss: 0.0200
Epoch [42/50], Step [200/589], Loss: 0.0989
Epoch [42/50], Step [300/589], Loss: 0.0438
Epoch [42/50], Step [400/589], Loss: 0.0892
Epoch [42/50], Step [500/589], Loss: 0.0514
Epoch [42/50], Train Accuracy 99.25%, Test Accuracy 74.58%
Epoch [43/50], Step [100/589], Loss: 0.0028
Epoch [43/50], Step [200/589], Loss: 0.0203
Epoch [43/50], Step [300/589], Loss: 0.0116
Epoch [43/50], Step [400/589], Loss: 0.0314
Epoch [43/50], Step [500/589], Loss: 0.0085
Epoch [43/50], Train Accuracy 98.91%, Test Accuracy 74.32%
Epoch [44/50], Step [100/589], Loss: 0.0197
Epoch [44/50], Step [200/589], Loss: 0.0125
Epoch [44/50], Step [300/589], Loss: 0.0527
Epoch [44/50], Step [400/589], Loss: 0.0318
Epoch [44/50], Step [500/589], Loss: 0.0091
Epoch [44/50], Train Accuracy 99.11%, Test Accuracy 74.35%
Epoch [45/50], Step [100/589], Loss: 0.0299
Epoch [45/50], Step [200/589], Loss: 0.0380
Epoch [45/50], Step [300/589], Loss: 0.0323
Epoch [45/50], Step [400/589], Loss: 0.0005
Epoch [45/50], Step [500/589], Loss: 0.0072
```

```
Epoch [45/50], Train Accuracy 99.23%, Test Accuracy 74.69%
Epoch [46/50], Step [100/589], Loss: 0.0092
Epoch [46/50], Step [200/589], Loss: 0.0090
Epoch [46/50], Step [300/589], Loss: 0.0035
Epoch [46/50], Step [400/589], Loss: 0.0239
Epoch [46/50], Step [500/589], Loss: 0.0038
Epoch [46/50], Train Accuracy 99.43%, Test Accuracy 74.13%
Epoch [47/50], Step [100/589], Loss: 0.0121
Epoch [47/50], Step [200/589], Loss: 0.0012
Epoch [47/50], Step [300/589], Loss: 0.0025
Epoch [47/50], Step [400/589], Loss: 0.0221
Epoch [47/50], Step [500/589], Loss: 0.0064
Epoch [47/50], Train Accuracy 99.30%, Test Accuracy 74.50%
Epoch [48/50], Step [100/589], Loss: 0.0018
Epoch [48/50], Step [200/589], Loss: 0.0120
Epoch [48/50], Step [300/589], Loss: 0.0075
Epoch [48/50], Step [400/589], Loss: 0.0448
Epoch [48/50], Step [500/589], Loss: 0.0063
Epoch [48/50], Train Accuracy 99.21%, Test Accuracy 74.36%
Epoch [49/50], Step [100/589], Loss: 0.0467
Epoch [49/50], Step [200/589], Loss: 0.0278
Epoch [49/50], Step [300/589], Loss: 0.0083
Epoch [49/50], Step [400/589], Loss: 0.0475
Epoch [49/50], Step [500/589], Loss: 0.0024
Epoch [49/50], Train Accuracy 99.45%, Test Accuracy 74.67%
Epoch [50/50], Step [100/589], Loss: 0.0018
Epoch [50/50], Step [200/589], Loss: 0.0091
Epoch [50/50], Step [300/589], Loss: 0.0100
Epoch [50/50], Step [400/589], Loss: 0.0174
Epoch [50/50], Step [500/589], Loss: 0.0900
Epoch [50/50], Train Accuracy 99.27%, Test Accuracy 74.49%
```

