Angel Nieto García 2211798052

Homework #4

1. Source code
2. Dataloader: I went with the build Dataloader from torch.utils.data for simplicity

batch\_size = 64

trainloader = torch.utils.data.DataLoader(trainset, batch\_size,

shuffle=True, num\_workers=2)

valloader = torch.utils.data.DataLoader(validationset, batch\_size, num\_workers=2)

testloader = torch.utils.data.DataLoader(testset, batch\_size,

shuffle=False, num\_workers=2)

1. Model: Uses additional Conv layers to get better classification and at the end we add a fully connected layer with Dropouts to increase variance in the distribution.

device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu') # use GPU if available

class Net(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.conv\_layer = nn.Sequential(

# Conv Layer block 1

nn.Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(inplace=True),

nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2, stride=2),

# Conv Layer block 2

nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, padding=1),

nn.BatchNorm2d(128),

nn.ReLU(inplace=True),

nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2, stride=2),

nn.Dropout2d(p=0.05),

# Conv Layer block 3

nn.Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3, padding=1),

nn.BatchNorm2d(256),

nn.ReLU(inplace=True),

nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2, stride=2),

)

self.fc\_layer = nn.Sequential(

nn.Dropout(p=0.1),

nn.Linear(4096, 1024),

nn.ReLU(inplace=True),

nn.Linear(1024, 512),

nn.ReLU(inplace=True),

nn.Dropout(p=0.1),

nn.Linear(512, 10)

)

def forward(self, x):

"""Perform forward."""

# conv layers

x = self.conv\_layer(x)

# flatten

x = x.view(x.size(0), -1)

# fc layer

x = self.fc\_layer(x)

return x

def training\_step(self, batch):

images, labels = batch[0].to(device), batch[1].to(device)

out = self(images) # Generate predictions

loss = F.cross\_entropy(out, labels) # Calculate loss

return loss

@torch.no\_grad()

def validation\_step(self, batch):

images, labels = batch[0].to(device), batch[1].to(device)

out = self(images) # Generate predictions

loss = F.cross\_entropy(out, labels) # Calculate loss

acc = accuracy(out, labels) # Calculate accuracy

return {'val\_loss': loss.detach(), 'val\_acc': acc.detach()}

@torch.no\_grad()

def validation\_epoch\_end(self, outputs):

batch\_losses = [x['val\_loss'] for x in outputs]

epoch\_loss = torch.stack(batch\_losses).mean() # Combine losses

batch\_accs = [x['val\_acc'] for x in outputs]

epoch\_acc = torch.stack(batch\_accs).mean() # Combine accuracies

return {'val\_loss': epoch\_loss.item(), 'val\_acc': epoch\_acc.item()}

def epoch\_end(self, train\_loss, epoch, result):

print("Epoch [{}], train\_loss: {:.4f}, val\_loss: {:.4f}, val\_acc: {:.4f}".format(epoch, train\_loss, result['val\_loss'], result['val\_acc']))

net = Net()

net = net.to(device)

c) Loss Function

criterion = nn.CrossEntropyLoss()

d) Optimizer

optimizer = optim.Adam(net.parameters(), lr=0.001)

e) Evaluation

# Check the ground truth images

dataiter = iter(testloader)

images, labels = dataiter.next()

imshow(torchvision.utils.make\_grid(images[:4]))

print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

class\_correct = list(0. for i in range(10))

class\_total = list(0. for i in range(10))

with torch.no\_grad():

for data in testloader:

images, labels = data[0].to(device), data[1].to(device)

outputs = net(images)

\_, predicted = torch.max(outputs, 1)

c = (predicted == labels).squeeze()

for i in range(4):

label = labels[i]

class\_correct[label] += c[i].item()

class\_total[label] += 1

for i in range(10):

print('Accuracy of %5s : %2d %%' % (

classes[i], 100 \* class\_correct[i] / class\_total[i]))

f) Training Loop

@torch.no\_grad()

def evaluate(model, val\_loader):

outputs = [model.validation\_step(batch) for batch in val\_loader]

return model.validation\_epoch\_end(outputs)

def train(num\_epochs, train\_loader, val\_loader, net, criterion, optimizer):

history = []

train\_it = 0

for epoch in range(num\_epochs):

running\_loss = 0.0

for i, data in enumerate(train\_loader):

optimizer.zero\_grad()

# forward

loss = net.training\_step(data)

# backward

loss.backward()

# update the weights

optimizer.step() # 1 step over optimizer

running\_loss += loss.item()

train\_it += 1

# Validation phase

running\_loss /= len(train\_loader)

result = evaluate(net, val\_loader)

net.epoch\_end(running\_loss, epoch, result)

result['train\_loss'] = running\_loss

history.append(result)

return history

g) Datasets and Data augmentation

transform = transforms.Compose(

[transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train\_transform = transforms.Compose([transforms.RandomCrop(32, padding=4, padding\_mode='reflect'),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5), inplace=True)])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,

download=True, transform=train\_transform)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,

download=True, transform=transform)

classes = ('plane', 'car', 'bird', 'cat',

'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

val\_size = 10000

train\_size = len(trainset) - val\_size

trainset, validationset = random\_split(trainset, [train\_size, val\_size])

len(trainset), len(validationset)

1. Discussion

2.1 Evolution of training losses and validation losses with standard LeNet using Adam as optimizer, learning rate = 0.001 and 15 epochs

Epoch [0], train\_loss: 1.7679, val\_loss: 1.5919, val\_acc: 0.4102

Epoch [1], train\_loss: 1.5254, val\_loss: 1.4541, val\_acc: 0.4551

Epoch [2], train\_loss: 1.4113, val\_loss: 1.4554, val\_acc: 0.4741

Epoch [3], train\_loss: 1.3371, val\_loss: 1.3202, val\_acc: 0.5188

Epoch [4], train\_loss: 1.2914, val\_loss: 1.2871, val\_acc: 0.5285

Epoch [5], train\_loss: 1.2551, val\_loss: 1.2529, val\_acc: 0.5528

Epoch [6], train\_loss: 1.2291, val\_loss: 1.2044, val\_acc: 0.5684

Epoch [7], train\_loss: 1.2031, val\_loss: 1.2021, val\_acc: 0.5692

Epoch [8], train\_loss: 1.1826, val\_loss: 1.1784, val\_acc: 0.5811

Epoch [9], train\_loss: 1.1623, val\_loss: 1.1772, val\_acc: 0.5785

Epoch [10], train\_loss: 1.1441, val\_loss: 1.1426, val\_acc: 0.5989

Epoch [11], train\_loss: 1.1302, val\_loss: 1.1499, val\_acc: 0.5896

Epoch [12], train\_loss: 1.1183, val\_loss: 1.1505, val\_acc: 0.5938

Epoch [13], train\_loss: 1.1022, val\_loss: 1.1149, val\_acc: 0.5993

Epoch [14], train\_loss: 1.0912, val\_loss: 1.1035, val\_acc: 0.6050

Chart, line chart

Description automatically generated

* 1. Effects of parameter choices
     1. SGD Optimizer

As it can be appreciated in the following images a different optimizer such as SGD in augmented data produces worse results since learning is not as effective as with Adam optimizer

Epoch [0], train\_loss: 1.3938, val\_loss: 1.3751, val\_acc: 0.5109

Epoch [1], train\_loss: 1.3822, val\_loss: 1.3796, val\_acc: 0.5045

Epoch [2], train\_loss: 1.3795, val\_loss: 1.3624, val\_acc: 0.5107

Epoch [3], train\_loss: 1.3744, val\_loss: 1.3667, val\_acc: 0.5087

Epoch [4], train\_loss: 1.3696, val\_loss: 1.3683, val\_acc: 0.5101

Epoch [5], train\_loss: 1.3679, val\_loss: 1.3643, val\_acc: 0.5057

Epoch [6], train\_loss: 1.3669, val\_loss: 1.3616, val\_acc: 0.5184

Epoch [7], train\_loss: 1.3618, val\_loss: 1.3483, val\_acc: 0.5206

Epoch [8], train\_loss: 1.3580, val\_loss: 1.3561, val\_acc: 0.5166

Epoch [9], train\_loss: 1.3584, val\_loss: 1.3498, val\_acc: 0.5089

Epoch [10], train\_loss: 1.3572, val\_loss: 1.3625, val\_acc: 0.5097

Epoch [11], train\_loss: 1.3508, val\_loss: 1.3528, val\_acc: 0.5138

Epoch [12], train\_loss: 1.3519, val\_loss: 1.3527, val\_acc: 0.5198

Epoch [13], train\_loss: 1.3467, val\_loss: 1.3442, val\_acc: 0.5190

Epoch [14], train\_loss: 1.3509, val\_loss: 1.3326, val\_acc: 0.5225

Chart, line chart

Description automatically generated

A picture containing indoor, photo, small, sitting

Description automatically generated

GroundTruth: cat ship ship plane

Overall Accuracy: 55.46%

* + 1. Using 4x4 filter size instead of 5x5 in Convolutional layers

Epoch [0], train\_loss: 1.8183, val\_loss: 1.6362, val\_acc: 0.3952

Epoch [1], train\_loss: 1.5674, val\_loss: 1.4934, val\_acc: 0.4547

Epoch [2], train\_loss: 1.4569, val\_loss: 1.4402, val\_acc: 0.4790

Epoch [3], train\_loss: 1.3959, val\_loss: 1.3747, val\_acc: 0.4983

Epoch [4], train\_loss: 1.3431, val\_loss: 1.3266, val\_acc: 0.5211

Epoch [5], train\_loss: 1.2973, val\_loss: 1.3119, val\_acc: 0.5256

Epoch [6], train\_loss: 1.2687, val\_loss: 1.2681, val\_acc: 0.5452

Epoch [7], train\_loss: 1.2437, val\_loss: 1.2754, val\_acc: 0.5404

Epoch [8], train\_loss: 1.2258, val\_loss: 1.2327, val\_acc: 0.5588

Epoch [9], train\_loss: 1.2077, val\_loss: 1.2171, val\_acc: 0.5634

Epoch [10], train\_loss: 1.1879, val\_loss: 1.2080, val\_acc: 0.5634

Epoch [11], train\_loss: 1.1703, val\_loss: 1.1932, val\_acc: 0.5717

Epoch [12], train\_loss: 1.1531, val\_loss: 1.1466, val\_acc: 0.5898

Epoch [13], train\_loss: 1.1362, val\_loss: 1.1870, val\_acc: 0.5740

Epoch [14], train\_loss: 1.1191, val\_loss: 1.1597, val\_acc: 0.5874

Chart, line chart

Description automatically generated

A picture containing indoor, photo, small, sitting

Description automatically generated

GroundTruth: cat ship ship plane

Accuracy: 63.64%

* + 1. Using Batch normalization and Leaky\_Relu activation functions

Epoch [0], train\_loss: 1.6639, val\_loss: 1.5182, val\_acc: 0.4392

Epoch [1], train\_loss: 1.4383, val\_loss: 1.4117, val\_acc: 0.4836

Epoch [2], train\_loss: 1.3537, val\_loss: 1.3204, val\_acc: 0.5219

Epoch [3], train\_loss: 1.2949, val\_loss: 1.2780, val\_acc: 0.5491

Epoch [4], train\_loss: 1.2569, val\_loss: 1.2776, val\_acc: 0.5450

Epoch [5], train\_loss: 1.2269, val\_loss: 1.2515, val\_acc: 0.5539

Epoch [6], train\_loss: 1.1909, val\_loss: 1.2202, val\_acc: 0.5665

Epoch [7], train\_loss: 1.1788, val\_loss: 1.1920, val\_acc: 0.5770

Epoch [8], train\_loss: 1.1538, val\_loss: 1.1824, val\_acc: 0.5807

Epoch [9], train\_loss: 1.1398, val\_loss: 1.1773, val\_acc: 0.5833

Epoch [10], train\_loss: 1.1294, val\_loss: 1.1540, val\_acc: 0.5926

Epoch [11], train\_loss: 1.1067, val\_loss: 1.1678, val\_acc: 0.5923

Epoch [12], train\_loss: 1.1011, val\_loss: 1.1608, val\_acc: 0.5913

Epoch [13], train\_loss: 1.0888, val\_loss: 1.1391, val\_acc: 0.5970

Epoch [14], train\_loss: 1.0772, val\_loss: 1.1194, val\_acc: 0.5997

Chart, line chart

Description automatically generated

A picture containing indoor, photo, small, sitting

Description automatically generated

GroundTruth: cat ship ship plane

* + 1. Optimal parameters

In comparison Adam optimizer and learning rate = 0.001 produces much better results than SGD. Moreover, adding convolutional layers, dropout and batch normalization improves accuracy significantly as it can be appreciated in the following results:

Epoch [0], train\_loss: 1.6764, val\_loss: 1.3976, val\_acc: 0.4755

Epoch [1], train\_loss: 1.2933, val\_loss: 1.1780, val\_acc: 0.5775

Epoch [2], train\_loss: 1.1287, val\_loss: 1.0484, val\_acc: 0.6246

Epoch [3], train\_loss: 1.0226, val\_loss: 1.0036, val\_acc: 0.6450

Epoch [4], train\_loss: 0.9406, val\_loss: 0.8996, val\_acc: 0.6835

Epoch [5], train\_loss: 0.8790, val\_loss: 0.8807, val\_acc: 0.6936

Epoch [6], train\_loss: 0.8361, val\_loss: 0.8140, val\_acc: 0.7134

Epoch [7], train\_loss: 0.7906, val\_loss: 0.8205, val\_acc: 0.7237

Epoch [8], train\_loss: 0.7587, val\_loss: 0.7507, val\_acc: 0.7377

Epoch [9], train\_loss: 0.7215, val\_loss: 0.7441, val\_acc: 0.7389

Epoch [10], train\_loss: 0.6978, val\_loss: 0.7449, val\_acc: 0.7451

Epoch [11], train\_loss: 0.6702, val\_loss: 0.7387, val\_acc: 0.7478

Epoch [12], train\_loss: 0.6487, val\_loss: 0.6753, val\_acc: 0.7714

Epoch [13], train\_loss: 0.6320, val\_loss: 0.6648, val\_acc: 0.7722

Epoch [14], train\_loss: 0.6117, val\_loss: 0.6626, val\_acc: 0.7763

Chart, line chart

Description automatically generated

A picture containing indoor, photo, looking, sitting

Description automatically generated

GroundTruth: ship dog truck ship

Overall Accuracy: 78.54%

Top-1 prediction accuracy:

Accuracy of plane : 83 %

Accuracy of car : 88 %

Accuracy of bird : 65 %

Accuracy of cat : 58 %

Accuracy of deer : 65 %

Accuracy of dog : 67 %

Accuracy of frog : 85 %

Accuracy of horse : 71 %

Accuracy of ship : 94 %

Accuracy of truck : 89 %

Confusion matrix

|  |  |
| --- | --- |
|  | [746 19 53 12 13 0 5 8 109 35] (0) plane  [ 9 871 2 1 1 1 5 1 32 77] (1) car  [ 46 2 677 43 52 57 76 25 17 5] (2) bird  [ 10 6 56 615 44 135 67 28 15 24] (3) cat  [ 10 0 56 55 733 14 61 61 5 5] (4) deer  [ 7 2 41 132 34 726 12 36 5 5] (5) dog  [ 6 1 52 41 7 10 867 1 11 4] (6) frog  [ 14 2 27 30 59 60 5 784 7 12] (7) horse  [ 16 16 12 5 4 3 7 4 922 11] (8) ship  [ 14 25 1 3 1 2 3 2 36 913] (9) truck |

* 1. Example of failed cases

Changing drastically the training data with transformations such as a different hue space or saturation values or random rotations gives above 50% accuracy which shows the strength of Convolutional Neural Networks to work well any type data initialization and regularization.

A screen shot of a computer

Description automatically generated

GroundTruth: bird cat ship cat

More important are hyperparameters such as learning rate which can affect learning significantly if the wrong value is chosen.

* + 1. High learning rate = Fail

As seen in class a low learning rate will take a long time to learn where as a high learning rate can make the gradient either to explode or to jump around without failing in local minima which is what occurs in the following example:

Epoch [0], train\_loss: 211266.4182, val\_loss: 2.3638, val\_acc: 0.0999

Epoch [1], train\_loss: 2.3686, val\_loss: 2.4382, val\_acc: 0.0989

Epoch [2], train\_loss: 2.3774, val\_loss: 2.3666, val\_acc: 0.0989

Epoch [3], train\_loss: 2.3712, val\_loss: 2.3770, val\_acc: 0.1028

Epoch [4], train\_loss: 2.3667, val\_loss: 2.4131, val\_acc: 0.1042

Epoch [5], train\_loss: 2.3759, val\_loss: 2.3973, val\_acc: 0.0989

Epoch [6], train\_loss: 2.3689, val\_loss: 2.3469, val\_acc: 0.0989

Epoch [7], train\_loss: 2.3724, val\_loss: 2.3493, val\_acc: 0.1003

Epoch [8], train\_loss: 118.2178, val\_loss: 2.3806, val\_acc: 0.0989

Epoch [9], train\_loss: 2.3677, val\_loss: 2.4584, val\_acc: 0.1028

Epoch [10], train\_loss: 2.3771, val\_loss: 2.3687, val\_acc: 0.0999

Epoch [11], train\_loss: 2.3705, val\_loss: 2.4444, val\_acc: 0.0884

Epoch [12], train\_loss: 2.3813, val\_loss: 2.5112, val\_acc: 0.1028

Epoch [13], train\_loss: 2.3778, val\_loss: 2.4851, val\_acc: 0.1032

Epoch [14], train\_loss: 2.3776, val\_loss: 2.3423, val\_acc: 0.0989

A picture containing line chart

Description automatically generated

This model makes all test cases to fall under one category. It is straightforward to see that it would reach only around 10% accuracy which demonstrates why is important to assign an appropriate learning rate that does not underfit (high learning rate) nor overfit (low learning rate) our model.

A picture containing photo, sitting, food, small

Description automatically generated

GroundTruth: truck truck truck truck