Appendix-8: Residual Neural Network (ResNet):

ResNet is a deep neural network architecture that uses residual connections to enable the training of very deep networks. The ResNet18 model consisted of several convolutional layers, each followed by a batch normalization layer and a ReLU activation function, and several residual blocks, each consisting of two convolutional layers and a shortcut connection.

We used PyTorch to implement the model and trained it using the cross-entropy loss function and the SGD optimizer with a learning rate scheduler.

In [1]:
```
# import library dependencies
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
```

In [2]:
```
# check if GPU is available

if torch.cuda.is_available():
    print('Training on GPU!')
    device = torch.device('cuda')
elif torch.has_mps:
    print('Training on Macbook Metal GPU!')
    device = torch.device('mps')
else:
    print('No GPU available. Training on CPU!')
    device = torch.device('cpu')

device
```

```
Training on Macbook Metal GPU!
```

Out[2]:
```
device(type='mps')
```

Import Data

In [3]:
```
ROOT_PATH='../'
```

In [4]:
```
BATCH_SIZE=16
```

In [5]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

In [6]:
```
train_dataset = CIFAR10(root=ROOT_PATH, download=True, train=True, transform
eval_dataset = CIFAR10(root=ROOT_PATH, train=False, transform=transform)
```

```
Files already downloaded and verified
```

Preprocess Data

In [7]:
```
train_data_loader = DataLoader(dataset=train_dataset, num_workers=4, batch_s
eval_data_loader = DataLoader(dataset=eval_dataset, num_workers=4, batch_siz
```

Define model and train

In [8]:
```
class BasicBlock(nn.Module):
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stri
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padd
        self.bn2 = nn.BatchNorm2d(planes)

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

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

In [9]:
```
class ResNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bi
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = nn.Sequential(
            BasicBlock(64, 64, stride=1),
            BasicBlock(64, 64, stride=1)
        )
        self.layer2 = nn.Sequential(
            BasicBlock(64, 128, stride=2),
            BasicBlock(128, 128, stride=1)
        )
        self.layer3 = nn.Sequential(
            BasicBlock(128, 256, stride=2),
            BasicBlock(256, 256, stride=1)
        )
        self.layer4 = nn.Sequential(
            BasicBlock(256, 512, stride=2),
            BasicBlock(512, 512, stride=1)
        )
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512, num_classes)

    def forward(self, x):
        out = nn.functional.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

In [10]:
```
# create an instance of the ResNet class
net = ResNet()
net.to(device)
```

Out[10]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
```

```
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1
, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
```

```
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (shortcut): Sequential()
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (linear): Linear(in_features=512, out_features=10, bias=True)
)
```

In [11]:

```
# define the loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [12]:

```
num_epochs = 50
```

In [13]:
```
# train the ResNet
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_data_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 1000 == 999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  1000] loss: 2.102
[1,  2000] loss: 1.806
[1,  3000] loss: 1.663
[1,  4000] loss: 1.542
[1,  5000] loss: 1.433
[1,  6000] loss: 1.340
[1,  7000] loss: 1.289
[1,  8000] loss: 1.189
[1,  9000] loss: 1.137
[1, 10000] loss: 1.072
[1, 11000] loss: 1.065
[1, 12000] loss: 1.008
[2,  1000] loss: 0.918
[2,  2000] loss: 0.893
[2,  3000] loss: 0.869
[2,  4000] loss: 0.860
[2,  5000] loss: 0.834
[2,  6000] loss: 0.827
[2,  7000] loss: 0.785
[2,  8000] loss: 0.763
[2,  9000] loss: 0.785
[2, 10000] loss: 0.775
[2, 11000] loss: 0.770
[2, 12000] loss: 0.733
[3,  1000] loss: 0.586
[3,  2000] loss: 0.598
[3,  3000] loss: 0.623
[3,  4000] loss: 0.611
[3,  5000] loss: 0.594
[3,  6000] loss: 0.616
```

```
[3,  7000] loss: 0.607
[3,  8000] loss: 0.589
[3,  9000] loss: 0.642
[3, 10000] loss: 0.566
[3, 11000] loss: 0.562
[3, 12000] loss: 0.576
[4,  1000] loss: 0.444
[4,  2000] loss: 0.444
[4,  3000] loss: 0.439
[4,  4000] loss: 0.461
[4,  5000] loss: 0.449
[4,  6000] loss: 0.462
[4,  7000] loss: 0.477
[4,  8000] loss: 0.473
[4,  9000] loss: 0.460
[4, 10000] loss: 0.471
[4, 11000] loss: 0.454
[4, 12000] loss: 0.475
[5,  1000] loss: 0.312
[5,  2000] loss: 0.337
[5,  3000] loss: 0.322
[5,  4000] loss: 0.351
[5,  5000] loss: 0.362
[5,  6000] loss: 0.326
[5,  7000] loss: 0.339
[5,  8000] loss: 0.353
[5,  9000] loss: 0.330
[5, 10000] loss: 0.361
[5, 11000] loss: 0.363
[5, 12000] loss: 0.382
[6,  1000] loss: 0.226
[6,  2000] loss: 0.230
[6,  3000] loss: 0.230
[6,  4000] loss: 0.247
[6,  5000] loss: 0.256
[6,  6000] loss: 0.258
[6,  7000] loss: 0.265
[6,  8000] loss: 0.284
[6,  9000] loss: 0.268
[6, 10000] loss: 0.275
[6, 11000] loss: 0.273
[6, 12000] loss: 0.259
[7,  1000] loss: 0.157
[7,  2000] loss: 0.150
[7,  3000] loss: 0.161
[7,  4000] loss: 0.164
[7,  5000] loss: 0.160
[7,  6000] loss: 0.173
[7,  7000] loss: 0.187
[7,  8000] loss: 0.194
[7,  9000] loss: 0.195
[7, 10000] loss: 0.190
[7, 11000] loss: 0.195
[7, 12000] loss: 0.210
[8,  1000] loss: 0.109
[8,  2000] loss: 0.097
[8,  3000] loss: 0.099
```

```
[8,  4000] loss: 0.111
[8,  5000] loss: 0.112
[8,  6000] loss: 0.122
[8,  7000] loss: 0.132
[8,  8000] loss: 0.129
[8,  9000] loss: 0.121
[8, 10000] loss: 0.131
[8, 11000] loss: 0.134
[8, 12000] loss: 0.146
[9,  1000] loss: 0.071
[9,  2000] loss: 0.072
[9,  3000] loss: 0.064
[9,  4000] loss: 0.079
[9,  5000] loss: 0.081
[9,  6000] loss: 0.102
[9,  7000] loss: 0.097
[9,  8000] loss: 0.090
[9,  9000] loss: 0.079
[9, 10000] loss: 0.105
[9, 11000] loss: 0.081
[9, 12000] loss: 0.089
[10,  1000] loss: 0.057
[10,  2000] loss: 0.060
[10,  3000] loss: 0.051
[10,  4000] loss: 0.049
[10,  5000] loss: 0.063
[10,  6000] loss: 0.055
[10,  7000] loss: 0.056
[10,  8000] loss: 0.072
[10,  9000] loss: 0.070
[10, 10000] loss: 0.070
[10, 11000] loss: 0.058
[10, 12000] loss: 0.055
[11,  1000] loss: 0.047
[11,  2000] loss: 0.032
[11,  3000] loss: 0.035
[11,  4000] loss: 0.037
[11,  5000] loss: 0.043
[11,  6000] loss: 0.049
[11,  7000] loss: 0.056
[11,  8000] loss: 0.055
[11,  9000] loss: 0.052
[11, 10000] loss: 0.060
[11, 11000] loss: 0.059
[11, 12000] loss: 0.053
[12,  1000] loss: 0.038
[12,  2000] loss: 0.035
[12,  3000] loss: 0.032
[12,  4000] loss: 0.039
[12,  5000] loss: 0.034
[12,  6000] loss: 0.030
[12,  7000] loss: 0.034
[12,  8000] loss: 0.037
[12,  9000] loss: 0.038
[12, 10000] loss: 0.049
[12, 11000] loss: 0.045
[12, 12000] loss: 0.043
```

```
[13,  1000] loss: 0.023
[13,  2000] loss: 0.019
[13,  3000] loss: 0.020
[13,  4000] loss: 0.026
[13,  5000] loss: 0.026
[13,  6000] loss: 0.032
[13,  7000] loss: 0.025
[13,  8000] loss: 0.018
[13,  9000] loss: 0.022
[13, 10000] loss: 0.023
[13, 11000] loss: 0.028
[13, 12000] loss: 0.026
[14,  1000] loss: 0.016
[14,  2000] loss: 0.015
[14,  3000] loss: 0.017
[14,  4000] loss: 0.014
[14,  5000] loss: 0.019
[14,  6000] loss: 0.018
[14,  7000] loss: 0.013
[14,  8000] loss: 0.013
[14,  9000] loss: 0.018
[14, 10000] loss: 0.018
[14, 11000] loss: 0.024
[14, 12000] loss: 0.023
[15,  1000] loss: 0.016
[15,  2000] loss: 0.013
[15,  3000] loss: 0.011
[15,  4000] loss: 0.011
[15,  5000] loss: 0.011
[15,  6000] loss: 0.015
[15,  7000] loss: 0.010
[15,  8000] loss: 0.010
[15,  9000] loss: 0.013
[15, 10000] loss: 0.013
[15, 11000] loss: 0.011
[15, 12000] loss: 0.013
[16,  1000] loss: 0.012
[16,  2000] loss: 0.011
[16,  3000] loss: 0.010
[16,  4000] loss: 0.013
[16,  5000] loss: 0.007
[16,  6000] loss: 0.010
[16,  7000] loss: 0.007
[16,  8000] loss: 0.013
[16,  9000] loss: 0.013
[16, 10000] loss: 0.015
[16, 11000] loss: 0.020
[16, 12000] loss: 0.013
[17,  1000] loss: 0.005
[17,  2000] loss: 0.004
[17,  3000] loss: 0.008
[17,  4000] loss: 0.006
[17,  5000] loss: 0.009
[17,  6000] loss: 0.009
[17,  7000] loss: 0.009
[17,  8000] loss: 0.005
[17,  9000] loss: 0.004
```

```
[17, 10000] loss: 0.005
[17, 11000] loss: 0.009
[17, 12000] loss: 0.006
[18,  1000] loss: 0.006
[18,  2000] loss: 0.004
[18,  3000] loss: 0.005
[18,  4000] loss: 0.006
[18,  5000] loss: 0.006
[18,  6000] loss: 0.007
[18,  7000] loss: 0.007
[18,  8000] loss: 0.005
[18,  9000] loss: 0.007
[18, 10000] loss: 0.007
[18, 11000] loss: 0.008
[18, 12000] loss: 0.005
[19,  1000] loss: 0.005
[19,  2000] loss: 0.004
[19,  3000] loss: 0.005
[19,  4000] loss: 0.006
[19,  5000] loss: 0.004
[19,  6000] loss: 0.006
[19,  7000] loss: 0.004
[19,  8000] loss: 0.002
[19,  9000] loss: 0.003
[19, 10000] loss: 0.002
[19, 11000] loss: 0.005
[19, 12000] loss: 0.004
[20,  1000] loss: 0.003
[20,  2000] loss: 0.004
[20,  3000] loss: 0.003
[20,  4000] loss: 0.003
[20,  5000] loss: 0.003
[20,  6000] loss: 0.007
[20,  7000] loss: 0.005
[20,  8000] loss: 0.004
[20,  9000] loss: 0.002
[20, 10000] loss: 0.004
[20, 11000] loss: 0.003
[20, 12000] loss: 0.005
[21,  1000] loss: 0.003
[21,  2000] loss: 0.003
[21,  3000] loss: 0.002
[21,  4000] loss: 0.002
[21,  5000] loss: 0.002
[21,  6000] loss: 0.002
[21,  7000] loss: 0.001
[21,  8000] loss: 0.001
[21,  9000] loss: 0.002
[21, 10000] loss: 0.002
[21, 11000] loss: 0.002
[21, 12000] loss: 0.002
[22,  1000] loss: 0.002
[22,  2000] loss: 0.002
[22,  3000] loss: 0.001
[22,  4000] loss: 0.002
[22,  5000] loss: 0.004
[22,  6000] loss: 0.006
```

```
[22,  7000] loss: 0.003
[22,  8000] loss: 0.002
[22,  9000] loss: 0.004
[22, 10000] loss: 0.005
[22, 11000] loss: 0.003
[22, 12000] loss: 0.004
[23,  1000] loss: 0.003
[23,  2000] loss: 0.002
[23,  3000] loss: 0.003
[23,  4000] loss: 0.006
[23,  5000] loss: 0.003
[23,  6000] loss: 0.002
[23,  7000] loss: 0.004
[23,  8000] loss: 0.003
[23,  9000] loss: 0.003
[23, 10000] loss: 0.002
[23, 11000] loss: 0.002
[23, 12000] loss: 0.002
[24,  1000] loss: 0.001
[24,  2000] loss: 0.001
[24,  3000] loss: 0.001
[24,  4000] loss: 0.001
[24,  5000] loss: 0.001
[24,  6000] loss: 0.001
[24,  7000] loss: 0.001
[24,  8000] loss: 0.001
[24,  9000] loss: 0.002
[24, 10000] loss: 0.001
[24, 11000] loss: 0.002
[24, 12000] loss: 0.001
[25,  1000] loss: 0.001
[25,  2000] loss: 0.002
[25,  3000] loss: 0.001
[25,  4000] loss: 0.001
[25,  5000] loss: 0.002
[25,  6000] loss: 0.001
[25,  7000] loss: 0.001
[25,  8000] loss: 0.001
[25,  9000] loss: 0.001
[25, 10000] loss: 0.001
[25, 11000] loss: 0.001
[25, 12000] loss: 0.001
[26,  1000] loss: 0.001
[26,  2000] loss: 0.000
[26,  3000] loss: 0.001
[26,  4000] loss: 0.001
[26,  5000] loss: 0.001
[26,  6000] loss: 0.001
[26,  7000] loss: 0.001
[26,  8000] loss: 0.001
[26,  9000] loss: 0.001
[26, 10000] loss: 0.001
[26, 11000] loss: 0.001
[26, 12000] loss: 0.001
[27,  1000] loss: 0.001
[27,  2000] loss: 0.001
[27,  3000] loss: 0.001
```

```
[27,  4000] loss: 0.000
[27,  5000] loss: 0.000
[27,  6000] loss: 0.000
[27,  7000] loss: 0.000
[27,  8000] loss: 0.001
[27,  9000] loss: 0.001
[27, 10000] loss: 0.000
[27, 11000] loss: 0.000
[27, 12000] loss: 0.001
[28,  1000] loss: 0.001
[28,  2000] loss: 0.001
[28,  3000] loss: 0.001
[28,  4000] loss: 0.001
[28,  5000] loss: 0.001
[28,  6000] loss: 0.000
[28,  7000] loss: 0.000
[28,  8000] loss: 0.000
[28,  9000] loss: 0.000
[28, 10000] loss: 0.001
[28, 11000] loss: 0.001
[28, 12000] loss: 0.001
[29,  1000] loss: 0.001
[29,  2000] loss: 0.001
[29,  3000] loss: 0.001
[29,  4000] loss: 0.001
[29,  5000] loss: 0.001
[29,  6000] loss: 0.000
[29,  7000] loss: 0.000
[29,  8000] loss: 0.000
[29,  9000] loss: 0.001
[29, 10000] loss: 0.001
[29, 11000] loss: 0.000
[29, 12000] loss: 0.000
[30,  1000] loss: 0.000
[30,  2000] loss: 0.000
[30,  3000] loss: 0.001
[30,  4000] loss: 0.000
[30,  5000] loss: 0.001
[30,  6000] loss: 0.000
[30,  7000] loss: 0.000
[30,  8000] loss: 0.001
[30,  9000] loss: 0.000
[30, 10000] loss: 0.001
[30, 11000] loss: 0.000
[30, 12000] loss: 0.000
[31,  1000] loss: 0.000
[31,  2000] loss: 0.000
[31,  3000] loss: 0.000
[31,  4000] loss: 0.001
[31,  5000] loss: 0.001
[31,  6000] loss: 0.001
[31,  7000] loss: 0.000
[31,  8000] loss: 0.001
[31,  9000] loss: 0.000
[31, 10000] loss: 0.000
[31, 11000] loss: 0.001
[31, 12000] loss: 0.000
```

```
[32,  1000] loss: 0.000
[32,  2000] loss: 0.000
[32,  3000] loss: 0.001
[32,  4000] loss: 0.000
[32,  5000] loss: 0.000
[32,  6000] loss: 0.001
[32,  7000] loss: 0.000
[32,  8000] loss: 0.000
[32,  9000] loss: 0.000
[32, 10000] loss: 0.000
[32, 11000] loss: 0.000
[32, 12000] loss: 0.001
[33,  1000] loss: 0.000
[33,  2000] loss: 0.000
[33,  3000] loss: 0.000
[33,  4000] loss: 0.000
[33,  5000] loss: 0.000
[33,  6000] loss: 0.001
[33,  7000] loss: 0.000
[33,  8000] loss: 0.001
[33,  9000] loss: 0.001
[33, 10000] loss: 0.001
[33, 11000] loss: 0.000
[33, 12000] loss: 0.000
[34,  1000] loss: 0.000
[34,  2000] loss: 0.001
[34,  3000] loss: 0.000
[34,  4000] loss: 0.000
[34,  5000] loss: 0.000
[34,  6000] loss: 0.001
[34,  7000] loss: 0.000
[34,  8000] loss: 0.000
[34,  9000] loss: 0.000
[34, 10000] loss: 0.000
[34, 11000] loss: 0.000
[34, 12000] loss: 0.000
[35,  1000] loss: 0.000
[35,  2000] loss: 0.000
[35,  3000] loss: 0.000
[35,  4000] loss: 0.000
[35,  5000] loss: 0.001
[35,  6000] loss: 0.000
[35,  7000] loss: 0.000
[35,  8000] loss: 0.000
[35,  9000] loss: 0.000
[35, 10000] loss: 0.000
[35, 11000] loss: 0.000
[35, 12000] loss: 0.000
[36,  1000] loss: 0.001
[36,  2000] loss: 0.000
[36,  3000] loss: 0.000
[36,  4000] loss: 0.000
[36,  5000] loss: 0.000
[36,  6000] loss: 0.000
[36,  7000] loss: 0.000
[36,  8000] loss: 0.000
[36,  9000] loss: 0.000
```

```
[36, 10000] loss: 0.000
[36, 11000] loss: 0.000
[36, 12000] loss: 0.000
[37,  1000] loss: 0.000
[37,  2000] loss: 0.000
[37,  3000] loss: 0.000
[37,  4000] loss: 0.001
[37,  5000] loss: 0.000
[37,  6000] loss: 0.000
[37,  7000] loss: 0.000
[37,  8000] loss: 0.000
[37,  9000] loss: 0.000
[37, 10000] loss: 0.000
[37, 11000] loss: 0.000
[37, 12000] loss: 0.000
[38,  1000] loss: 0.000
[38,  2000] loss: 0.000
[38,  3000] loss: 0.000
[38,  4000] loss: 0.000
[38,  5000] loss: 0.000
[38,  6000] loss: 0.000
[38,  7000] loss: 0.000
[38,  8000] loss: 0.000
[38,  9000] loss: 0.000
[38, 10000] loss: 0.000
[38, 11000] loss: 0.000
[38, 12000] loss: 0.000
[39,  1000] loss: 0.000
[39,  2000] loss: 0.000
[39,  3000] loss: 0.000
[39,  4000] loss: 0.000
[39,  5000] loss: 0.000
[39,  6000] loss: 0.000
[39,  7000] loss: 0.000
[39,  8000] loss: 0.000
[39,  9000] loss: 0.000
[39, 10000] loss: 0.000
[39, 11000] loss: 0.000
[39, 12000] loss: 0.000
[40,  1000] loss: 0.000
[40,  2000] loss: 0.000
[40,  3000] loss: 0.000
[40,  4000] loss: 0.000
[40,  5000] loss: 0.000
[40,  6000] loss: 0.000
[40,  7000] loss: 0.000
[40,  8000] loss: 0.000
[40,  9000] loss: 0.000
[40, 10000] loss: 0.000
[40, 11000] loss: 0.000
[40, 12000] loss: 0.000
[41,  1000] loss: 0.000
[41,  2000] loss: 0.000
[41,  3000] loss: 0.000
[41,  4000] loss: 0.000
[41,  5000] loss: 0.000
[41,  6000] loss: 0.000
```

```
[41,  7000] loss: 0.000
[41,  8000] loss: 0.000
[41,  9000] loss: 0.000
[41, 10000] loss: 0.000
[41, 11000] loss: 0.000
[41, 12000] loss: 0.000
[42,  1000] loss: 0.000
[42,  2000] loss: 0.000
[42,  3000] loss: 0.000
[42,  4000] loss: 0.000
[42,  5000] loss: 0.000
[42,  6000] loss: 0.000
[42,  7000] loss: 0.000
[42,  8000] loss: 0.000
[42,  9000] loss: 0.000
[42, 10000] loss: 0.000
[42, 11000] loss: 0.000
[42, 12000] loss: 0.000
[43,  1000] loss: 0.000
[43,  2000] loss: 0.000
[43,  3000] loss: 0.000
[43,  4000] loss: 0.000
[43,  5000] loss: 0.000
[43,  6000] loss: 0.000
[43,  7000] loss: 0.000
[43,  8000] loss: 0.000
[43,  9000] loss: 0.000
[43, 10000] loss: 0.000
[43, 11000] loss: 0.000
[43, 12000] loss: 0.000
[44,  1000] loss: 0.000
[44,  2000] loss: 0.000
[44,  3000] loss: 0.000
[44,  4000] loss: 0.000
[44,  5000] loss: 0.000
[44,  6000] loss: 0.000
[44,  7000] loss: 0.000
[44,  8000] loss: 0.000
[44,  9000] loss: 0.000
[44, 10000] loss: 0.000
[44, 11000] loss: 0.000
[44, 12000] loss: 0.000
[45,  1000] loss: 0.000
[45,  2000] loss: 0.000
[45,  3000] loss: 0.000
[45,  4000] loss: 0.000
[45,  5000] loss: 0.000
[45,  6000] loss: 0.000
[45,  7000] loss: 0.000
[45,  8000] loss: 0.000
[45,  9000] loss: 0.000
[45, 10000] loss: 0.000
[45, 11000] loss: 0.000
[45, 12000] loss: 0.000
[46,  1000] loss: 0.001
[46,  2000] loss: 0.000
[46,  3000] loss: 0.000
```

```
[46,  4000] loss: 0.000
[46,  5000] loss: 0.000
[46,  6000] loss: 0.000
[46,  7000] loss: 0.000
[46,  8000] loss: 0.000
[46,  9000] loss: 0.000
[46, 10000] loss: 0.000
[46, 11000] loss: 0.000
[46, 12000] loss: 0.000
[47,  1000] loss: 0.000
[47,  2000] loss: 0.000
[47,  3000] loss: 0.000
[47,  4000] loss: 0.000
[47,  5000] loss: 0.000
[47,  6000] loss: 0.000
[47,  7000] loss: 0.000
[47,  8000] loss: 0.000
[47,  9000] loss: 0.000
[47, 10000] loss: 0.000
[47, 11000] loss: 0.000
[47, 12000] loss: 0.000
[48,  1000] loss: 0.000
[48,  2000] loss: 0.000
[48,  3000] loss: 0.000
[48,  4000] loss: 0.000
[48,  5000] loss: 0.000
[48,  6000] loss: 0.001
[48,  7000] loss: 0.001
[48,  8000] loss: 0.000
[48,  9000] loss: 0.000
[48, 10000] loss: 0.000
[48, 11000] loss: 0.000
[48, 12000] loss: 0.000
[49,  1000] loss: 0.000
[49,  2000] loss: 0.000
[49,  3000] loss: 0.000
[49,  4000] loss: 0.000
[49,  5000] loss: 0.000
[49,  6000] loss: 0.001
[49,  7000] loss: 0.000
[49,  8000] loss: 0.000
[49,  9000] loss: 0.000
[49, 10000] loss: 0.000
[49, 11000] loss: 0.000
[49, 12000] loss: 0.000
[50,  1000] loss: 0.000
[50,  2000] loss: 0.000
[50,  3000] loss: 0.000
[50,  4000] loss: 0.000
[50,  5000] loss: 0.000
[50,  6000] loss: 0.000
[50,  7000] loss: 0.000
[50,  8000] loss: 0.000
[50,  9000] loss: 0.000
[50, 10000] loss: 0.000
[50, 11000] loss: 0.000
[50, 12000] loss: 0.000
```

```
Finished Training
```

Save and load model

In [ ]:
```
# save the trained model
torch.save(net, 'resnet.pt')
```

In [ ]:
```
# load the saved model
net = torch.load('resnet.pt')
```

Evaluate the model

In [14]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in eval_data_loader:
        images, labels = data
        images = images.to(device)
        labels = 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 %d test images: %d %%' % (len(eval_dat
```

```
Accuracy of the network on the 10000 test images: 85 %
```

In [15]:
```
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 's
```

In [16]:
```
class_correct = [0] * 10
class_total = [0] * 10
with torch.no_grad():
    for data in eval_data_loader:
        images, labels = data
        images = images.to(device)
        labels = 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 plane : 89 %
Accuracy of   car : 93 %
Accuracy of  bird : 78 %
Accuracy of   cat : 71 %
Accuracy of  deer : 81 %
Accuracy of   dog : 76 %
Accuracy of  frog : 88 %
Accuracy of horse : 88 %
Accuracy of  ship : 91 %
Accuracy of truck : 91 %
```

In [17]:

```
TP = 0
FP = 0
TN = 0
FN = 0

with torch.no_grad():
    for data in eval_data_loader:
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        for i in range(len(labels)):
            if predicted[i] == labels[i]:
                if predicted[i] == 1:
                    TP += 1
                else:
                    TN += 1
            else:
                if predicted[i] == 1:
                    FP += 1
                else:
                    FN += 1

accuracy = 100 * (TP + TN) / (TP + TN + FP + FN)
precision = 100 * TP / (TP + FP)
recall = 100 * TP / (TP + FN)
f1_score = 2 * precision * recall / (precision + recall)

print('Accuracy: %.2f %%' % (accuracy))
print('Precision: %.2f %%' % (precision))
print('Recall: %.2f %%' % (recall))
print('F1 Score: %.2f %%' % (f1_score))
```

```
Accuracy: 85.11 %
Precision: 93.62 %
Recall: 39.72 %
F1 Score: 55.78 %
```