A1_4_2

March 27, 2024

CIFAR10 on AlexNet

[7]:

```
import torch
import torchvision
import torchvision.transforms as trans
import torchvision.models as models
import torch.nn as nn
import torch.optim as op

# Set GPU
dev = torch.device("cuda:2" if torch.cuda.is_available() else "cpu")

# Define dataset transformations
tf = trans.Compose([
    trans.Resize((224, 224)),  # AlexNet expects 224x224 input
    trans.ToTensor(),
    trans.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR10 Train
train_ds = torchvision.datasets.CIFAR10(root='./data', train=True,␣
 ↪download=True, transform=tf)
train_l = torch.utils.data.DataLoader(train_ds, batch_size=64, shuffle=True,␣
 ↪num_workers=2)

# Load CIFAR10 Test
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False,␣
 ↪download=True, transform=tf)
test_l = torch.utils.data.DataLoader(test_ds, batch_size=64, shuffle=False,␣
 ↪num_workers=2)

#Load AlexNet
Alexnet = models.alexnet()

# Modified last FC layer to fit CIFAR10
n_ft = Alexnet.classifier[6].in_features
Alexnet.classifier[6] = nn.Linear(n_ft, 10)
```

```python
# To GPU
Alexnet.to(dev)

# Loss function and optimizer
c = nn.CrossEntropyLoss()
o = op.SGD(Alexnet.parameters(), lr=0.01, momentum=0.9)

# Training
num_ep = 10
for ep in range(num_ep):
    run_loss = 0.0
    for i, data in enumerate(train_l, 0):
        inputs, labels = data[0].to(dev), data[1].to(dev)
        o.zero_grad()

        # Forward pass
        outputs = Alexnet(inputs)
        loss = c(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        o.step()

        run_loss += loss.item()
        if i % 100 == 99:
            print('[Ep: %d, S: %5d] Loss: %.3f' %
                  (ep + 1, i + 1, run_loss / 100))
            run_loss = 0.0

print('Finished Training')

# Accuracy
def accuracy(ntw, dloader):
    corr = 0
    total = 0
    with torch.no_grad():
        for data in dloader:
            inputs, labels = data[0].to(dev), data[1].to(dev)
            outputs = ntw(inputs)
            _, pred = torch.max(outputs.data, 1)
            total += labels.size(0)
            corr += (pred == labels).sum().item()
    return corr / total

# Test
acc = accuracy(Alexnet, test_l)
print('Accuracy: %.2f %%' % (100 * acc))
```

```
Files already downloaded and verified
Files already downloaded and verified
[Ep: 1, S:   100] Loss: 2.302
[Ep: 1, S:   200] Loss: 2.276
[Ep: 1, S:   300] Loss: 2.084
[Ep: 1, S:   400] Loss: 1.965
[Ep: 1, S:   500] Loss: 1.825
[Ep: 1, S:   600] Loss: 1.705
[Ep: 1, S:   700] Loss: 1.609
[Ep: 2, S:   100] Loss: 1.470
[Ep: 2, S:   200] Loss: 1.409
[Ep: 2, S:   300] Loss: 1.357
[Ep: 2, S:   400] Loss: 1.291
[Ep: 2, S:   500] Loss: 1.269
[Ep: 2, S:   600] Loss: 1.241
[Ep: 2, S:   700] Loss: 1.186
[Ep: 3, S:   100] Loss: 1.068
[Ep: 3, S:   200] Loss: 1.043
[Ep: 3, S:   300] Loss: 1.021
[Ep: 3, S:   400] Loss: 0.989
[Ep: 3, S:   500] Loss: 0.956
[Ep: 3, S:   600] Loss: 0.924
[Ep: 3, S:   700] Loss: 0.909
[Ep: 4, S:   100] Loss: 0.861
[Ep: 4, S:   200] Loss: 0.852
[Ep: 4, S:   300] Loss: 0.834
[Ep: 4, S:   400] Loss: 0.800
[Ep: 4, S:   500] Loss: 0.793
[Ep: 4, S:   600] Loss: 0.801
[Ep: 4, S:   700] Loss: 0.748
[Ep: 5, S:   100] Loss: 0.672
[Ep: 5, S:   200] Loss: 0.677
[Ep: 5, S:   300] Loss: 0.710
[Ep: 5, S:   400] Loss: 0.682
[Ep: 5, S:   500] Loss: 0.658
[Ep: 5, S:   600] Loss: 0.674
[Ep: 5, S:   700] Loss: 0.659
[Ep: 6, S:   100] Loss: 0.577
[Ep: 6, S:   200] Loss: 0.591
[Ep: 6, S:   300] Loss: 0.580
[Ep: 6, S:   400] Loss: 0.574
[Ep: 6, S:   500] Loss: 0.576
[Ep: 6, S:   600] Loss: 0.591
[Ep: 6, S:   700] Loss: 0.616
[Ep: 7, S:   100] Loss: 0.497
[Ep: 7, S:   200] Loss: 0.499
[Ep: 7, S:   300] Loss: 0.504
[Ep: 7, S:   400] Loss: 0.498
```

```
[Ep: 7, S:   500] Loss: 0.522
[Ep: 7, S:   600] Loss: 0.483
[Ep: 7, S:   700] Loss: 0.504
[Ep: 8, S:   100] Loss: 0.402
[Ep: 8, S:   200] Loss: 0.423
[Ep: 8, S:   300] Loss: 0.431
[Ep: 8, S:   400] Loss: 0.459
[Ep: 8, S:   500] Loss: 0.447
[Ep: 8, S:   600] Loss: 0.455
[Ep: 8, S:   700] Loss: 0.448
[Ep: 9, S:   100] Loss: 0.346
[Ep: 9, S:   200] Loss: 0.359
[Ep: 9, S:   300] Loss: 0.376
[Ep: 9, S:   400] Loss: 0.384
[Ep: 9, S:   500] Loss: 0.374
[Ep: 9, S:   600] Loss: 0.394
[Ep: 9, S:   700] Loss: 0.391
[Ep: 10, S:   100] Loss: 0.319
[Ep: 10, S:   200] Loss: 0.343
[Ep: 10, S:   300] Loss: 0.335
[Ep: 10, S:   400] Loss: 0.319
[Ep: 10, S:   500] Loss: 0.339
[Ep: 10, S:   600] Loss: 0.349
[Ep: 10, S:   700] Loss: 0.378
Finished Training
Accuracy: 79.87 %
```

CIFAR10 on VGG16

[23]:

```python
import torch
import torchvision
import torchvision.transforms as trans
import torchvision.models as models
import torch.nn as nn
import torch.optim as op

# Set GPU
dev = torch.device("cuda:2" if torch.cuda.is_available() else "cpu")

# Define dataset transformations
tf = trans.Compose([
    trans.Resize((224, 224)),  # AlexNet expects 224x224 input
    trans.ToTensor(),
    trans.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR10 Train
```

```
train_ds = torchvision.datasets.CIFAR10(root='./data', train=True,␣
 ↪download=True, transform=tf)
train_l = torch.utils.data.DataLoader(train_ds, batch_size=64, shuffle=True,␣
 ↪num_workers=2)

# Load CIFAR10 Test
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False,␣
 ↪download=True, transform=tf)
test_l = torch.utils.data.DataLoader(test_ds, batch_size=64, shuffle=False,␣
 ↪num_workers=2)

#Load VGG-16
Vgg = models.vgg16(num_classes=10)

# To GPU
Vgg.to(dev)

# Loss function and optimizer
c = nn.CrossEntropyLoss()
o = op.SGD(Vgg.parameters(), lr=0.01, momentum=0.9)

# Training
num_ep = 10
for ep in range(num_ep):
    run_loss = 0.0
    for i, data in enumerate(train_l, 0):
        inputs, labels = data[0].to(dev), data[1].to(dev)
        o.zero_grad()

        # Forward pass
        outputs = Vgg(inputs)
        loss = c(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        o.step()

        run_loss += loss.item()
        if i % 100 == 99:
            print('[Ep: %d, S: %5d] Loss: %.3f' %
                  (ep + 1, i + 1, run_loss / 100))
            run_loss = 0.0

print('Finished Training')

# Accuracy
def accuracy(ntw, dloader):
    corr = 0
    total = 0
    with torch.no_grad():
        for data in dloader:
            inputs, labels = data[0].to(dev), data[1].to(dev)
            outputs = ntw(inputs)
            _, pred = torch.max(outputs.data, 1)
            total += labels.size(0)
            corr += (pred == labels).sum().item()
    return corr / total

# Test
acc = accuracy(Vgg, test_l)
print('Accuracy: %.2f %%' % (100 * acc))
```

```
Files already downloaded and verified
Files already downloaded and verified
[Ep: 1, S:   100] Loss: 2.254
[Ep: 1, S:   200] Loss: 2.065
[Ep: 1, S:   300] Loss: 1.939
[Ep: 1, S:   400] Loss: 1.821
[Ep: 1, S:   500] Loss: 1.720
[Ep: 1, S:   600] Loss: 1.615
[Ep: 1, S:   700] Loss: 1.542
[Ep: 2, S:   100] Loss: 1.425
[Ep: 2, S:   200] Loss: 1.362
[Ep: 2, S:   300] Loss: 1.302
[Ep: 2, S:   400] Loss: 1.262
[Ep: 2, S:   500] Loss: 1.196
[Ep: 2, S:   600] Loss: 1.180
[Ep: 2, S:   700] Loss: 1.120
[Ep: 3, S:   100] Loss: 1.022
[Ep: 3, S:   200] Loss: 0.953
[Ep: 3, S:   300] Loss: 0.942
[Ep: 3, S:   400] Loss: 0.909
[Ep: 3, S:   500] Loss: 0.907
[Ep: 3, S:   600] Loss: 0.868
[Ep: 3, S:   700] Loss: 0.844
[Ep: 4, S:   100] Loss: 0.752
[Ep: 4, S:   200] Loss: 0.714
[Ep: 4, S:   300] Loss: 0.739
[Ep: 4, S:   400] Loss: 0.722
[Ep: 4, S:   500] Loss: 0.704
[Ep: 4, S:   600] Loss: 0.682
[Ep: 4, S:   700] Loss: 0.694
[Ep: 5, S:   100] Loss: 0.553
[Ep: 5, S:   200] Loss: 0.521
```

```
[Ep: 5, S:   300] Loss: 0.579
[Ep: 5, S:   400] Loss: 0.559
[Ep: 5, S:   500] Loss: 0.582
[Ep: 5, S:   600] Loss: 0.565
[Ep: 5, S:   700] Loss: 0.548
[Ep: 6, S:   100] Loss: 0.409
[Ep: 6, S:   200] Loss: 0.417
[Ep: 6, S:   300] Loss: 0.414
[Ep: 6, S:   400] Loss: 0.438
[Ep: 6, S:   500] Loss: 0.418
[Ep: 6, S:   600] Loss: 0.458
[Ep: 6, S:   700] Loss: 0.430
[Ep: 7, S:   100] Loss: 0.267
[Ep: 7, S:   200] Loss: 0.301
[Ep: 7, S:   300] Loss: 0.341
[Ep: 7, S:   400] Loss: 0.314
[Ep: 7, S:   500] Loss: 0.343
[Ep: 7, S:   600] Loss: 0.326
[Ep: 7, S:   700] Loss: 0.340
[Ep: 8, S:   100] Loss: 0.212
[Ep: 8, S:   200] Loss: 0.211
[Ep: 8, S:   300] Loss: 0.241
[Ep: 8, S:   400] Loss: 0.224
[Ep: 8, S:   500] Loss: 0.236
[Ep: 8, S:   600] Loss: 0.275
[Ep: 8, S:   700] Loss: 0.245
[Ep: 9, S:   100] Loss: 0.136
[Ep: 9, S:   200] Loss: 0.164
[Ep: 9, S:   300] Loss: 0.150
[Ep: 9, S:   400] Loss: 0.184
[Ep: 9, S:   500] Loss: 0.166
[Ep: 9, S:   600] Loss: 0.181
[Ep: 9, S:   700] Loss: 0.196
[Ep: 10, S:   100] Loss: 0.113
[Ep: 10, S:   200] Loss: 0.132
[Ep: 10, S:   300] Loss: 0.135
[Ep: 10, S:   400] Loss: 0.130
[Ep: 10, S:   500] Loss: 0.146
[Ep: 10, S:   600] Loss: 0.144
[Ep: 10, S:   700] Loss: 0.159
Finished Training
Accuracy: 77.87 %
```

CIFAR10 on GoogleNet

[24]:
```
import torch
import torchvision
import torchvision.transforms as trans
import torchvision.models as models
import torch.nn as nn
import torch.optim as op

# Set GPU
dev = torch.device("cuda:2" if torch.cuda.is_available() else "cpu")

# Define dataset transformations
tf = trans.Compose([
    trans.Resize((224, 224)),  # AlexNet expects 224x224 input
    trans.ToTensor(),
    trans.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR10 Train
train_ds = torchvision.datasets.CIFAR10(root='./data', train=True,␣
 ↪download=True, transform=tf)
train_l = torch.utils.data.DataLoader(train_ds, batch_size=64, shuffle=True,␣
 ↪num_workers=2)

# Load CIFAR10 Test
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False,␣
 ↪download=True, transform=tf)
test_l = torch.utils.data.DataLoader(test_ds, batch_size=64, shuffle=False,␣
 ↪num_workers=2)

#Load VGG-16
Googlenet = models.googlenet(init_weights=True)

# Modified last FC layer to fit CIFAR10
Googlenet.fc = nn.Linear(1024, 10)


# To GPU
Googlenet.to(dev)

# Loss function and optimizer
c = nn.CrossEntropyLoss()
o = op.SGD(Googlenet.parameters(), lr=0.01, momentum=0.9)

# Training
num_ep = 10
for ep in range(num_ep):
    run_loss = 0.0
    for i, data in enumerate(train_l, 0):
        inputs, labels = data[0].to(dev), data[1].to(dev)
        o.zero_grad()
```

```python
        # Forward pass
        outputs = Googlenet(inputs)[0]
        loss = c(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        o.step()

        run_loss += loss.item()
        if i % 100 == 99:
            print('[Ep: %d, S: %5d] Loss: %.3f' %
                  (ep + 1, i + 1, run_loss / 100))
            run_loss = 0.0

print('Finished Training')

# Accuracy
def accuracy(ntw, dloader):
    corr = 0
    total = 0
    with torch.no_grad():
        for data in dloader:
            inputs, labels = data[0].to(dev), data[1].to(dev)
            outputs = ntw(inputs)[0]
            _, pred = torch.max(outputs.data, 1)
            total += labels.size(0)
            corr += (pred == labels).sum().item()
    return corr / total

# Test
acc = accuracy(Googlenet, test_l)
print('Accuracy: %.2f %%' % (100 * acc))
```

```
Files already downloaded and verified
Files already downloaded and verified
[Ep: 1, S:   100] Loss: 1.951
[Ep: 1, S:   200] Loss: 1.737
[Ep: 1, S:   300] Loss: 1.647
[Ep: 1, S:   400] Loss: 1.531
[Ep: 1, S:   500] Loss: 1.493
[Ep: 1, S:   600] Loss: 1.406
[Ep: 1, S:   700] Loss: 1.343
[Ep: 2, S:   100] Loss: 1.205
[Ep: 2, S:   200] Loss: 1.133
[Ep: 2, S:   300] Loss: 1.059
[Ep: 2, S:   400] Loss: 1.043
```

```
[Ep: 2, S:   500] Loss: 0.989
[Ep: 2, S:   600] Loss: 0.951
[Ep: 2, S:   700] Loss: 0.904
[Ep: 3, S:   100] Loss: 0.829
[Ep: 3, S:   200] Loss: 0.819
[Ep: 3, S:   300] Loss: 0.803
[Ep: 3, S:   400] Loss: 0.774
[Ep: 3, S:   500] Loss: 0.773
[Ep: 3, S:   600] Loss: 0.765
[Ep: 3, S:   700] Loss: 0.704
[Ep: 4, S:   100] Loss: 0.649
[Ep: 4, S:   200] Loss: 0.628
[Ep: 4, S:   300] Loss: 0.615
[Ep: 4, S:   400] Loss: 0.614
[Ep: 4, S:   500] Loss: 0.612
[Ep: 4, S:   600] Loss: 0.630
[Ep: 4, S:   700] Loss: 0.568
[Ep: 5, S:   100] Loss: 0.492
[Ep: 5, S:   200] Loss: 0.533
[Ep: 5, S:   300] Loss: 0.496
[Ep: 5, S:   400] Loss: 0.518
[Ep: 5, S:   500] Loss: 0.489
[Ep: 5, S:   600] Loss: 0.492
[Ep: 5, S:   700] Loss: 0.485
[Ep: 6, S:   100] Loss: 0.410
[Ep: 6, S:   200] Loss: 0.393
[Ep: 6, S:   300] Loss: 0.421
[Ep: 6, S:   400] Loss: 0.422
[Ep: 6, S:   500] Loss: 0.453
[Ep: 6, S:   600] Loss: 0.424
[Ep: 6, S:   700] Loss: 0.424
[Ep: 7, S:   100] Loss: 0.334
[Ep: 7, S:   200] Loss: 0.340
[Ep: 7, S:   300] Loss: 0.344
[Ep: 7, S:   400] Loss: 0.352
[Ep: 7, S:   500] Loss: 0.357
[Ep: 7, S:   600] Loss: 0.371
[Ep: 7, S:   700] Loss: 0.357
[Ep: 8, S:   100] Loss: 0.278
[Ep: 8, S:   200] Loss: 0.276
[Ep: 8, S:   300] Loss: 0.282
[Ep: 8, S:   400] Loss: 0.304
[Ep: 8, S:   500] Loss: 0.311
[Ep: 8, S:   600] Loss: 0.310
[Ep: 8, S:   700] Loss: 0.313
[Ep: 9, S:   100] Loss: 0.230
[Ep: 9, S:   200] Loss: 0.248
[Ep: 9, S:   300] Loss: 0.235
```

```
[Ep: 9, S:   400] Loss: 0.245
[Ep: 9, S:   500] Loss: 0.243
[Ep: 9, S:   600] Loss: 0.274
[Ep: 9, S:   700] Loss: 0.279
[Ep: 10, S:   100] Loss: 0.197
[Ep: 10, S:   200] Loss: 0.168
[Ep: 10, S:   300] Loss: 0.195
[Ep: 10, S:   400] Loss: 0.194
[Ep: 10, S:   500] Loss: 0.218
[Ep: 10, S:   600] Loss: 0.218
[Ep: 10, S:   700] Loss: 0.228
Finished Training
Accuracy: 83.54 %
```

CIFAR10 on ResNet152

[4]:

```python
import torch
import torchvision
import torchvision.transforms as trans
import torchvision.models as models
import torch.nn as nn
import torch.optim as op

# Set GPU
dev = torch.device("cuda:3" if torch.cuda.is_available() else "cpu")

# Define dataset transformations
tf = trans.Compose([
    trans.Resize((224, 224)),  # AlexNet expects 224x224 input
    trans.ToTensor(),
    trans.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR10 Train
train_ds = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=tf)
train_l = torch.utils.data.DataLoader(train_ds, batch_size=64, shuffle=True, num_workers=2)

# Load CIFAR10 Test
test_ds = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=tf)
test_l = torch.utils.data.DataLoader(test_ds, batch_size=64, shuffle=False, num_workers=2)

#Load Resnet152
Resnet = models.resnet152()
```

```
# Modified last FC layer to fit CIFAR10
Resnet.fc = nn.Linear(2048, 10)


# To GPU
Resnet.to(dev)

# Loss function and optimizer
c = nn.CrossEntropyLoss()
o = op.SGD(Resnet.parameters(), lr=0.01, momentum=0.9)

# Training
num_ep = 10
for ep in range(num_ep):
    run_loss = 0.0
    for i, data in enumerate(train_l, 0):
        inputs, labels = data[0].to(dev), data[1].to(dev)
        o.zero_grad()

        # Forward pass
        outputs = Resnet(inputs)
        loss = c(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        o.step()

        run_loss += loss.item()
        if i % 100 == 99:
            print('[Ep: %d, S: %5d] Loss: %.3f' %
                  (ep + 1, i + 1, run_loss / 100))
            run_loss = 0.0

print('Finished Training')

# Accuracy
def accuracy(ntw, dloader):
    corr = 0
    total = 0
    with torch.no_grad():
        for data in dloader:
            inputs, labels = data[0].to(dev), data[1].to(dev)
            outputs = ntw(inputs)
            _, pred = torch.max(outputs.data, 1)
            total += labels.size(0)
            corr += (pred == labels).sum().item()
    return corr / total
```

```python
# Test
acc = accuracy(Resnet, test_l)
print('Accuracy: %.2f %%' % (100 * acc))
```

```
Files already downloaded and verified
Files already downloaded and verified
[Ep: 1, S:   100] Loss: 2.800
[Ep: 1, S:   200] Loss: 2.070
[Ep: 1, S:   300] Loss: 1.904
[Ep: 1, S:   400] Loss: 1.818
[Ep: 1, S:   500] Loss: 1.730
[Ep: 1, S:   600] Loss: 1.683
[Ep: 1, S:   700] Loss: 1.594
[Ep: 2, S:   100] Loss: 1.546
[Ep: 2, S:   200] Loss: 1.502
[Ep: 2, S:   300] Loss: 1.444
[Ep: 2, S:   400] Loss: 1.432
[Ep: 2, S:   500] Loss: 1.419
[Ep: 2, S:   600] Loss: 1.354
[Ep: 2, S:   700] Loss: 1.328
[Ep: 3, S:   100] Loss: 1.250
[Ep: 3, S:   200] Loss: 1.236
[Ep: 3, S:   300] Loss: 1.188
[Ep: 3, S:   400] Loss: 1.172
[Ep: 3, S:   500] Loss: 1.179
[Ep: 3, S:   600] Loss: 1.143
[Ep: 3, S:   700] Loss: 1.100
[Ep: 4, S:   100] Loss: 0.989
[Ep: 4, S:   200] Loss: 0.990
[Ep: 4, S:   300] Loss: 0.970
[Ep: 4, S:   400] Loss: 0.946
[Ep: 4, S:   500] Loss: 0.947
[Ep: 4, S:   600] Loss: 0.957
[Ep: 4, S:   700] Loss: 0.882
[Ep: 5, S:   100] Loss: 0.816
[Ep: 5, S:   200] Loss: 0.781
[Ep: 5, S:   300] Loss: 0.790
[Ep: 5, S:   400] Loss: 0.779
[Ep: 5, S:   500] Loss: 0.749
[Ep: 5, S:   600] Loss: 0.748
[Ep: 5, S:   700] Loss: 0.739
[Ep: 6, S:   100] Loss: 0.629
[Ep: 6, S:   200] Loss: 0.624
[Ep: 6, S:   300] Loss: 0.606
[Ep: 6, S:   400] Loss: 0.635
[Ep: 6, S:   500] Loss: 0.625
```

```
[Ep: 6, S:   600] Loss: 0.610
[Ep: 6, S:   700] Loss: 0.614
[Ep: 7, S:   100] Loss: 0.455
[Ep: 7, S:   200] Loss: 0.476
[Ep: 7, S:   300] Loss: 0.493
[Ep: 7, S:   400] Loss: 0.509
[Ep: 7, S:   500] Loss: 0.484
[Ep: 7, S:   600] Loss: 0.523
[Ep: 7, S:   700] Loss: 0.497
[Ep: 8, S:   100] Loss: 0.340
[Ep: 8, S:   200] Loss: 0.357
[Ep: 8, S:   300] Loss: 0.414
[Ep: 8, S:   400] Loss: 0.414
[Ep: 8, S:   500] Loss: 0.439
[Ep: 8, S:   600] Loss: 0.416
[Ep: 8, S:   700] Loss: 0.424
[Ep: 9, S:   100] Loss: 0.253
[Ep: 9, S:   200] Loss: 0.296
[Ep: 9, S:   300] Loss: 0.347
[Ep: 9, S:   400] Loss: 0.335
[Ep: 9, S:   500] Loss: 0.355
[Ep: 9, S:   600] Loss: 0.377
[Ep: 9, S:   700] Loss: 0.388
[Ep: 10, S:   100] Loss: 0.193
[Ep: 10, S:   200] Loss: 0.211
[Ep: 10, S:   300] Loss: 0.267
[Ep: 10, S:   400] Loss: 0.279
[Ep: 10, S:   500] Loss: 0.285
[Ep: 10, S:   600] Loss: 0.282
[Ep: 10, S:   700] Loss: 0.300
Finished Training
Accuracy: 77.77 %
```