eyi9nsoqp

March 14, 2024

1 Assignment 3 - AI535 - Deep Learning - Loc Nguyen - 933628639

[1]:
```python
# AI535 - Deep Learning Assgignment 3
# Loc Nguyen
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
# Prepare CIFAR-10 dataset
# Also implement the
# "We follow the simple data augmentation in [24] for training: 4 pixels are padded on each side,
# and a 32×32 crop is randomly sampled from the padded
# image or its horizontal flip" part
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

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

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False, num_workers=2)
```

```


classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz

100%|      | 170498071/170498071 [00:05<00:00, 29172997.10it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

2 Helper function to show some image

[2]:

```
# Helper function to show some image
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def matplotlib_imshow(img, one_channel=False):
    if one_channel:
        img = img.mean(dim=0)
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    if one_channel:
        plt.imshow(npimg, cmap="Greys")
    else:
        plt.imshow(np.transpose(npimg, (1, 2, 0)))
```

3 1 - Create a Resnet-14 that classifies the CIFAR-10 dataset

[3]:

```
# 1 - Create a Resnet-14 that classifies the CIFAR-10 dataset
# Define BasicBlock for ResNet
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,␣
 ↪padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1,␣
 ↪padding=1, bias=False)
```

```
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

# Define ResNet architecture
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 16
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(block, 16, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 32, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 64, num_blocks[2], stride=2)
        self.linear = nn.Linear(64, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = F.avg_pool2d(out, 8) # Global average pooling
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return F.log_softmax(out, dim=1)
```

```
# Define ResNet model
def ResNet14():
    return ResNet(BasicBlock, [2, 2, 2])
```

4 2 - Setting up tensorboard writer

[4]:

```
# Setting up tensorboard writer
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter('runs/cifar_a3_1')
```

5 3 - Write some training images to tensorboard

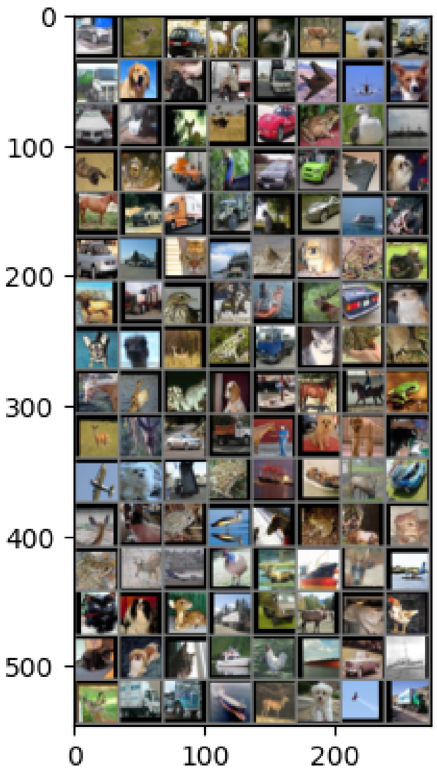
[5]:

```
# Get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

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

# show images
matplotlib_imshow(img_grid, one_channel=False)

# 3 Write some training image to tensorboard
writer.add_image('some_training_cifar_10_images', img_grid)
writer.close()
```



6 INSPECT THE MODEL STRUCTURE WITH TENSORBOARD

[6]:

```
# INSPECT THE MODEL STRUCTURE WITH TENSORBOARD

# Create an object of ResNet14 to write it to TensorBoard
# This object is purely created to be looked at
# Not for computation
net_structure = ResNet14()
writer.add_graph(net_structure, images)
writer.close()
```

7 Some helper functions from the tutorial

[7]:

```
# Some helper functions from the tutorial

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


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

8 3, 4 - Function to Train ResNet14 for 30 epochs

9 Takes a learning_rate as a parameter

10 Also write data to tensor board to visualize based on

11 what learning_rate is being used.

[8]:

```
def train_ResNet14(learning_rate):
    # Instantiate a ResNet14() object
    # also incorporate GPU computation if there's a GPU
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    # Instantiate the model
    net = ResNet14().to(device)
    # Define loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    # warm up with lr 0.1, and then change learning when milestones reached
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9,␣
↪weight_decay=0.0001)

    # Train the model
    for epoch in range(30):
        net.train()
        for batch_idx, (inputs, targets) in enumerate(trainloader):
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

        # Print training statistics
        print(f"Epoch {epoch+1}, Loss: {loss.item()}")
        # log training loss vs. epoch
        writer.add_scalar('training_loss_' + str(learning_rate), loss.item(),␣
↪epoch + 1)

        # Test the model per epoch
        net.eval()
        correct = 0
        total = 0
        test_loss = 0.0
        with torch.no_grad():
            for inputs, targets in testloader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = net(inputs)
                loss = criterion(outputs, targets)
                test_loss += loss.item()
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

        accuracy = 100. * correct / total
        avg_test_loss = test_loss / len(testloader)
        print(f"Test Accuracy: {accuracy}%")
        print(f"Average Test Loss: {avg_test_loss}")
        # Log testing accuracy and loss to TensorBoard
        writer.add_scalar('testing_accuracy_' + str(learning_rate), accuracy,␣
↪epoch + 1)
        writer.add_scalar('testing_loss' + str(learning_rate), avg_test_loss,␣
↪epoch + 1)
```

12 5 - Tuning learning rate

13 Trying learning_rate = 0.1, 0.01, 0.001, 0.0001

[9]:

```
train_ResNet14(0.1)
train_ResNet14(0.01)
train_ResNet14(0.001)
train_ResNet14(0.0001)
```

```
Epoch 1, Loss: 1.1886175870895386
Test Accuracy: 43.97%
Average Test Loss: 1.5854694199562074
Epoch 2, Loss: 0.7863050699234009
Test Accuracy: 59.29%
Average Test Loss: 1.2123463010787965
Epoch 3, Loss: 0.8136318325996399
Test Accuracy: 62.76%
Average Test Loss: 1.1634840834140778
Epoch 4, Loss: 0.5557407140731812
Test Accuracy: 73.02%
Average Test Loss: 0.7737915372848511
Epoch 5, Loss: 0.6067843437194824
Test Accuracy: 71.4%
Average Test Loss: 0.8438060879707336
Epoch 6, Loss: 0.7810858488082886
Test Accuracy: 75.62%
Average Test Loss: 0.7503524857759476
Epoch 7, Loss: 0.5064166188240051
Test Accuracy: 75.96%
Average Test Loss: 0.7236337608098984
Epoch 8, Loss: 0.5752705931663513
Test Accuracy: 74.19%
Average Test Loss: 0.8225128453969955
Epoch 9, Loss: 0.6300138235092163
Test Accuracy: 78.97%
Average Test Loss: 0.6196774181723594
Epoch 10, Loss: 0.4758372902870178
Test Accuracy: 78.41%
Average Test Loss: 0.6367863896489143
Epoch 11, Loss: 0.40627115964889526
Test Accuracy: 80.42%
Average Test Loss: 0.5885299181938172
Epoch 12, Loss: 0.5141482353210449
Test Accuracy: 80.61%
Average Test Loss: 0.5755460259318351
Epoch 13, Loss: 0.49615398049354553
Test Accuracy: 81.46%
```

```
Average Test Loss: 0.5735385650396347
Epoch 14, Loss: 0.4328024387359619
Test Accuracy: 82.34%
Average Test Loss: 0.544113977253437
Epoch 15, Loss: 0.6146947741508484
Test Accuracy: 82.31%
Average Test Loss: 0.5461732184886933
Epoch 16, Loss: 0.3902626931667328
Test Accuracy: 77.93%
Average Test Loss: 0.6561117461323738
Epoch 17, Loss: 0.5060499310493469
Test Accuracy: 81.86%
Average Test Loss: 0.5367414692044258
Epoch 18, Loss: 0.3713814616203308
Test Accuracy: 79.0%
Average Test Loss: 0.6591809687018394
Epoch 19, Loss: 0.22669751942157745
Test Accuracy: 80.21%
Average Test Loss: 0.6205789956450463
Epoch 20, Loss: 0.3457683026790619
Test Accuracy: 82.97%
Average Test Loss: 0.5362190590798854
Epoch 21, Loss: 0.2945208251476288
Test Accuracy: 83.08%
Average Test Loss: 0.4947455081343651
Epoch 22, Loss: 0.30726388096809387
Test Accuracy: 84.22%
Average Test Loss: 0.4666206979751587
Epoch 23, Loss: 0.4715932309627533
Test Accuracy: 84.64%
Average Test Loss: 0.4793688902258873
Epoch 24, Loss: 0.4534260332584381
Test Accuracy: 80.61%
Average Test Loss: 0.622709549665451
Epoch 25, Loss: 0.29083937406539917
Test Accuracy: 83.25%
Average Test Loss: 0.524231747686863
Epoch 26, Loss: 0.4054945409297943
Test Accuracy: 84.18%
Average Test Loss: 0.4812477348744869
Epoch 27, Loss: 0.3996894657611847
Test Accuracy: 81.37%
Average Test Loss: 0.6292722010612488
Epoch 28, Loss: 0.30149880051612854
Test Accuracy: 82.26%
Average Test Loss: 0.5603784194588661
Epoch 29, Loss: 0.3123488426208496
Test Accuracy: 84.2%
```

```
Average Test Loss: 0.49728916376829146
Epoch 30, Loss: 0.447988897562027
Test Accuracy: 83.84%
Average Test Loss: 0.5031423181295395
Epoch 1, Loss: 1.1388682126998901
Test Accuracy: 53.91%
Average Test Loss: 1.2535140585899354
Epoch 2, Loss: 0.9001132249832153
Test Accuracy: 55.65%
Average Test Loss: 1.2722173964977264
Epoch 3, Loss: 0.9855661392211914
Test Accuracy: 62.74%
Average Test Loss: 1.096794610619545
Epoch 4, Loss: 0.7722891569137573
Test Accuracy: 66.69%
Average Test Loss: 1.0282330870628358
Epoch 5, Loss: 0.6730998754501343
Test Accuracy: 66.59%
Average Test Loss: 1.0496321123838426
Epoch 6, Loss: 0.513959527015686
Test Accuracy: 72.97%
Average Test Loss: 0.8074255383014679
Epoch 7, Loss: 0.6575276851654053
Test Accuracy: 77.49%
Average Test Loss: 0.6699701943993568
Epoch 8, Loss: 0.5427938103675842
Test Accuracy: 75.94%
Average Test Loss: 0.7135250303149223
Epoch 9, Loss: 0.3630017936229706
Test Accuracy: 76.8%
Average Test Loss: 0.6886607310175896
Epoch 10, Loss: 0.5423012375831604
Test Accuracy: 80.2%
Average Test Loss: 0.5908241873979568
Epoch 11, Loss: 0.506608784198761
Test Accuracy: 79.39%
Average Test Loss: 0.6170845419168473
Epoch 12, Loss: 0.4863855242729187
Test Accuracy: 77.03%
Average Test Loss: 0.7209270960092544
Epoch 13, Loss: 0.34954118728637695
Test Accuracy: 75.39%
Average Test Loss: 0.7342845922708512
Epoch 14, Loss: 0.5532934665679932
Test Accuracy: 79.94%
Average Test Loss: 0.5975188341736793
Epoch 15, Loss: 0.353233277797699
Test Accuracy: 79.38%
```

```
Average Test Loss: 0.6437800770998001
Epoch 16, Loss: 0.49855464696884155
Test Accuracy: 80.2%
Average Test Loss: 0.5904763635993003
Epoch 17, Loss: 0.42818403244018555
Test Accuracy: 81.69%
Average Test Loss: 0.5686482821404933
Epoch 18, Loss: 0.5564732551574707
Test Accuracy: 82.1%
Average Test Loss: 0.5338846370577812
Epoch 19, Loss: 0.42931514978408813
Test Accuracy: 82.08%
Average Test Loss: 0.5415171700716018
Epoch 20, Loss: 0.4648149013519287
Test Accuracy: 84.11%
Average Test Loss: 0.46906937956809996
Epoch 21, Loss: 0.45557159185409546
Test Accuracy: 82.63%
Average Test Loss: 0.5182003584504128
Epoch 22, Loss: 0.3522607684135437
Test Accuracy: 82.5%
Average Test Loss: 0.560834299325943
Epoch 23, Loss: 0.4096686840057373
Test Accuracy: 83.31%
Average Test Loss: 0.5202001179754734
Epoch 24, Loss: 0.3935508131980896
Test Accuracy: 84.95%
Average Test Loss: 0.46900354579091075
Epoch 25, Loss: 0.3174384832382202
Test Accuracy: 84.98%
Average Test Loss: 0.46126652538776397
Epoch 26, Loss: 0.4824884533882141
Test Accuracy: 82.41%
Average Test Loss: 0.5589759320020675
Epoch 27, Loss: 0.2221202403306961
Test Accuracy: 84.43%
Average Test Loss: 0.47056042358279226
Epoch 28, Loss: 0.42984724044799805
Test Accuracy: 85.07%
Average Test Loss: 0.465376081764698
Epoch 29, Loss: 0.27670204639434814
Test Accuracy: 81.44%
Average Test Loss: 0.5852805680036545
Epoch 30, Loss: 0.37063318490982056
Test Accuracy: 84.74%
Average Test Loss: 0.4729249720275402
Epoch 1, Loss: 1.7261587381362915
Test Accuracy: 36.24%
```

```
Average Test Loss: 1.7021585381031037
Epoch 2, Loss: 1.3484593629837036
Test Accuracy: 44.76%
Average Test Loss: 1.4867102587223053
Epoch 3, Loss: 1.2240654230117798
Test Accuracy: 45.84%
Average Test Loss: 1.5059030771255493
Epoch 4, Loss: 1.3015284538269043
Test Accuracy: 55.75%
Average Test Loss: 1.2496578353643417
Epoch 5, Loss: 1.105072021484375
Test Accuracy: 58.1%
Average Test Loss: 1.1756844037771226
Epoch 6, Loss: 1.1451513767242432
Test Accuracy: 60.65%
Average Test Loss: 1.0983743023872377
Epoch 7, Loss: 0.9482304453849792
Test Accuracy: 64.52%
Average Test Loss: 0.99580431163311
Epoch 8, Loss: 1.0891282558441162
Test Accuracy: 65.1%
Average Test Loss: 0.9768343853950501
Epoch 9, Loss: 0.8790653347969055
Test Accuracy: 66.11%
Average Test Loss: 0.9462442004680633
Epoch 10, Loss: 0.7934421896934509
Test Accuracy: 66.76%
Average Test Loss: 0.9328238672018051
Epoch 11, Loss: 0.8722626566886902
Test Accuracy: 66.56%
Average Test Loss: 0.9498969542980195
Epoch 12, Loss: 0.8012292981147766
Test Accuracy: 68.06%
Average Test Loss: 0.8985247397422791
Epoch 13, Loss: 1.3183963298797607
Test Accuracy: 68.97%
Average Test Loss: 0.8784819930791855
Epoch 14, Loss: 0.8162153363227844
Test Accuracy: 70.57%
Average Test Loss: 0.8422540807723999
Epoch 15, Loss: 0.8055901527404785
Test Accuracy: 70.48%
Average Test Loss: 0.8612696689367294
Epoch 16, Loss: 0.7044140100479126
Test Accuracy: 70.32%
Average Test Loss: 0.8555798882246017
Epoch 17, Loss: 0.6008330583572388
Test Accuracy: 72.59%
```

```
Average Test Loss: 0.8106142091751098
Epoch 18, Loss: 0.7391735315322876
Test Accuracy: 72.9%
Average Test Loss: 0.7985450536012649
Epoch 19, Loss: 0.7541524171829224
Test Accuracy: 72.55%
Average Test Loss: 0.8211517584323883
Epoch 20, Loss: 0.5841436386108398
Test Accuracy: 73.52%
Average Test Loss: 0.7766760295629501
Epoch 21, Loss: 0.5629048943519592
Test Accuracy: 75.22%
Average Test Loss: 0.7237177565693855
Epoch 22, Loss: 0.7351367473602295
Test Accuracy: 76.01%
Average Test Loss: 0.7173412787914276
Epoch 23, Loss: 0.4872182309627533
Test Accuracy: 77.36%
Average Test Loss: 0.6780773302912713
Epoch 24, Loss: 0.5911656618118286
Test Accuracy: 76.18%
Average Test Loss: 0.7053483265638352
Epoch 25, Loss: 0.626050591468811
Test Accuracy: 77.16%
Average Test Loss: 0.677828688621521
Epoch 26, Loss: 0.4642335772514343
Test Accuracy: 78.45%
Average Test Loss: 0.6353330445289612
Epoch 27, Loss: 0.7219212055206299
Test Accuracy: 79.04%
Average Test Loss: 0.6200571417808532
Epoch 28, Loss: 0.7197347283363342
Test Accuracy: 76.92%
Average Test Loss: 0.700304608643055
Epoch 29, Loss: 0.47794729471206665
Test Accuracy: 78.79%
Average Test Loss: 0.6261018946766853
Epoch 30, Loss: 0.5295735001564026
Test Accuracy: 77.85%
Average Test Loss: 0.6576252403855324
Epoch 1, Loss: 2.1130971908569336
Test Accuracy: 21.83%
Average Test Loss: 2.1199641466140746
Epoch 2, Loss: 2.0151000022888184
Test Accuracy: 26.03%
Average Test Loss: 1.9826887047290802
Epoch 3, Loss: 1.951157569885254
Test Accuracy: 28.91%
```

```
Average Test Loss: 1.896073851585388
Epoch 4, Loss: 1.7753889560699463
Test Accuracy: 31.72%
Average Test Loss: 1.8258749902248383
Epoch 5, Loss: 1.9849843978881836
Test Accuracy: 33.67%
Average Test Loss: 1.7849309849739075
Epoch 6, Loss: 1.7050321102142334
Test Accuracy: 35.49%
Average Test Loss: 1.7151200115680694
Epoch 7, Loss: 1.6119356155395508
Test Accuracy: 37.3%
Average Test Loss: 1.6736756575107574
Epoch 8, Loss: 1.6430084705352783
Test Accuracy: 37.7%
Average Test Loss: 1.657423006296158
Epoch 9, Loss: 1.5228976011276245
Test Accuracy: 39.64%
Average Test Loss: 1.6099168026447297
Epoch 10, Loss: 1.548617959022522
Test Accuracy: 39.82%
Average Test Loss: 1.610041437149048
Epoch 11, Loss: 1.6587060689926147
Test Accuracy: 41.76%
Average Test Loss: 1.5521673345565796
Epoch 12, Loss: 1.5270836353302002
Test Accuracy: 42.58%
Average Test Loss: 1.542763783931732
Epoch 13, Loss: 1.5580203533172607
Test Accuracy: 43.6%
Average Test Loss: 1.5186910915374756
Epoch 14, Loss: 1.5021522045135498
Test Accuracy: 44.26%
Average Test Loss: 1.4941359210014342
Epoch 15, Loss: 1.5592600107192993
Test Accuracy: 45.01%
Average Test Loss: 1.4678609049320221
Epoch 16, Loss: 1.5997248888015747
Test Accuracy: 45.94%
Average Test Loss: 1.4618403804302216
Epoch 17, Loss: 1.4036003351211548
Test Accuracy: 46.4%
Average Test Loss: 1.4462925052642823
Epoch 18, Loss: 1.2742531299591064
Test Accuracy: 47.88%
Average Test Loss: 1.4098870980739593
Epoch 19, Loss: 1.2458622455596924
Test Accuracy: 47.89%
```

```
Average Test Loss: 1.4038107192516327
Epoch 20, Loss: 1.4104801416397095
Test Accuracy: 49.33%
Average Test Loss: 1.386195878982544
Epoch 21, Loss: 1.4003843069076538
Test Accuracy: 49.78%
Average Test Loss: 1.3656639790534972
Epoch 22, Loss: 1.451578140258789
Test Accuracy: 50.77%
Average Test Loss: 1.3420802855491638
Epoch 23, Loss: 1.3508851528167725
Test Accuracy: 51.01%
Average Test Loss: 1.3423824799060822
Epoch 24, Loss: 1.5017389059066772
Test Accuracy: 51.71%
Average Test Loss: 1.3257273769378661
Epoch 25, Loss: 1.385807752609253
Test Accuracy: 52.41%
Average Test Loss: 1.3055656993389129
Epoch 26, Loss: 1.5061572790145874
Test Accuracy: 52.85%
Average Test Loss: 1.2900380408763885
Epoch 27, Loss: 1.2172694206237793
Test Accuracy: 52.99%
Average Test Loss: 1.2934287261962891
Epoch 28, Loss: 1.372196912765503
Test Accuracy: 52.81%
Average Test Loss: 1.2986718374490738
Epoch 29, Loss: 1.3635036945343018
Test Accuracy: 53.89%
Average Test Loss: 1.2677104735374451
Epoch 30, Loss: 1.1420496702194214
Test Accuracy: 55.12%
Average Test Loss: 1.2406552916765212
```

14 2 - Load TensorBoard

```
[10]: %load_ext tensorboard
```

15 2 - Launch TensorBoard

```
[11]: %tensorboard --logdir runs
```

```
<IPython.core.display.Javascript object>
```

16 After trying all the learning rate, learning_rate = 0.01 performs the best with the maximum testing accuracy of 85.07%.

17 In addition, learning_rate = 0.0001 still need a lot more epoch to get better accuracy but it's definitely not worth it.