In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets
from torch.utils.data import DataLoader
```

In [2]:
```
print(torch.version.cuda)
```

```
11.8
```

In [3]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_dataset = datasets.ImageFolder('Dataset_train', transform=transform)
trainloader = DataLoader(train_dataset, batch_size=18, shuffle=True, num_workers=2)

test_dataset = datasets.ImageFolder('Dataset_test', transform=transform)
testloader = DataLoader(test_dataset, batch_size=18, shuffle=True, num_workers=2)

classes = ("Among Us", "Apex Legends", "Fortnite", "Forza Horizon", "Free Fire", "Genshin Impact", "God of War", "Mir
```

In [129…
```
# The function to show an image.
def imshow(img):
    img = img / 2 + 0.5     # Unnormalize.
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# Get some random training images.
dataiter = iter(trainloader)
images, labels = next(dataiter)
# Show images.
imshow(torchvision.utils.make_grid(images[:4]))
```

```
# Print labels.
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```


```
Fortnite Roblox Minecraft Fortnite
```

In [4]:
```
# If there are GPUs, choose the first one for computing. Otherwise use CPU.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# If 'cuda:0' is printed, it means GPU is available.#### Choose a Device
```

```
cuda:0
```

3 by 4 Residual Block

In [153…
```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, padding=12):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.relu = nn.ReLU(inplace=False)


    def forward(self, x):
```

```
        residual = x
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = self.conv3(out)
        out += residual
        out = F.relu(out)
        return out
```

In [154…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(3, 10, kernel_size=25, stride=1, padding=12),
            nn.BatchNorm2d(10),
            nn.ReLU(inplace=False)

        )

        self.res_block1 = ResidualBlock(10, 10)

        self.res_block2 = ResidualBlock(10, 10)

        self.res_block3 = ResidualBlock(10, 10)

        self.res_block4 = ResidualBlock(10, 10)

        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(8800, 100),
            nn.ReLU(inplace=False),
            nn.Linear(100, 10)
        )

        self.pool = nn.AvgPool2d(2)
    def forward(self, x):
        x = self.conv_block1(x)
        x = self.pool(x)
        x = self.res_block1(x)
        x = self.pool(x)
        x = self.res_block2(x)
```

```
        x = self.pool(x)
        x = self.res_block3(x)
        x = self.pool(x)
        x = self.res_block4(x)
        x = self.fc_layers(x)
        return x

net = Net()     # Create the network instance.
net.to(device)  # Move the network parameters to the specified device.
```

Out[154…

```
Net(
  (conv_block1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
    (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (res_block1): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block2): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block3): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
```

```
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block4): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (fc_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=8800, out_features=100, bias=True)
    (2): ReLU()
    (3): Linear(in_features=100, out_features=10, bias=True)
  )
  (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
)
```

In [155…

```
# We use cross-entropy as loss function.
loss_func = nn.CrossEntropyLoss()
# We use stochastic gradient descent (SGD) as optimizer.
opt = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [156…

```
avg_losses = []   # Avg. losses.
epochs = 5        # Total epochs.
print_freq = 20  # Print frequency.
```

```
for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0       # Initialize running loss.
    for i, data in enumerate(trainloader, 0):
        # Get the inputs.
        inputs, labels = data

        # Move the inputs to the specified device.
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients.
        opt.zero_grad()

        # Forward step.
        outputs = net(inputs)
        loss = loss_func(outputs, labels)

        # Backward step.
        loss.backward()

        # Optimization step (update the parameters).
        opt.step()

        # Print statistics.
        running_loss += loss.item()
        if i % print_freq == print_freq - 1: # Print every several mini-batches.
            avg_loss = running_loss / print_freq
            print('[epoch: {}, i: {:5d}] avg mini-batch loss: {:.3f}'.format(
                epoch, i, avg_loss))
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
[epoch: 0, i:    19] avg mini-batch loss: 2.190
[epoch: 0, i:    39] avg mini-batch loss: 1.921
[epoch: 0, i:    59] avg mini-batch loss: 1.702
[epoch: 0, i:    79] avg mini-batch loss: 1.441
[epoch: 0, i:    99] avg mini-batch loss: 1.045
[epoch: 0, i:   119] avg mini-batch loss: 0.966
[epoch: 0, i:   139] avg mini-batch loss: 0.829
[epoch: 0, i:   159] avg mini-batch loss: 0.840
[epoch: 0, i:   179] avg mini-batch loss: 0.689
[epoch: 0, i:   199] avg mini-batch loss: 0.630
[epoch: 0, i:   219] avg mini-batch loss: 0.714
[epoch: 0, i:   239] avg mini-batch loss: 0.703
[epoch: 0, i:   259] avg mini-batch loss: 0.598
[epoch: 1, i:    19] avg mini-batch loss: 0.300
[epoch: 1, i:    39] avg mini-batch loss: 0.312
[epoch: 1, i:    59] avg mini-batch loss: 0.324
[epoch: 1, i:    79] avg mini-batch loss: 0.367
[epoch: 1, i:    99] avg mini-batch loss: 0.369
[epoch: 1, i:   119] avg mini-batch loss: 0.307
[epoch: 1, i:   139] avg mini-batch loss: 0.269
[epoch: 1, i:   159] avg mini-batch loss: 0.326
[epoch: 1, i:   179] avg mini-batch loss: 0.368
[epoch: 1, i:   199] avg mini-batch loss: 0.334
[epoch: 1, i:   219] avg mini-batch loss: 0.282
[epoch: 1, i:   239] avg mini-batch loss: 0.249
[epoch: 1, i:   259] avg mini-batch loss: 0.290
[epoch: 2, i:    19] avg mini-batch loss: 0.162
[epoch: 2, i:    39] avg mini-batch loss: 0.270
[epoch: 2, i:    59] avg mini-batch loss: 0.160
[epoch: 2, i:    79] avg mini-batch loss: 0.171
[epoch: 2, i:    99] avg mini-batch loss: 0.117
[epoch: 2, i:   119] avg mini-batch loss: 0.176
[epoch: 2, i:   139] avg mini-batch loss: 0.165
[epoch: 2, i:   159] avg mini-batch loss: 0.196
[epoch: 2, i:   179] avg mini-batch loss: 0.191
[epoch: 2, i:   199] avg mini-batch loss: 0.137
[epoch: 2, i:   219] avg mini-batch loss: 0.109
[epoch: 2, i:   239] avg mini-batch loss: 0.146
[epoch: 2, i:   259] avg mini-batch loss: 0.179
[epoch: 3, i:    19] avg mini-batch loss: 0.121
[epoch: 3, i:    39] avg mini-batch loss: 0.076
[epoch: 3, i:    59] avg mini-batch loss: 0.144
```

```
[epoch: 3, i:    79] avg mini-batch loss: 0.158
[epoch: 3, i:    99] avg mini-batch loss: 0.143
[epoch: 3, i:   119] avg mini-batch loss: 0.116
[epoch: 3, i:   139] avg mini-batch loss: 0.113
[epoch: 3, i:   159] avg mini-batch loss: 0.121
[epoch: 3, i:   179] avg mini-batch loss: 0.102
[epoch: 3, i:   199] avg mini-batch loss: 0.101
[epoch: 3, i:   219] avg mini-batch loss: 0.100
[epoch: 3, i:   239] avg mini-batch loss: 0.075
[epoch: 3, i:   259] avg mini-batch loss: 0.097
[epoch: 4, i:    19] avg mini-batch loss: 0.079
[epoch: 4, i:    39] avg mini-batch loss: 0.063
[epoch: 4, i:    59] avg mini-batch loss: 0.109
[epoch: 4, i:    79] avg mini-batch loss: 0.062
[epoch: 4, i:    99] avg mini-batch loss: 0.036
[epoch: 4, i:   119] avg mini-batch loss: 0.051
[epoch: 4, i:   139] avg mini-batch loss: 0.069
[epoch: 4, i:   159] avg mini-batch loss: 0.043
[epoch: 4, i:   179] avg mini-batch loss: 0.045
[epoch: 4, i:   199] avg mini-batch loss: 0.064
[epoch: 4, i:   219] avg mini-batch loss: 0.054
[epoch: 4, i:   239] avg mini-batch loss: 0.064
[epoch: 4, i:   259] avg mini-batch loss: 0.055
Finished Training.
```

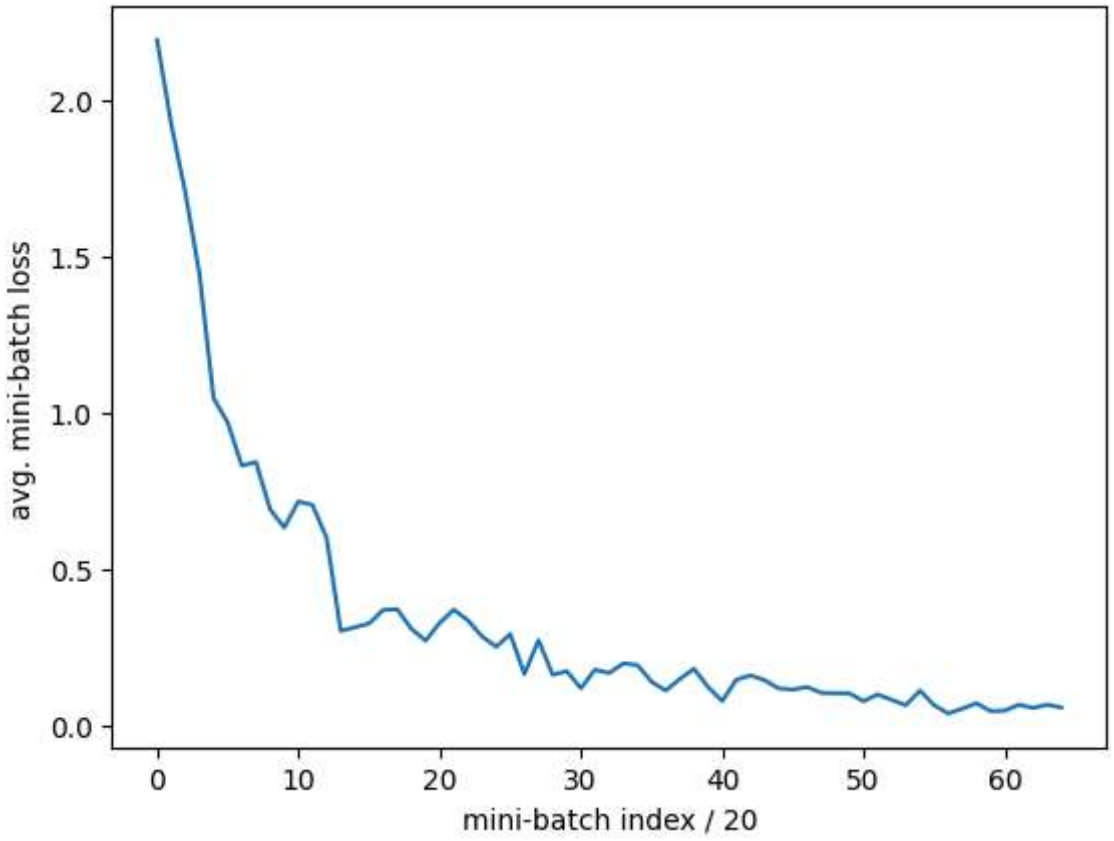
In [201…
```
torch.cuda.empty_cache()
```

In [192…
```
torch.autograd.set_detect_anomaly(True)
```

Out[192…
```
<torch.autograd.anomaly_mode.set_detect_anomaly at 0x1434e96a650>
```

In [157…
```
torch.save(net.state_dict(), 'cnn_res_3by4_model.pth')
```

In [159…
```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```

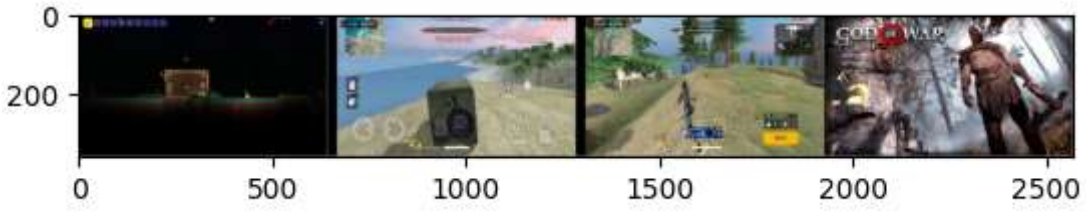


Evaluate on Test Dataset

In [160…

```
# Check several images.
dataiter = iter(testloader)
images, labels = next(dataiter)
imshow(torchvision.utils.make_grid(images[:4]))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
outputs = net(images.to(device))
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```




```
GroundTruth:  Terraria Free Fire Free Fire God of War
Predicted:  Terraria Free Fire Free Fire Genshin Impact
```

In [161…

```python
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 87 %
```

In [134…

```python
# Get test accuracy for each class.
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
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(len(labels)):
            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]))
```

```
Accuracy of Among Us : 97 %
Accuracy of Apex Legends : 68 %
Accuracy of Fortnite : 92 %
Accuracy of Forza Horizon : 78 %
Accuracy of Free Fire : 98 %
Accuracy of Genshin Impact : 91 %
Accuracy of God of War : 80 %
Accuracy of Minecraft : 97 %
Accuracy of Roblox : 93 %
Accuracy of Terraria : 100 %
```

4 by 3 Residual Block

In [168…

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, padding=12):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )

        self.relu = nn.ReLU(inplace=False)
```

```
    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out = F.relu(out)
        out = self.conv3(out)
        out = F.relu(out)
        out = self.conv4(out)
        out += residual
        out = F.relu(out)
        return out
```

In [169…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(3, 10, kernel_size=25, stride=1, padding=12),
            nn.BatchNorm2d(10),
            nn.ReLU(inplace=False)

        )

        self.res_block1 = ResidualBlock(10, 10)

        self.res_block2 = ResidualBlock(10, 10)

        self.res_block3 = ResidualBlock(10, 10)


        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(8800, 100),
            nn.ReLU(inplace=False),
            nn.Linear(100, 10)
        )

        self.pool = nn.AvgPool2d(2)
    def forward(self, x):
        x = self.conv_block1(x)
        x = self.pool(x)
        x = self.res_block1(x)
```

```
        x = self.pool(x)
        x = self.res_block2(x)
        x = self.pool(x)
        x = self.res_block3(x)
        x = self.pool(x)
        x = self.fc_layers(x)
        return x

net = Net()     # Create the network instance.
net.to(device)  # Move the network parameters to the specified device.
```

Out[169…

```
Net(
  (conv_block1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
    (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (res_block1): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv4): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block2): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv4): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

```
    (relu): ReLU()
  )
  (res_block3): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv3): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv4): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (fc_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=8800, out_features=100, bias=True)
    (2): ReLU()
    (3): Linear(in_features=100, out_features=10, bias=True)
  )
  (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
)
```

In [170…

```
# We use cross-entropy as loss function.
loss_func = nn.CrossEntropyLoss()
# We use stochastic gradient descent (SGD) as optimizer.
opt = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [171…

```
avg_losses = []   # Avg. losses.
epochs = 5        # Total epochs.
print_freq = 20  # Print frequency.

for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0       # Initialize running loss.
    for i, data in enumerate(trainloader, 0):
```

```
        # Get the inputs.
        inputs, labels = data

        # Move the inputs to the specified device.
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients.
        opt.zero_grad()

        # Forward step.
        outputs = net(inputs)
        loss = loss_func(outputs, labels)

        # Backward step.
        loss.backward()

        # Optimization step (update the parameters).
        opt.step()

        # Print statistics.
        running_loss += loss.item()
        if i % print_freq == print_freq - 1: # Print every several mini-batches.
            avg_loss = running_loss / print_freq
            print('[epoch: {}, i: {:5d}] avg mini-batch loss: {:.3f}'.format(
                epoch, i, avg_loss))
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
[epoch: 0, i:    19] avg mini-batch loss: 2.175
[epoch: 0, i:    39] avg mini-batch loss: 1.964
[epoch: 0, i:    59] avg mini-batch loss: 1.589
[epoch: 0, i:    79] avg mini-batch loss: 1.360
[epoch: 0, i:    99] avg mini-batch loss: 0.956
[epoch: 0, i:   119] avg mini-batch loss: 0.869
[epoch: 0, i:   139] avg mini-batch loss: 0.793
[epoch: 0, i:   159] avg mini-batch loss: 0.720
[epoch: 0, i:   179] avg mini-batch loss: 0.519
[epoch: 0, i:   199] avg mini-batch loss: 0.591
[epoch: 0, i:   219] avg mini-batch loss: 0.473
[epoch: 0, i:   239] avg mini-batch loss: 0.467
[epoch: 0, i:   259] avg mini-batch loss: 0.561
[epoch: 1, i:    19] avg mini-batch loss: 0.349
[epoch: 1, i:    39] avg mini-batch loss: 0.343
[epoch: 1, i:    59] avg mini-batch loss: 0.377
[epoch: 1, i:    79] avg mini-batch loss: 0.303
[epoch: 1, i:    99] avg mini-batch loss: 0.281
[epoch: 1, i:   119] avg mini-batch loss: 0.317
[epoch: 1, i:   139] avg mini-batch loss: 0.311
[epoch: 1, i:   159] avg mini-batch loss: 0.259
[epoch: 1, i:   179] avg mini-batch loss: 0.164
[epoch: 1, i:   199] avg mini-batch loss: 0.230
[epoch: 1, i:   219] avg mini-batch loss: 0.300
[epoch: 1, i:   239] avg mini-batch loss: 0.236
[epoch: 1, i:   259] avg mini-batch loss: 0.285
[epoch: 2, i:    19] avg mini-batch loss: 0.153
[epoch: 2, i:    39] avg mini-batch loss: 0.148
[epoch: 2, i:    59] avg mini-batch loss: 0.165
[epoch: 2, i:    79] avg mini-batch loss: 0.121
[epoch: 2, i:    99] avg mini-batch loss: 0.120
[epoch: 2, i:   119] avg mini-batch loss: 0.136
[epoch: 2, i:   139] avg mini-batch loss: 0.094
[epoch: 2, i:   159] avg mini-batch loss: 0.104
[epoch: 2, i:   179] avg mini-batch loss: 0.087
[epoch: 2, i:   199] avg mini-batch loss: 0.175
[epoch: 2, i:   219] avg mini-batch loss: 0.125
[epoch: 2, i:   239] avg mini-batch loss: 0.116
[epoch: 2, i:   259] avg mini-batch loss: 0.110
[epoch: 3, i:    19] avg mini-batch loss: 0.123
[epoch: 3, i:    39] avg mini-batch loss: 0.100
[epoch: 3, i:    59] avg mini-batch loss: 0.128
```

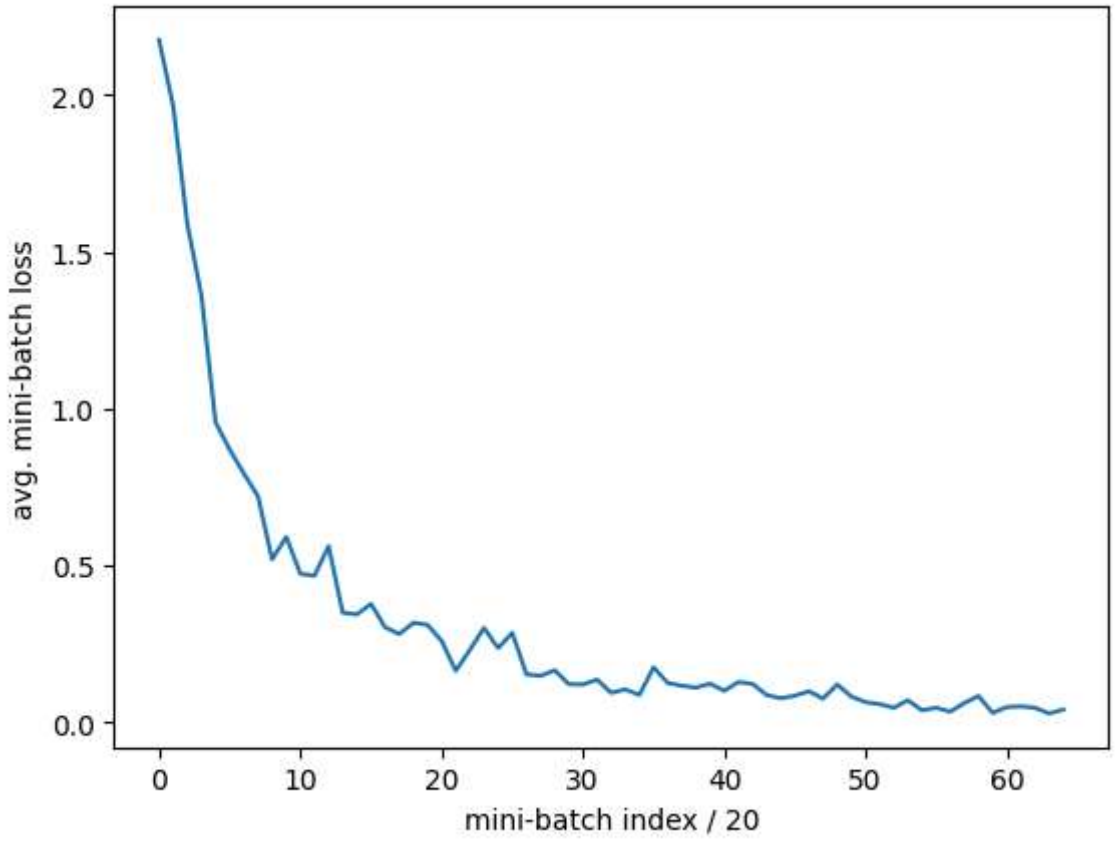
```
[epoch: 3, i:    79] avg mini-batch loss: 0.122
[epoch: 3, i:    99] avg mini-batch loss: 0.087
[epoch: 3, i:   119] avg mini-batch loss: 0.076
[epoch: 3, i:   139] avg mini-batch loss: 0.084
[epoch: 3, i:   159] avg mini-batch loss: 0.098
[epoch: 3, i:   179] avg mini-batch loss: 0.075
[epoch: 3, i:   199] avg mini-batch loss: 0.120
[epoch: 3, i:   219] avg mini-batch loss: 0.081
[epoch: 3, i:   239] avg mini-batch loss: 0.064
[epoch: 3, i:   259] avg mini-batch loss: 0.058
[epoch: 4, i:    19] avg mini-batch loss: 0.046
[epoch: 4, i:    39] avg mini-batch loss: 0.070
[epoch: 4, i:    59] avg mini-batch loss: 0.038
[epoch: 4, i:    79] avg mini-batch loss: 0.046
[epoch: 4, i:    99] avg mini-batch loss: 0.034
[epoch: 4, i:   119] avg mini-batch loss: 0.061
[epoch: 4, i:   139] avg mini-batch loss: 0.084
[epoch: 4, i:   159] avg mini-batch loss: 0.029
[epoch: 4, i:   179] avg mini-batch loss: 0.048
[epoch: 4, i:   199] avg mini-batch loss: 0.050
[epoch: 4, i:   219] avg mini-batch loss: 0.046
[epoch: 4, i:   239] avg mini-batch loss: 0.028
[epoch: 4, i:   259] avg mini-batch loss: 0.040
Finished Training.
```

In [172…

```
torch.save(net.state_dict(), 'cnn_res_4_by_3_model.pth')
```

In [174…

```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



In [175…

```
# Check several images.
dataiter = iter(testloader)
images, labels = next(dataiter)
imshow(torchvision.utils.make_grid(images[:4]))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
outputs = net(images.to(device))
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```


```
GroundTruth:  Apex Legends Free Fire Forza Horizon Minecraft
Predicted:  Roblox Free Fire Forza Horizon Minecraft
```

In [176…

```python
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 89 %
```

In [177…

```python
# Get test accuracy for each class.
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
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(len(labels)):
            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]))
```

```
Accuracy of Among Us : 95 %
Accuracy of Apex Legends : 72 %
Accuracy of Fortnite : 91 %
Accuracy of Forza Horizon : 69 %
Accuracy of Free Fire : 97 %
Accuracy of Genshin Impact : 91 %
Accuracy of God of War : 82 %
Accuracy of Minecraft : 93 %
Accuracy of Roblox : 93 %
Accuracy of Terraria : 100 %
```

2 by 6 Residual Block

In [196…

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, padding=12):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=25, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
        )
        self.relu = nn.ReLU(inplace=False)


    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = F.relu(out)
        out = self.conv2(out)
        out += residual
        out = F.relu(out)
        return out
```

In [197…

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(3, 10, kernel_size=25, stride=1, padding=12),
            nn.BatchNorm2d(10),
            nn.ReLU(inplace=False)

        )

        self.res_block1 = ResidualBlock(10, 10)

        self.res_block2 = ResidualBlock(10, 10)

        self.res_block3 = ResidualBlock(10, 10)

        self.res_block4 = ResidualBlock(10, 10)

        self.res_block5 = ResidualBlock(10, 10)

        self.res_block6 = ResidualBlock(10, 10)

        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(8800, 100),
            nn.ReLU(inplace=False),
            nn.Linear(100, 10)
        )

        self.pool = nn.AvgPool2d(2)
    def forward(self, x):
        x = self.conv_block1(x)
        x = self.res_block1(x)
        x = self.pool(x)
        x = self.res_block2(x)
        x = self.pool(x)
        x = self.res_block3(x)
        x = self.pool(x)
        x = self.res_block4(x)
        x = self.pool(x)
        x = self.res_block5(x)
```

```
        x = self.res_block6(x)
        x = self.fc_layers(x)
        return x

net = Net()     # Create the network instance.
net.to(device)  # Move the network parameters to the specified device.
```

Out[197…

```
Net(
  (conv_block1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
    (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (res_block1): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block2): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block3): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block4): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
```

```
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block5): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (res_block6): ResidualBlock(
    (conv1): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (conv2): Sequential(
      (0): Conv2d(10, 10, kernel_size=(25, 25), stride=(1, 1), padding=(12, 12))
      (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (relu): ReLU()
  )
  (fc_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=8800, out_features=100, bias=True)
    (2): ReLU()
    (3): Linear(in_features=100, out_features=10, bias=True)
  )
  (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
)
```

In [198…

```python
# We use cross-entropy as loss function.
loss_func = nn.CrossEntropyLoss()
```

```
# We use stochastic gradient descent (SGD) as optimizer.
opt = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [199…

```
avg_losses = []   # Avg. losses.
epochs = 5        # Total epochs.
print_freq = 20  # Print frequency.

for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0       # Initialize running loss.
    for i, data in enumerate(trainloader, 0):
        # Get the inputs.
        inputs, labels = data

        # Move the inputs to the specified device.
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients.
        opt.zero_grad()

        # Forward step.
        outputs = net(inputs)
        loss = loss_func(outputs, labels)

        # Backward step.
        loss.backward()

        # Optimization step (update the parameters).
        opt.step()

        # Print statistics.
        running_loss += loss.item()
        if i % print_freq == print_freq - 1: # Print every several mini-batches.
            avg_loss = running_loss / print_freq
            print('[epoch: {}, i: {:5d}] avg mini-batch loss: {:.3f}'.format(
                epoch, i, avg_loss))
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
[epoch: 0, i:    19] avg mini-batch loss: 2.251
[epoch: 0, i:    39] avg mini-batch loss: 1.911
[epoch: 0, i:    59] avg mini-batch loss: 1.658
[epoch: 0, i:    79] avg mini-batch loss: 1.182
[epoch: 0, i:    99] avg mini-batch loss: 1.098
[epoch: 0, i:   119] avg mini-batch loss: 0.721
[epoch: 0, i:   139] avg mini-batch loss: 0.643
[epoch: 0, i:   159] avg mini-batch loss: 0.614
[epoch: 0, i:   179] avg mini-batch loss: 0.581
[epoch: 0, i:   199] avg mini-batch loss: 0.600
[epoch: 0, i:   219] avg mini-batch loss: 0.539
[epoch: 0, i:   239] avg mini-batch loss: 0.342
[epoch: 0, i:   259] avg mini-batch loss: 0.337
[epoch: 1, i:    19] avg mini-batch loss: 0.361
[epoch: 1, i:    39] avg mini-batch loss: 0.238
[epoch: 1, i:    59] avg mini-batch loss: 0.263
[epoch: 1, i:    79] avg mini-batch loss: 0.261
[epoch: 1, i:    99] avg mini-batch loss: 0.231
[epoch: 1, i:   119] avg mini-batch loss: 0.211
[epoch: 1, i:   139] avg mini-batch loss: 0.209
[epoch: 1, i:   159] avg mini-batch loss: 0.222
[epoch: 1, i:   179] avg mini-batch loss: 0.210
[epoch: 1, i:   199] avg mini-batch loss: 0.168
[epoch: 1, i:   219] avg mini-batch loss: 0.207
[epoch: 1, i:   239] avg mini-batch loss: 0.153
[epoch: 1, i:   259] avg mini-batch loss: 0.175
[epoch: 2, i:    19] avg mini-batch loss: 0.132
[epoch: 2, i:    39] avg mini-batch loss: 0.059
[epoch: 2, i:    59] avg mini-batch loss: 0.053
[epoch: 2, i:    79] avg mini-batch loss: 0.055
[epoch: 2, i:    99] avg mini-batch loss: 0.099
[epoch: 2, i:   119] avg mini-batch loss: 0.100
[epoch: 2, i:   139] avg mini-batch loss: 0.142
[epoch: 2, i:   159] avg mini-batch loss: 0.111
[epoch: 2, i:   179] avg mini-batch loss: 0.139
[epoch: 2, i:   199] avg mini-batch loss: 0.149
[epoch: 2, i:   219] avg mini-batch loss: 0.100
[epoch: 2, i:   239] avg mini-batch loss: 0.130
[epoch: 2, i:   259] avg mini-batch loss: 0.127
[epoch: 3, i:    19] avg mini-batch loss: 0.060
[epoch: 3, i:    39] avg mini-batch loss: 0.037
[epoch: 3, i:    59] avg mini-batch loss: 0.033
```

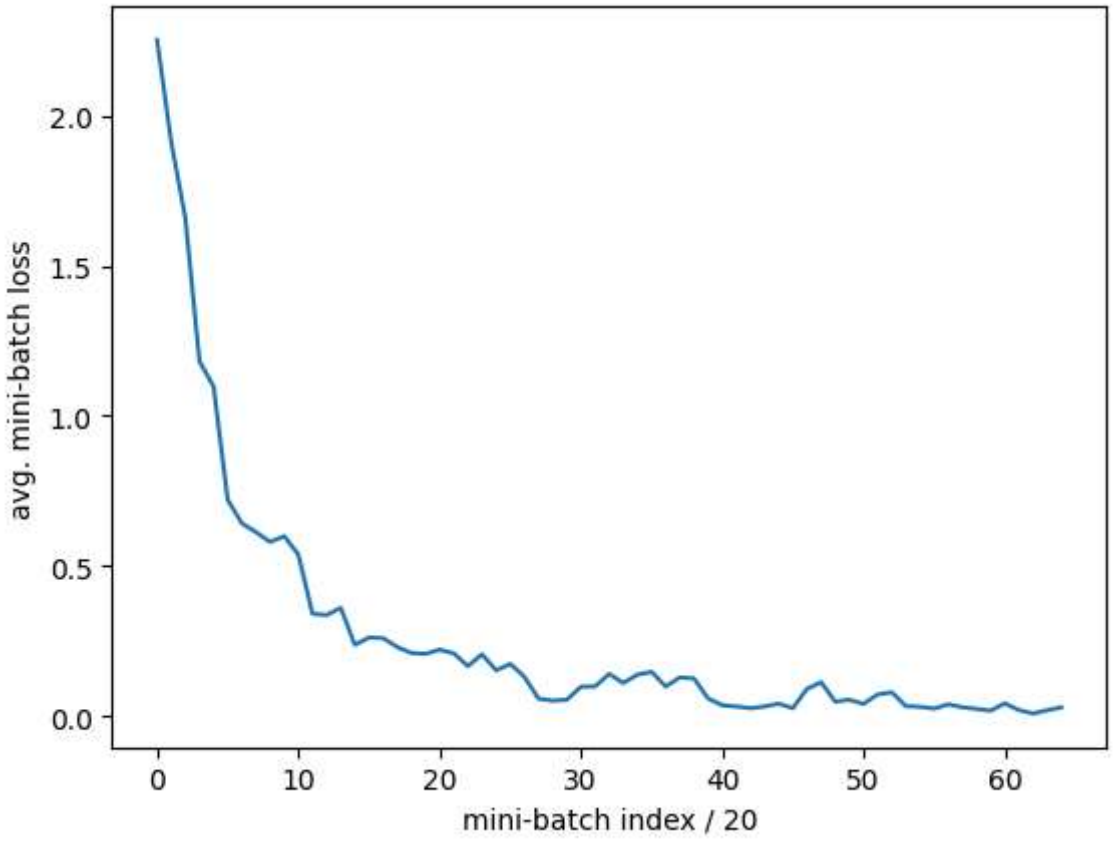
```
[epoch: 3, i:    79] avg mini-batch loss: 0.027
[epoch: 3, i:    99] avg mini-batch loss: 0.033
[epoch: 3, i:   119] avg mini-batch loss: 0.043
[epoch: 3, i:   139] avg mini-batch loss: 0.028
[epoch: 3, i:   159] avg mini-batch loss: 0.092
[epoch: 3, i:   179] avg mini-batch loss: 0.114
[epoch: 3, i:   199] avg mini-batch loss: 0.049
[epoch: 3, i:   219] avg mini-batch loss: 0.056
[epoch: 3, i:   239] avg mini-batch loss: 0.042
[epoch: 3, i:   259] avg mini-batch loss: 0.073
[epoch: 4, i:    19] avg mini-batch loss: 0.081
[epoch: 4, i:    39] avg mini-batch loss: 0.035
[epoch: 4, i:    59] avg mini-batch loss: 0.032
[epoch: 4, i:    79] avg mini-batch loss: 0.026
[epoch: 4, i:    99] avg mini-batch loss: 0.041
[epoch: 4, i:   119] avg mini-batch loss: 0.031
[epoch: 4, i:   139] avg mini-batch loss: 0.025
[epoch: 4, i:   159] avg mini-batch loss: 0.019
[epoch: 4, i:   179] avg mini-batch loss: 0.044
[epoch: 4, i:   199] avg mini-batch loss: 0.021
[epoch: 4, i:   219] avg mini-batch loss: 0.010
[epoch: 4, i:   239] avg mini-batch loss: 0.021
[epoch: 4, i:   259] avg mini-batch loss: 0.030
Finished Training.
```

In [200…

```
torch.save(net.state_dict(), 'cnn_res_2_by_6_model.pth')
```

In [202…

```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



In [203…

```
# Check several images.
dataiter = iter(testloader)
images, labels = next(dataiter)
imshow(torchvision.utils.make_grid(images[:4]))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
outputs = net(images.to(device))
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```


```
GroundTruth:  Free Fire Among Us Genshin Impact Terraria
Predicted:  Free Fire Among Us Genshin Impact Terraria
```

In [204…

```python
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 90 %
```

In [205…

```python
# Get test accuracy for each class.
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
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(len(labels)):
            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]))
```

```
Accuracy of Among Us : 96 %
Accuracy of Apex Legends : 68 %
Accuracy of Fortnite : 96 %
Accuracy of Forza Horizon : 79 %
Accuracy of Free Fire : 99 %
Accuracy of Genshin Impact : 95 %
Accuracy of God of War : 80 %
Accuracy of Minecraft : 98 %
Accuracy of Roblox : 97 %
Accuracy of Terraria : 100 %
```

In [ ]: