Packages

```
%matplotlib inline
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import random
import torchvision.transforms.functional as TF
import copy
from resnet import resnet18

plt.ion()   # interactive mode
```

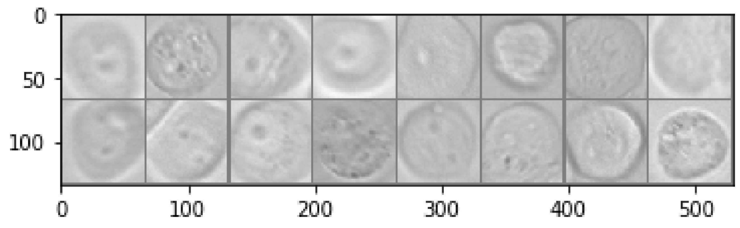
Hyperparameters

```
num_epochs = 200
batch_size = 16
learning_rate = 1e-3
```

Data load

```
# Data augmentation and normalization for training
# Just normalization for validation
def discrete_rotation(img):
    angles = [0,90,180,270]
    angle = random.choice(angles)
    img = TF.rotate(img, angle)
    return img

data_transforms = {
    'train': transforms.Compose([
        transforms.Grayscale(),
        transforms.Resize((64,64)),
        transforms.Lambda(discrete_rotation),
        transforms.RandomVerticalFlip(),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ]),
    'val': transforms.Compose([
        transforms.Grayscale(),
        transforms.Resize((64,64)),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ]),
}

data_dir = 'celldata'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=batch_size,
                                             shuffle=True, num_workers=4, drop_last=True)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}

class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


def imshow(img):
    img = img / 2 + 0.5
    img = img.numpy()
    plt.imshow(np.transpose(img, (1,2,0)))
    plt.show

# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

imshow(torchvision.utils.make_grid(inputs))
print(classes)
```

```
     tensor([1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0])
```



Network training

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            print("Starting phase {}".format(phase))
            if phase == 'train':
                #scheduler.step()
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                #print(inputs.shape)
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

Neural Network

```
model = resnet18()
model = model.to(device)

num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print('Number of parameters: %d' % num_params)
```

```
Number of parameters: 11171266
```

```
print(model)
```

```
ResNet(
  (conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=F
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=F
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
      (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_running_
      )
    )
    (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
        (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_running_
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
        (relu): ReLU(inplace)
```

Training of the CNN

```
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
#exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

```
model = train_model(model, criterion, optimizer_ft, None,
                       num_epochs)
```

```
Epoch 0/199
----------
Starting phase train
train Loss: 0.6583 Acc: 0.5300
Starting phase val
val Loss: 0.5599 Acc: 0.4000

Epoch 1/199
----------
Starting phase train
train Loss: 0.7045 Acc: 0.5050
Starting phase val
val Loss: 0.5892 Acc: 0.3750

Epoch 2/199
----------
Starting phase train
train Loss: 0.6339 Acc: 0.5650
Starting phase val
val Loss: 0.6951 Acc: 0.4000

Epoch 3/199
----------
Starting phase train
train Loss: 0.6356 Acc: 0.5400
Starting phase val
val Loss: 0.7384 Acc: 0.3750

Epoch 4/199
----------
Starting phase train
train Loss: 0.5703 Acc: 0.6650
Starting phase val
val Loss: 0.6365 Acc: 0.2500

Epoch 5/199
----------
Starting phase train
train Loss: 0.5602 Acc: 0.6400
Starting phase val
val Loss: 0.7368 Acc: 0.2250

Epoch 6/199
----------
Starting phase train
train Loss: 0.5475 Acc: 0.6850
Starting phase val
val Loss: 0.8024 Acc: 0.4250

Epoch 7/199
----------
Starting phase train
train Loss: 0.5884 Acc: 0.6200
Starting phase val
val Loss: 1.1772 Acc: 0.3750

Epoch 8/199
----------
Starting phase train
train Loss: 0.5975 Acc: 0.6600
```

```
Starting phase val
val Loss: 0.9599 Acc: 0.2500

Epoch 9/199
----------
Starting phase train
train Loss: 0.6209 Acc: 0.6100
Starting phase val
val Loss: 0.8278 Acc: 0.2750

Epoch 10/199
----------
```