In [1]:
```
# Using ResNet-18 for image classification
# Data:
    # cinic10 --> https://www.kaggle.com/mengcius/cinic10
    # a subset of cinic10 containing only four classes - airplane, automobile,
ship, and truck - is used
        # files from cinic10 'train' and 'val' folders are combined into one t
o provide 18000 training images from each class
        # test folders contain 9000 images from each class

# Note: here GPU processing is used --> running times with CPU would be greate
r
```

In [2]:
```
# import libraries

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
from torchvision.utils import make_grid
import os

import numpy as np
import pandas as pd
import seaborn as sns  # for heatmaps
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style = "whitegrid", font_scale = 1.2)

# ignore non-critical warnings
import warnings
warnings.filterwarnings("ignore")

# check if GPU computing with CUDA is available and set the device accordingly
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [3]:
```
# define transforms

cinic_mean = [0.4789, 0.4723, 0.4305] # from cinic10 'Readme' file
cinic_std = [0.2421, 0.2383, 0.2587] # from cinic10 'Readme' file

# we are using the same transforms for both train and test images
img_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean = cinic_mean, std = cinic_std)
])
```

In [4]:
```
# prepare train and test sets, loaders

# define root directory path
root = 'datasets/cinic10_classes4'

# batch size
b_size = 16

train_data = datasets.ImageFolder(os.path.join(root, 'train'), transform = img
_transform)
test_data = datasets.ImageFolder(os.path.join(root, 'test'), transform = img_t
ransform)

torch.manual_seed(42)
train_loader = DataLoader(train_data, batch_size = b_size, shuffle = True)
test_loader = DataLoader(test_data, batch_size = b_size, shuffle = True)

class_names = train_data.classes

print(class_names)
print(f'Training images available: {len(train_data)}')
print(f'Testing images available:  {len(test_data)}')
```

```
['airplane', 'automobile', 'ship', 'truck']
Training images available: 72000
Testing images available:  36000
```

In [5]:
```
# display a batch of images --> grab first batch of 16 images from train data

for images,labels in train_loader:
    break

images.shape
```

Out[5]:
```
torch.Size([16, 3, 32, 32])
```

In [6]:
```
# 16 images in one batch, 3 color channels (R-G-B), 32 x 32 pixels
```

In [7]:
```
# show images and their classes

# print labels
print('Label:', labels.numpy())
print('Class:', *np.array([class_names[i] for i in labels]))

im = make_grid(images)  # the default nrow is 8

# inverse normalize the images

inv_normalize = transforms.Normalize(
    mean=[-0.4789/0.2421, -0.4723/0.2383, -0.4305/0.2587],
    std=[1/0.2421, 1/0.2383, 1/0.2587]
)
im_inv = inv_normalize(im)

# print images
plt.figure(figsize=(16,6))
plt.imshow(np.transpose(im_inv.numpy(), (1, 2, 0)));
```

```
Label: [1 1 1 1 1 3 2 2 1 1 1 1 0 0 3 0]
Class: automobile automobile automobile automobile automobile truck ship ship
automobile automobile automobile automobile airplane airplane truck airplane
```


In [8]:
```
# everything works fine!
```

In [9]:
```
# select resnet18 model from torchvision models
ResNet18model = models.resnet18()
```

In [10]:
```
# display structure of resnet18
ResNet18model
```

Out[10]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
ias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_m
ode=False)
  (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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_stats=True)
    )
    (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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [11]:
```
# modify last layer of model to have 4 outputs --> we work with four classes o
nly
torch.manual_seed(42)

ResNet18model.fc = nn.Sequential(nn.Linear(512, 4),
                                 nn.LogSoftmax(dim=1))

ResNet18model.to(device) # set device
```

Out[11]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
ias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_m
ode=False)
  (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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_stats=True)
    )
    (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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=4, bias=True)
    (1): LogSoftmax()
  )
)
```

In [12]:
```
# last layer of model has been modified to meet our needs
```

In [13]:
```
# count model parameters

def count_parameters(model):
    params = [p.numel() for p in model.parameters() if p.requires_grad]
    for item in params:
        print(f'{item:>6}')
    print(f'______\n{sum(params):>6}')

count_parameters(ResNet18model)
```

```
  9408
    64
    64
 36864
    64
    64
 36864
    64
    64
 36864
    64
    64
 36864
    64
    64
 73728
   128
   128
147456
   128
   128
  8192
   128
   128
147456
   128
   128
147456
   128
   128
294912
   256
   256
589824
   256
   256
 32768
   256
   256
589824
   256
   256
589824
   256
   256
1179648
   512
   512
2359296
   512
   512
131072
   512
   512
2359296
   512
   512
```

```
2359296
    512
    512
   2048
      4
_______
11178564
```

In [14]:
```
# approximately 11 million parameters which is not so large of a number
```

In [15]:
```
# define loss function and optimizer

# define loss function and set device
criterion = nn.CrossEntropyLoss().to(device)

# optimizer needs to be defined after device for model is selected --> model p
arameters with different devices are different!
optimizer = torch.optim.Adam(ResNet18model.parameters(), lr = 0.001)
```

In [16]:
```
# start training

import time
start_time = time.time()

epochs = 40

# limit number of train and test images --> optional - saves time to run with
 small numbers first to see if model works
max_trn_batch = 1000
max_tst_batch = 500

# instantiate trackers for model performance
train_losses = []
test_losses = []
train_correct = []
test_correct = []

for i in range(epochs):
    trn_corr = 0
    tst_corr = 0

    # run the training batches
    for b, (X_train, y_train) in enumerate(train_loader):

        # limit the number of batches
        if b == max_trn_batch:
            break
        b+=1

        # apply the model
        y_pred = ResNet18model(X_train.to(device))
        loss = criterion(y_pred, y_train.to(device))

        # tally the number of correct predictions
        predicted = torch.max(y_pred.data, 1)[1]
        batch_corr = (predicted == y_train.to(device)).sum()
        trn_corr += batch_corr

        # update parameters
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # print interim results
        if b%max_trn_batch == 0:
            print(f'Epoch: {i+1:3}   Batch: {b:4}   Loss: {loss.item():10.4f}
Accuracy: {trn_corr.item()*100/(b_size*b):7.2f}% \
 Time Elapsed Since Start: {(time.time() - start_time)/60:7.2f} min')

    train_losses.append(loss)
    train_correct.append(trn_corr)

    # run the test batches
    with torch.no_grad():
        for b, (X_test, y_test) in enumerate(test_loader):
```

```
            # limit the number of batches
            if b == max_tst_batch:
                break

            # apply the model
            y_val = ResNet18model(X_test.to(device))

            # tally the number of correct predictions
            predicted = torch.max(y_val.data, 1)[1]
            tst_corr += (predicted == y_test.to(device)).sum()

    loss = criterion(y_val, y_test.to(device))
    test_losses.append(loss)
    test_correct.append(tst_corr)
```

```
Epoch:   1   Batch: 1000   Loss:     0.9393   Accuracy:   49.15%  Time Elapse
d Since Start:    0.73 min
Epoch:   2   Batch: 1000   Loss:     1.0233   Accuracy:   56.04%  Time Elapse
d Since Start:    1.53 min
Epoch:   3   Batch: 1000   Loss:     0.9782   Accuracy:   61.43%  Time Elapse
d Since Start:    2.33 min
Epoch:   4   Batch: 1000   Loss:     1.3356   Accuracy:   63.71%  Time Elapse
d Since Start:    3.14 min
Epoch:   5   Batch: 1000   Loss:     0.6735   Accuracy:   66.08%  Time Elapse
d Since Start:    3.94 min
Epoch:   6   Batch: 1000   Loss:     0.9234   Accuracy:   67.83%  Time Elapse
d Since Start:    4.82 min
Epoch:   7   Batch: 1000   Loss:     0.6212   Accuracy:   70.58%  Time Elapse
d Since Start:    5.69 min
Epoch:   8   Batch: 1000   Loss:     0.4811   Accuracy:   71.34%  Time Elapse
d Since Start:    6.54 min
Epoch:   9   Batch: 1000   Loss:     0.8843   Accuracy:   72.24%  Time Elapse
d Since Start:    7.38 min
Epoch:  10   Batch: 1000   Loss:     0.3796   Accuracy:   74.07%  Time Elapse
d Since Start:    8.21 min
Epoch:  11   Batch: 1000   Loss:     1.0706   Accuracy:   74.71%  Time Elapse
d Since Start:    9.04 min
Epoch:  12   Batch: 1000   Loss:     0.6988   Accuracy:   75.54%  Time Elapse
d Since Start:    9.86 min
Epoch:  13   Batch: 1000   Loss:     0.6407   Accuracy:   76.58%  Time Elapse
d Since Start:   10.69 min
Epoch:  14   Batch: 1000   Loss:     0.4520   Accuracy:   77.59%  Time Elapse
d Since Start:   11.52 min
Epoch:  15   Batch: 1000   Loss:     0.5094   Accuracy:   77.51%  Time Elapse
d Since Start:   12.35 min
Epoch:  16   Batch: 1000   Loss:     0.5058   Accuracy:   78.47%  Time Elapse
d Since Start:   13.17 min
Epoch:  17   Batch: 1000   Loss:     0.9032   Accuracy:   78.99%  Time Elapse
d Since Start:   13.98 min
Epoch:  18   Batch: 1000   Loss:     1.0348   Accuracy:   80.16%  Time Elapse
d Since Start:   14.79 min
Epoch:  19   Batch: 1000   Loss:     0.6745   Accuracy:   79.69%  Time Elapse
d Since Start:   15.60 min
Epoch:  20   Batch: 1000   Loss:     0.6603   Accuracy:   81.23%  Time Elapse
d Since Start:   16.42 min
Epoch:  21   Batch: 1000   Loss:     0.2936   Accuracy:   82.04%  Time Elapse
d Since Start:   17.23 min
Epoch:  22   Batch: 1000   Loss:     0.5107   Accuracy:   81.99%  Time Elapse
d Since Start:   18.04 min
Epoch:  23   Batch: 1000   Loss:     0.9354   Accuracy:   83.46%  Time Elapse
d Since Start:   18.85 min
Epoch:  24   Batch: 1000   Loss:     0.3793   Accuracy:   83.41%  Time Elapse
d Since Start:   19.65 min
Epoch:  25   Batch: 1000   Loss:     0.4244   Accuracy:   84.31%  Time Elapse
d Since Start:   20.46 min
Epoch:  26   Batch: 1000   Loss:     0.2853   Accuracy:   84.51%  Time Elapse
d Since Start:   21.30 min
Epoch:  27   Batch: 1000   Loss:     0.9477   Accuracy:   85.10%  Time Elapse
d Since Start:   22.12 min
Epoch:  28   Batch: 1000   Loss:     0.1592   Accuracy:   85.32%  Time Elapse
d Since Start:   22.92 min
Epoch:  29   Batch: 1000   Loss:     0.2434   Accuracy:   86.53%  Time Elapse
```

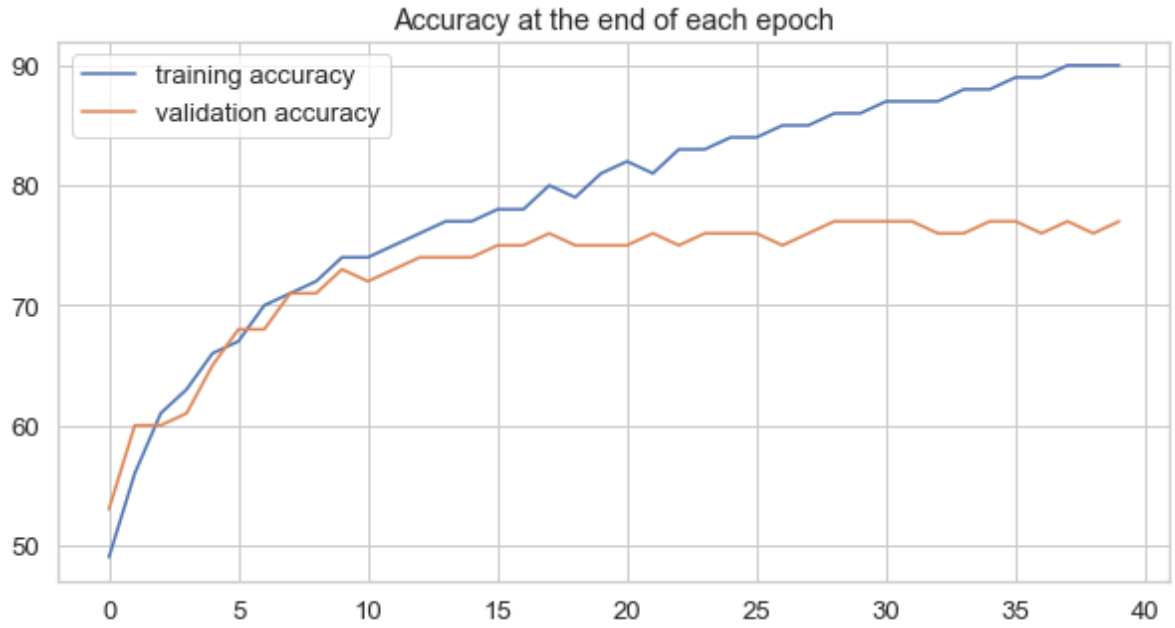
```
d Since Start:   23.73 min
Epoch:  30   Batch: 1000   Loss:     0.2986   Accuracy:   86.85%  Time Elapse
d Since Start:   24.53 min
Epoch:  31   Batch: 1000   Loss:     0.7745   Accuracy:   87.21%  Time Elapse
d Since Start:   25.34 min
Epoch:  32   Batch: 1000   Loss:     0.3200   Accuracy:   87.58%  Time Elapse
d Since Start:   26.14 min
Epoch:  33   Batch: 1000   Loss:     0.6090   Accuracy:   87.52%  Time Elapse
d Since Start:   26.95 min
Epoch:  34   Batch: 1000   Loss:     0.1243   Accuracy:   88.46%  Time Elapse
d Since Start:   27.75 min
Epoch:  35   Batch: 1000   Loss:     0.2393   Accuracy:   88.78%  Time Elapse
d Since Start:   28.57 min
Epoch:  36   Batch: 1000   Loss:     0.3746   Accuracy:   89.17%  Time Elapse
d Since Start:   29.38 min
Epoch:  37   Batch: 1000   Loss:     0.1953   Accuracy:   89.42%  Time Elapse
d Since Start:   30.20 min
Epoch:  38   Batch: 1000   Loss:     0.0828   Accuracy:   90.14%  Time Elapse
d Since Start:   31.01 min
Epoch:  39   Batch: 1000   Loss:     0.2272   Accuracy:   90.23%  Time Elapse
d Since Start:   31.82 min
Epoch:  40   Batch: 1000   Loss:     0.2472   Accuracy:   90.61%  Time Elapse
d Since Start:   32.62 min
```

In [17]:
```
# save the trained model

torch.save(ResNet18model.state_dict(), 'resnet18cinic10cl4.pt')
```

In [18]:
```
# evaluate model performance

# plot losses
plt.figure(figsize = (10,5))
plt.plot(train_losses, label='training loss')
plt.plot(test_losses, label='validation loss')
plt.title('Loss at the end of each epoch')
plt.legend();
```

In [19]:
```
# train and test losses do not converge
```

In [20]:
```
# plot accuracy
plt.figure(figsize = (10,5))
plt.plot([t*100/(max_trn_batch*b_size) for t in train_correct], label='trainin
g accuracy')
plt.plot([t*100/(max_tst_batch*b_size) for t in test_correct], label='validati
on accuracy')
plt.title('Accuracy at the end of each epoch')
plt.legend();
```




In [21]:
```
# accuracy diverges as well
# model achieves 90% training accuracy and ~ 77% testing (validation) accuracy
# after ~ 30 epochs the validation accuracy reaches a plateau, while the train
ing accuracy continues to increase
    # if the number of epochs is increased, model could reach even higher trai
ning accuracy by "learning" the training set
    # this, however, will not lead to significant increase in validation accur
acy which is the true model accuracy
```

In [22]:
```
# print final test accuracy
print(f'Test accuracy: {test_correct[-1].item()*100/(max_tst_batch*b_size):.2
f}%') # take the value after the last epoch
```

```
Test accuracy: 77.00%
```

In [23]:
```
# test accuracy is 77%
```

In [24]:
```
# compare predictions against ground truth
# for better visualization we will use heatmap
```

In [25]:
```
# use all test images
test_load_all = DataLoader(test_data, batch_size = 36000, shuffle = False)
print(f'Testing images available:  {len(test_data)}')
```

```
Testing images available:  36000
```

In [26]:
```
# make predictions with all test images and show confusion matrix results as a
heat map

# import confusion matrix
from sklearn.metrics import confusion_matrix

with torch.no_grad():
    correct = 0
    for X_test, y_test in test_load_all:
        y_val = ResNet18model(X_test.to(device))
        predicted = torch.max(y_val,1)[1]
        correct += (predicted == y_test.to(device)).sum()

# convert results to CPU tensors to be able to use them as numpy arrays!
device = torch.device("cpu")
y_test = y_test.to(device)
predicted = predicted.to(device)

# create heat map from confusion matrix and plot
arr = confusion_matrix(y_test.view(-1), predicted.view(-1))
df_cm = pd.DataFrame(arr, class_names, class_names)
plt.figure(figsize = (10,8))
sns.heatmap(df_cm, annot = True, fmt = "d", cmap = 'BuGn')
plt.xlabel("Prediction")
plt.ylabel("Label (ground truth)")
plt.show()
```

In [27]:
```
# Confusion matrix shows:
    # class 'airplane' has highest prediction accuracy of 85.4% (7690/9000)
        # most often mistaken for 'ship'
    # class 'ship' is close second with prediction accuracy of 84.7% (7622/900
0)
        # most often mistaken for 'airplane'
    # class 'automobile' is third in prediction accuracy with 74.3% (6684/900
0)
        # most often mistaken for 'truck'
    # class 'truck' has lowest prediction accuracy of 66.4% (5980/9000)
        # most often mistaken for 'automobile'
    # the corresponding classes causing highest confusion are netural to expec
t due to the similarities between these classes

# In conclusion:
    # overall model accuracy is 77.7% (27976/36000) which is well above random
guess (25%)
    # given the similarity between the objects selected here and the small ima
ge resolution we consider this to be good accuracy
```