yuxuanz6_cs544_project

May 9, 2019

1 Enviroment Settings

```
In [8]: from google.colab import drive
        drive.mount('/content/gdrive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/gdrive/
```

```
In [9]: ls
```

```
gdrive/  sample_data/
```

```
In [0]: import os
        os.chdir("/content/gdrive/My Drive/UIUC/2019_spring/CS544--optimizer computer vision/pr
```

```
In [11]: ls # Check if this is your MP4 folder
```

```
 data/                                'yuxuanz6_cs544_project(1).ipynb'
 Drift_Data/                          'yuxuanz6_cs544_project(2).ipynb'
 opt_res_part1/                       'yuxuanz6_cs544_project(3).ipynb'
 opt_res_part2/                       'yuxuanz6_cs544_project(4).ipynb'
 SGD-plot.png                         'yuxuanz6_cs544_project(5-part1).ipynb'
 submission/                          'yuxuanz6_cs544_project(6).ipynb'
'yuxuanz6_cs544_project(0).ipynb'      yuxuanz6_cs544_project.ipynb
```

```
In [0]: !mkdir /data
        !cp data/cifar100.tar.gz /data/
        !tar -xf /data/cifar100.tar.gz -C /data/
        !cp data/test.tar.gz /data
        !tar -xf /data/test.tar.gz -C /data
        !cp data/train.tar.gz /data
        !tar -xf /data/train.tar.gz -C /data/
```

In [13]:
```
ls /data
```

```
cifar100/  cifar100.tar.gz  test/  test.tar.gz  train/  train.tar.gz
```

In [14]:
```
!pip3 install torch torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages (1.1.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages (0.2.2.po
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from torch) (1
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-packages (from t
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from torchvision
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (from pillow>
```

In [15]:
```
import torch
a = torch.Tensor([1]).cuda()
print(a)
```

```
tensor([1.], device='cuda:0')
```

In [16]:
```
torch.cuda.is_available()
```

Out[16]:
```
True
```

2 Part1 - BaseNet(adapt from vgg net) Optimization

2.1 Part1.0 - Load the CIFAR-100 Data

In [17]:
```
"""Headers"""

from __future__ import print_function
from PIL import Image
import os
import os.path
import numpy as np
import sys
if sys.version_info[0] == 2:
    import cPickle as pickle
else:
    import pickle

import torch.utils.data as data
from torchvision.datasets.utils import download_url, check_integrity

import csv
%matplotlib inline
import matplotlib
```

```python
import matplotlib.pyplot as plt
import numpy as np
import os.path
import sys
import torch
import torch.utils.data
import torchvision
import torchvision.transforms as transforms

from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F

np.random.seed(111)
torch.cuda.manual_seed_all(111)
torch.manual_seed(111)
```

Out[17]: <torch._C.Generator at 0x7f8ef5e71f70>

In [0]:
```python
""""""

class CIFAR10_CS544(data.Dataset):
    """`CIFAR10 <https://www.cs.toronto.edu/~kriz/cifar.html>`_ Dataset.

    Args:
        root (string): Root directory of dataset where directory
            ``cifar-10-batches-py`` exists or will be saved to if download is set to T
        train (bool, optional): If True, creates dataset from training set, otherwise
            creates from test set.
        transform (callable, optional): A function/transform that  takes in an PIL ima
            and returns a transformed version. E.g, ``transforms.RandomCrop``
        target_transform (callable, optional): A function/transform that takes in the
            target and transforms it.
        download (bool, optional): If true, downloads the dataset from the internet an
            puts it in root directory. If dataset is already downloaded, it is not
            downloaded again.

    """
    base_folder = 'cifar100'
    url = "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
    filename = "cifar100.tar.gz"
    tgz_md5 = 'c58f30108f718f92721af3b95e74349a'
    train_list = [
        ['data_batch_1', 'c99cafc152244af753f735de768cd75f'],
        ['data_batch_2', 'd4bba439e000b95fd0a9bffe97cbabec'],
        ['data_batch_3', '54ebc095f3ab1f0389bbae665268c751'],
        ['data_batch_4', '634d18415352ddfa80567beed471001a'],
        ['data_batch_5', '482c414d41f54cd18b22e5b47cb7c3cb'],
```

```
]

test_list = [
    ['test_batch', '40351d587109b95175f43aff81a1287e'],
]

def __init__(self, root, fold="train",
             transform=None, target_transform=None,
             download=False):

    fold = fold.lower()

    self.train = False
    self.test = False
    self.val = False

    if fold == "train":
        self.train = True
    elif fold == "test":
        self.test = True
    elif fold == "val":
        self.val = True
    else:
        raise RuntimeError("Not train-val-test")


    self.root = os.path.expanduser(root)
    self.transform = transform
    self.target_transform = target_transform

    fpath = os.path.join(root, self.filename)
    if not self._check_integrity():
        raise RuntimeError('Dataset not found or corrupted.' +
                           ' Download it and extract the file again.')

    # now load the picked numpy arrays
    if self.train or self.val:
        self.train_data = []
        self.train_labels = []
        for fentry in self.train_list:
            f = fentry[0]
            file = os.path.join(self.root, self.base_folder, f)
            fo = open(file, 'rb')
            if sys.version_info[0] == 2:
                entry = pickle.load(fo)
            else:
                entry = pickle.load(fo, encoding='latin1')
            self.train_data.append(entry['data'])
```

```
                if 'labels' in entry:
                    self.train_labels += entry['labels']
                else:
                    self.train_labels += entry['fine_labels']
                fo.close()

            self.train_data = np.concatenate(self.train_data)
            self.train_data = self.train_data.reshape((50000, 3, 32, 32))
            self.train_data = self.train_data.transpose((0, 2, 3, 1))  # convert to HW

            p = np.arange(0,50000,10)
            mask_train = np.ones((50000,), dtype=bool)
            mask_train[p] = False
            mask_val = np.zeros((50000,), dtype=bool)
            mask_val[p] = True

            copy_all_data = np.array(self.train_data)
            self.val_data = np.array(copy_all_data[mask_val])
            self.train_data = np.array(copy_all_data[mask_train])

            copy_all_labels = np.array(self.train_labels)
            self.val_labels = np.array(copy_all_labels[mask_val])
            self.train_labels = np.array(copy_all_labels[mask_train])

        elif self.test:
            f = self.test_list[0][0]
            file = os.path.join(self.root, self.base_folder, f)
            fo = open(file, 'rb')
            if sys.version_info[0] == 2:
                entry = pickle.load(fo)
            else:
                entry = pickle.load(fo, encoding='latin1')
            self.test_data = entry['data']

            if 'labels' in entry:
                self.test_labels = entry['labels']
            else:
                self.test_labels = entry['fine_labels']
            fo.close()
            self.test_data = self.test_data.reshape((10000, 3, 32, 32))
            self.test_data = self.test_data.transpose((0, 2, 3, 1))  # convert to HWC

    def __getitem__(self, index):
        """
        Args:
            index (int): Index

        Returns:
```

```python
            tuple: (image, target) where target is index of the target class.
        """
        if self.train:
            img, target = self.train_data[index], self.train_labels[index]
        elif self.test:
            img, target = self.test_data[index], self.test_labels[index]
        elif self.val:
            img, target = self.val_data[index], self.val_labels[index]

        # doing this so that it is consistent with all other datasets
        # to return a PIL Image
        img = Image.fromarray(img)

        if self.transform is not None:
            img = self.transform(img)

        if self.target_transform is not None:
            target = self.target_transform(target)

        return img, target

    def __len__(self):
        if self.train:
            return len(self.train_data)
        elif self.test:
            return len(self.test_data)
        elif self.val:
            return len(self.val_data)

    def _check_integrity(self):
        root = self.root
        for fentry in (self.train_list + self.test_list):
            filename, md5 = fentry[0], fentry[1]
            fpath = os.path.join(root, self.base_folder, filename)
            if not check_integrity(fpath, md5):
                return False
        return True

    def __repr__(self):
        fmt_str = 'Dataset ' + self.__class__.__name__ + '\n'
        fmt_str += '    Number of datapoints: {}\n'.format(self.__len__())
        tmp = 'train' if self.train is True else 'test'
        fmt_str += '    Split: {}\n'.format(tmp)
        fmt_str += '    Root Location: {}\n'.format(self.root)
        tmp = '    Transforms (if any): '
        fmt_str += '{0}{1}\n'.format(tmp, self.transform.__repr__().replace('\n', '\n'
        tmp = '    Target Transforms (if any): '
        fmt_str += '{0}{1}'.format(tmp, self.target_transform.__repr__().replace('\n',
```

```
            return fmt_str


    class CIFAR100_CS544(CIFAR10_CS544):
        """`CIFAR100 <https://www.cs.toronto.edu/~kriz/cifar.html>`_ Dataset.

        This is a subclass of the `CIFAR10` Dataset.
        """
        base_folder = 'cifar100'
        filename = "cifar100.tar.gz"
        tgz_md5 = 'e68a4c763591787a0b39fe2209371f32'
        train_list = [
            ['train_cs544', '49eee854445c1e2ebe796cd93c20bb0f'],
        ]

        test_list = [
            ['test_cs544', 'd3fe9f6a9251bd443f428f896d27384f'],
        ]
```

2.2 part1.1 - Define picked optimiers result array

In [0]:
```
optimer_train_loss_over_epochs_opts = []
optimer_val_accuracy_over_epochs_opts = []
optimer_time_cost = []
```

2.3 Part1.2 - Set the Class and Epoches, optimier names

In [0]:
```
# ---------
EPOCHS = 20
# ---------
optimer_name = ['SGD','SGD+momentum0.9','ASGD','Adam','Adadelta_default',
                'Adagrad_default','Adamax_default','RMSprop_default','ASGD_default','R
IS_GPU = True
TEST_BS = 256
TOTAL_CLASSES = 100
TRAIN_BS = 32
PATH_TO_CIFAR100_CS544 = "/data/"
```

In [208]:
```
ls /data/cifar100/
```

```
test_cs544  train_cs544
```

2.4 Part1.3 - Define the val accuracy function

In [0]:
```
def calculate_val_accuracy(valloader, is_gpu):
    """ Util function to calculate val set accuracy,
    both overall and per class accuracy
    Args:
```

```
        valloader (torch.utils.data.DataLoader): val set
        is_gpu (bool): whether to run on GPU
    Returns:
        tuple: (overall accuracy, class level accuracy)
    """
    correct = 0.
    total = 0.
    predictions = []

    class_correct = list(0. for i in range(TOTAL_CLASSES))
    class_total = list(0. for i in range(TOTAL_CLASSES))

    for data in valloader:
        images, labels = data
        if is_gpu:
            images = images.cuda()
            labels = labels.cuda()
        outputs = net(Variable(images))
        _, predicted = torch.max(outputs.data, 1)
        predictions.extend(list(predicted.cpu().numpy()))
        total += labels.size(0)
        correct += (predicted == labels).sum()

        c = (predicted == labels).squeeze()
        for i in range(len(labels)):
            label = labels[i]
            class_correct[label] += c[i]
            class_total[label] += 1

    class_accuracy = 100 * np.divide(class_correct, class_total)
    return 100*correct/total, class_accuracy
```

2.5 Part1.4 - Augument and Normalize input data

In [210]:
```
# The output of torchvision datasets are PILImage images of range [0, 1].
# Using transforms.ToTensor(), transform them to Tensors of normalized range
# [-1, 1].


train_transform = transforms.Compose(
    [
    transforms.RandomCrop(32),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
#     transforms.Normalize([0.505, 0.496, 0.446], [0.259, 0.254, 0.275])
      transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ])
test_transform = transforms.Compose(
```

```
        [transforms.ToTensor(),
    #     transforms.Normalize([0.505, 0.496, 0.446], [0.259, 0.254, 0.275])
         transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
        ])

    # ---------------------

    trainset = CIFAR100_CS544(root=PATH_TO_CIFAR100_CS544, fold="train",
                                            download=True, transform=train_transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=TRAIN_BS,
                                              shuffle=True, num_workers=2)
    print("Train set size: "+str(len(trainset)))

    valset = CIFAR100_CS544(root=PATH_TO_CIFAR100_CS544, fold="val",
                                            download=True, transform=test_transform)
    valloader = torch.utils.data.DataLoader(valset, batch_size=TEST_BS,
                                              shuffle=False, num_workers=2)
    print("Val set size: "+str(len(valset)))

    testset = CIFAR100_CS544(root=PATH_TO_CIFAR100_CS544, fold="test",
                                            download=True, transform=test_transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=TEST_BS,
                                              shuffle=False, num_workers=2)
    print("Test set size: "+str(len(testset)))

    # The 100 classes for CIFAR100
    classes = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle
```

```
Train set size: 45000
Val set size: 5000
Test set size: 10000
```

2.6 Part1.5 - Define neural network layers

In [0]:
```
########################################################################
# 2. Define a Convolution Neural Network
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

import torch.nn as nn
import torch.nn.functional as F

def conv2d_norm_relu(in_channel, out_channel, kernel, stride=1, padding=1):
    layer = nn.Sequential(
        nn.Conv2d(in_channel, out_channel, kernel, stride=1, padding=1),
        nn.BatchNorm2d(out_channel),
        nn.ReLU(True)
    )
```

```
    return layer

class BaseNet(nn.Module):
    def __init__(self):
        super(BaseNet, self).__init__()


        self.conv1 = conv2d_norm_relu(3, 64, 3,  padding=1)
        self.conv2 = conv2d_norm_relu(64, 64, 3,  padding=1)
        self.conv3 = conv2d_norm_relu(64, 128, 3,  padding=1)
        self.conv4 = conv2d_norm_relu(128, 128, 3,  padding=1)
        self.conv5 = conv2d_norm_relu(128, 256, 3,  padding=1)
        self.conv6 = conv2d_norm_relu(256, 256, 3,  padding=1)
        self.conv7 = conv2d_norm_relu(256, 512, 3,  padding=1)
        self.conv8 = conv2d_norm_relu(512, 512, 3,  padding=1)
        self.pool = nn.MaxPool2d(2, 2, padding=0)


        class Flatten(torch.nn.Module):
           def forward(self, x):
            return x.view(x.size()[0], -1)

        self.fc_net = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 100)
        )

    def forward(self, x):

        x = self.conv1(x)
        x = self.conv2(x)
        x = self.pool(x) # output: 16 x 16 x 64
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.pool(x) # output: 8 x 8 x 128
        x = self.conv5(x)
        x = self.conv6(x)
        x = self.pool(x)# output: 4 x 4 x 256
        x = self.conv7(x)
        x = self.conv8(x)
        x = self.pool(x)# output: 2 x 2 x 512
        x = x.view(-1, 2048)
#         print('view:',x.shape)
```

```
        x = self.fc_net(x)

        return x

# Create an instance of the nn.module class defined above:
net = BaseNet()

# For training on GPU, we need to transfer net and data onto the GPU
# http://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#training-on-gpu
if IS_GPU:
    net = net.cuda()
```

2.7 Part1.6 - Define loss function and optimizers

In [0]:
```
########################################################################
# 3. Define a Loss function and optimizer
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Here we use Cross-Entropy loss and SGD with momentum.
# The CrossEntropyLoss criterion already includes softmax within its
# implementation. That's why we don't use a softmax in our model
# definition.

import torch.optim as optim
criterion = nn.CrossEntropyLoss()

# Tune the learning rate.
# See whether the momentum is useful or not
# optimizer = optim.SGD(net.parameters(),lr=1e-3)
# optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
# optimizer = optim.ASGD(net.parameters(),  lr=1e-3) # remove the alpha
# optimizer = optim.Adam(net.parameters(), lr=1e-3)
# optimizer = optim.Adadelta(net.parameters(), lr=1.0, rho=0.9, eps=1e-06, weight_deca
# optimizer = optim.Adagrad(net.parameters(), lr=0.01, lr_decay=0, weight_decay=0, ini
# optimizer = optim.Adamax(net.parameters(), lr=0.002, betas=(0.9, 0.999), eps=1e-08, 
# optimizer = optim.RMSprop(net.parameters(), lr=0.01, alpha=0.99, eps=1e-08, weight_d
# optimizer = optim.ASGD(net.parameters(), lr=0.01, lambd=0.0001, alpha=0.75, t0=10000
optimizer = optim.Rprop(net.parameters(), lr=0.01, etas=(0.5, 1.2), step_sizes=(1e-06,

timer  = []
train_loss_over_epochs = []
val_accuracy_over_epochs = []
plt.ioff()
fig = plt.figure()
```

2.8 Part1.7- Train, Validation

In [0]:
```
import time
```

```
In [214]: ########################################################################
          # 4. Train the network
          # ^^^^^^^^^^^^^^^^^^^^
          for epoch in range(EPOCHS):  # loop over the dataset multiple times
              start = time.time()
              running_loss = 0.0
              for i, data in enumerate(trainloader, 0):
                  # get the inputs
                  inputs, labels = data

                  if IS_GPU:
                      inputs = inputs.cuda()
                      labels = labels.cuda()

                  # wrap them in Variable
                  inputs, labels = Variable(inputs), Variable(labels)

                  # 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()

              # Normalizing the loss by the total number of train batches
              running_loss/=len(trainloader)
              print('[%d] loss: %.3f' %
                    (epoch + 1, running_loss))

              # Scale of 0.0 to 100.0
              # Calculate validation set accuracy of the existing model
              val_accuracy, val_classwise_accuracy = \
                  calculate_val_accuracy(valloader, IS_GPU)
              print('Accuracy of the network on the val images: %d %%' % (val_accuracy))

              # # Optionally print classwise accuracies
              # for c_i in range(TOTAL_CLASSES):
              #     print('Accuracy of %5s : %2d %%' % (
              #         classes[c_i], 100 * val_classwise_accuracy[c_i]))
              end = time.time()
              ti = end - start
              timer.append(ti)
              train_loss_over_epochs.append(running_loss)
```

```
        val_accuracy_over_epochs.append(val_accuracy)

    # -----------------------------
```

```
[1] loss: 271.317
Accuracy of the network on the val images: 1 %
[2] loss: 371.431
Accuracy of the network on the val images: 1 %
[3] loss: 353.140
Accuracy of the network on the val images: 1 %
[4] loss: 329.056
Accuracy of the network on the val images: 1 %
[5] loss: 336.361
Accuracy of the network on the val images: 1 %
[6] loss: 330.920
Accuracy of the network on the val images: 1 %
[7] loss: 357.755
Accuracy of the network on the val images: 1 %
[8] loss: 352.342
Accuracy of the network on the val images: 1 %
[9] loss: 327.984
Accuracy of the network on the val images: 1 %
[10] loss: 323.862
Accuracy of the network on the val images: 1 %
[11] loss: 318.324
Accuracy of the network on the val images: 1 %
[12] loss: 349.535
Accuracy of the network on the val images: 1 %
[13] loss: 368.678
Accuracy of the network on the val images: 1 %
[14] loss: 354.838
Accuracy of the network on the val images: 1 %
[15] loss: 352.203
Accuracy of the network on the val images: 1 %
[16] loss: 345.508
Accuracy of the network on the val images: 1 %
[17] loss: 352.870
Accuracy of the network on the val images: 1 %
[18] loss: 413.680
Accuracy of the network on the val images: 1 %
[19] loss: 418.777
Accuracy of the network on the val images: 1 %
[20] loss: 445.755
Accuracy of the network on the val images: 0 %
```

2.9 part1.8 - Visualizing single optimier prediction result

```
In [215]: # Plot train loss over epochs and val set accuracy over epochs
          # Nothing to change here
          # -------------
          plt.subplot(3, 1, 1)
          plt.ylabel('time sec')
          plt.plot(np.arange(EPOCHS), timer, 'g-')
          plt.title(str(optimer_name[9]) + ' train time,loss and val accuracy')
          plt.xticks(np.arange(EPOCHS, dtype=int))
          plt.grid(True)

          plt.subplot(3, 1, 2)
          plt.ylabel('train loss')
          plt.plot(np.arange(EPOCHS), train_loss_over_epochs, 'k-')
          plt.xticks(np.arange(EPOCHS, dtype=int))
          plt.grid(True)

          plt.subplot(3, 1, 3)
          plt.plot(np.arange(EPOCHS), val_accuracy_over_epochs, 'b-')
          plt.ylabel('val accuracy')
          plt.xlabel('Epochs')
          plt.xticks(np.arange(EPOCHS, dtype=int))
          plt.grid(True)
          plt.savefig(str(optimer_name[9])+"_plot.png")
          plt.close(fig)
          print(str(optimer_name[9]) + ' finished training')
          # -------------
```

```
Rprop_default finished training
```

```
In [216]: print("train_loss_over_epochs_opts =  ", train_loss_over_epochs)
          print("val_accuracy_over_epochs_opts = ", val_accuracy_over_epochs)
          print("time_cost = ", timer)
          if(len(train_loss_over_epochs) == 20 and
             len(val_accuracy_over_epochs) == 20 and
             len(timer) == 20):
          #      optimer_train_loss_over_epochs_opts[0] = train_loss_over_epochs
          #      optimer_val_accuracy_over_epochs_opts[0] = val_accuracy_over_epochs
          #      optimer_time_cost[0] = timer
             optimer_train_loss_over_epochs_opts.append(train_loss_over_epochs)
             optimer_val_accuracy_over_epochs_opts.append(val_accuracy_over_epochs)
             optimer_time_cost.append(timer)
             t = np.arange(0, EPOCHS, 1) # equals epoch number
             print(len(train_loss_over_epochs),len(val_accuracy_over_epochs),len(timer),len(t)
             print(len(optimer_train_loss_over_epochs_opts),len(optimer_val_accuracy_over_epoc
```

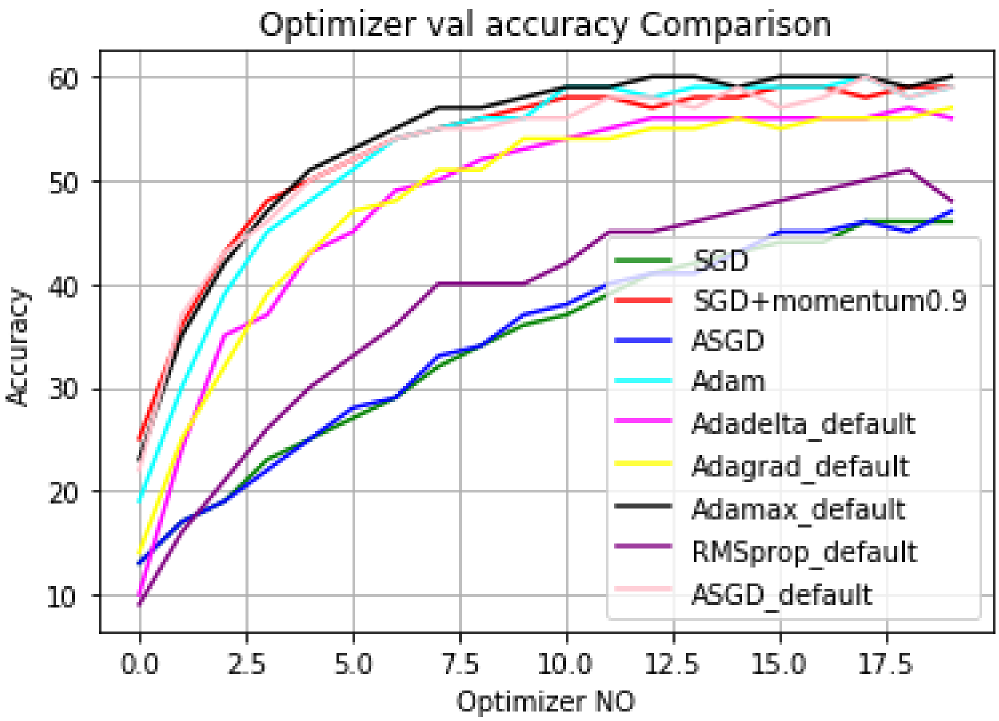
```
train_loss_over_epochs_opts =   [271.31657391909425, 371.43133382926027, 353.1397174162143, 32
val_accuracy_over_epochs_opts =  [tensor(1, device='cuda:0'), tensor(1, device='cuda:0'), tens
```

```
time_cost =  [90.36958765983582, 91.55633068084717, 90.33762764930725, 90.08072853088379, 89.99
20 20 20 20
10 10 10
```

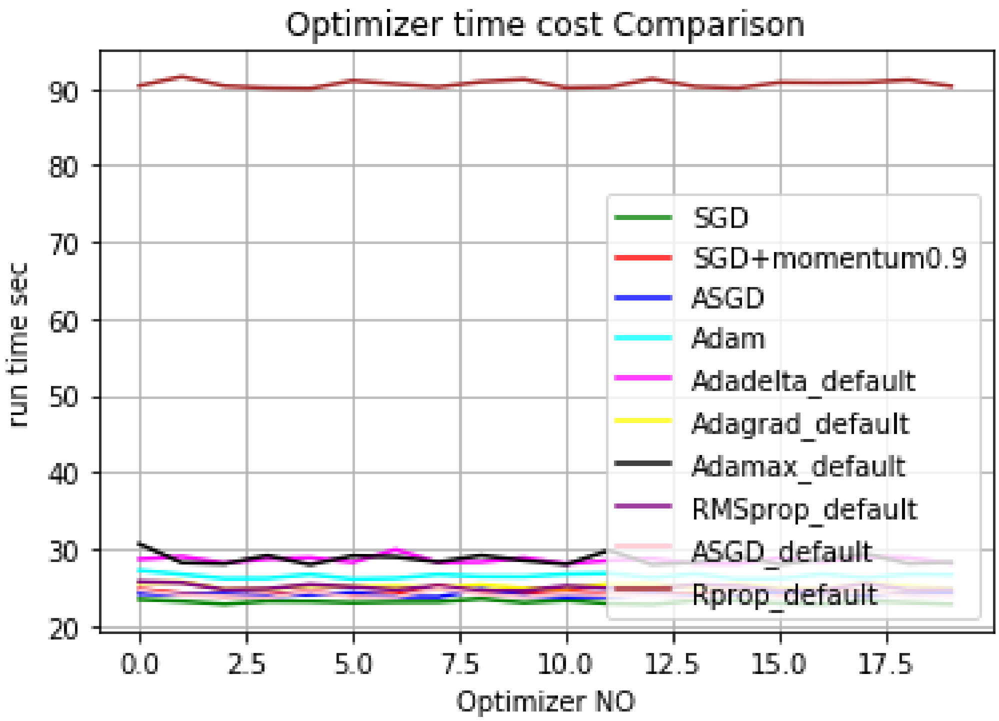
2.10 part1.9 - Visualizing picked optimiers prediction result

```
In [251]: if(len(optimer_train_loss_over_epochs_opts) == 10 and
             len(optimer_val_accuracy_over_epochs_opts) == 10 and
             len(optimer_time_cost) == 10):
             plt.title('Optimizer train loss Comparison')
             plt.ylabel('Loss')
             plt.xlabel('Optimizer NO')
             plt.plot(t,optimer_train_loss_over_epochs_opts[0],label=str(optimer_name[0]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[1],label=str(optimer_name[1]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[2],label=str(optimer_name[2]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[3],label=str(optimer_name[3]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[4],label=str(optimer_name[4]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[5],label=str(optimer_name[5]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[6],label=str(optimer_name[6]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[7],label=str(optimer_name[7]),colo
             plt.plot(t,optimer_train_loss_over_epochs_opts[8],label=str(optimer_name[8]),colo
          #    plt.plot(t,optimer_train_loss_over_epochs_opts[9],label=str(optimer_name[9]),co
             plt.grid(True)
             plt.legend([str(optimer_name[0]),str(optimer_name[1]),str(optimer_name[2]),str(op
                         str(optimer_name[4]),str(optimer_name[5]),str(optimer_name[6]),str(opti
                         str(optimer_name[8])], loc='lower right')
             plt.show()
```


```
In [238]: if(len(optimer_train_loss_over_epochs_opts) == 10 and
             len(optimer_val_accuracy_over_epochs_opts) == 10 and
             len(optimer_time_cost) == 10):
             plt.title('Optimizer val accuracy Comparison')
             plt.ylabel('Accuracy')
             plt.xlabel('Optimizer NO')
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[0],label=str(optimer_name[0]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[1],label=str(optimer_name[1]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[2],label=str(optimer_name[2]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[3],label=str(optimer_name[3]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[4],label=str(optimer_name[4]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[5],label=str(optimer_name[5]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[6],label=str(optimer_name[6]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[7],label=str(optimer_name[7]),co
             plt.plot(t,optimer_val_accuracy_over_epochs_opts[8],label=str(optimer_name[8]),co
          #    plt.plot(t,optimer_val_accuracy_over_epochs_opts[9],label=str(optimer_name[9]),
             plt.grid(True)
             plt.legend([str(optimer_name[0]),str(optimer_name[1]),str(optimer_name[2]),str(op
                         str(optimer_name[4]),str(optimer_name[5]),str(optimer_name[6]),str(opt
                         str(optimer_name[8])], loc='lower right')
             plt.show()
```




```
In [250]: if(len(optimer_train_loss_over_epochs_opts) == 10 and
             len(optimer_val_accuracy_over_epochs_opts) == 10 and
             len(optimer_time_cost) == 10):
             plt.title('Optimizer time cost Comparison')
             plt.ylabel('run time sec')
             plt.xlabel('Optimizer NO')
             plt.plot(t,optimer_time_cost[0],label=str(optimer_name[0]),color='green')
             plt.plot(t,optimer_time_cost[1],label=str(optimer_name[1]),color='red')
             plt.plot(t,optimer_time_cost[2],label=str(optimer_name[2]),color='blue')
             plt.plot(t,optimer_time_cost[3],label=str(optimer_name[3]),color='cyan')
             plt.plot(t,optimer_time_cost[4],label=str(optimer_name[4]),color='magenta')
             plt.plot(t,optimer_time_cost[5],label=str(optimer_name[5]),color='yellow')
             plt.plot(t,optimer_time_cost[6],label=str(optimer_name[6]),color='black')
             plt.plot(t,optimer_time_cost[7],label=str(optimer_name[7]),color='purple')
             plt.plot(t,optimer_time_cost[8],label=str(optimer_name[8]),color='pink')
          #    plt.plot(t,optimer_time_cost[9],label=str(optimer_name[9]),color='brown')
             plt.grid(True)
             plt.legend([str(optimer_name[0]),str(optimer_name[1]),str(optimer_name[2]),str(op
                        str(optimer_name[4]),str(optimer_name[5]),str(optimer_name[6]),str(opti
                        str(optimer_name[8])], loc='lower right')
             plt.show()
```



3 Part 2 - ResNet Model Optimization

3.1 part2.0 - Define picked optimiers result array

In [0]:
```
optimer_train_loss_over_epochs_opts2 = []
optimer_train_accuracy_over_epochs_opts2 = []
optimer_time_cost2 = []
optimizer_test_loss = []
optimizer_test_accuracy = []
optimizer_time_cost3  = []
```

In [0]:
```
optimer_name2 = ['SGD','SGD+momentum0.9','ASGD','Adam','Adadelta_default',
                 'Adagrad_default','Adamax_default','RMSprop_default','ASGD_default','Rp
```

3.2 Part 2.1 - Load the pre-trained resnet model

In [0]:
```
"""Headers"""
import os
import os.path as osp
import time

%matplotlib inline
import matplotlib.pyplot as plt
```

```
import torch
import torch.nn as nn
import torchvision.models as models
import torch.optim as optim

from torchvision import datasets
import torchvision.transforms as transforms
```

```
In [0]: class PreTrainedResNet(nn.Module):
          def __init__(self, num_classes, feature_extracting):
            super(PreTrainedResNet, self).__init__()

            self.resnet18 = models.resnet18(pretrained=True)
        #     self.resnet152 = models.resnet152(pretrained=True) # will run out of memory
        #     self.resnet101 = models.resnet101(pretrained=True) # will run out of memory
        #     self.resnet50 = models.resnet50(pretrained=True)
        #     self.inception = models.inception_v3(pretrained=True) # softmax problem
            #Set gradients to false
            if feature_extracting:
              for param in self.resnet18.parameters():
                  param.requires_grad = False

            #Replace last fc layer
            num_feats = self.resnet18.fc.in_features
            self.resnet18.fc = nn.Linear(num_feats, 200)

          def forward(self, x):
            x = self.resnet18(x)
            return x
```

3.3 Part 2.2 - Define train function

```
In [0]: def train(model, optimizer, criterion, epoch, num_epochs):
          model.train()
          epoch_loss = 0.0
          epoch_acc = 0.0

          for batch_idx, (images, labels) in enumerate(dataloaders['train']):
            #zero the parameter gradients
            optimizer.zero_grad()

            #move to GPU
            images, labels = images.cuda(), labels.cuda()

            #forward
            outputs = model.forward(images)
```

```
            loss = criterion(outputs, labels)

            _, preds = torch.max(outputs.data, 1)

            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
            epoch_acc += torch.sum(preds == labels).item()

        epoch_loss /= dataset_sizes['train']
        epoch_acc /= dataset_sizes['train']

        print('TRAINING Epoch %d/%d Loss %.4f Accuracy %.4f' % (epoch, num_epochs, epoch_loss
        return epoch_loss,epoch_acc
```

3.4 Part 2.3 - Define main function

1. Vary hyperparams
2. Data augmentation

```
In [0]: NUM_EPOCHS = 40
        LEARNING_RATE = 0.01 #
        BATCH_SIZE = 80
        RESNET_LAST_ONLY = False #Fine tunes only the last layer. Set to False to fine tune en

        root_path = '/data/'

        data_transforms = {
            'train': transforms.Compose([
                transforms.Resize(256),
        #       transforms.CenterCrop(224),
                transforms.RandomResizedCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ]),
            'test': transforms.Compose([
                transforms.Resize(256),
        #         transforms.CenterCrop(224),
                transforms.RandomResizedCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ]),
        }

        # loading datasets with PyTorch ImageFolder
        image_datasets = {x: datasets.ImageFolder(os.path.join(root_path, x),
                                                  data_transforms[x])
```

```
                   for x in ['train', 'test']}

# defining data loaders to load data using image_datasets and transforms, here we also
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=BATCH_SIZE
                                              shuffle=True, num_workers=4)
              for x in ['train', 'test']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'test']}
class_names = image_datasets['train'].classes

#Initialize the model
model = PreTrainedResNet(len(class_names), RESNET_LAST_ONLY)
model = model.cuda()
```

3.5 Part 2.4 - Define loss function and optimiers

In [0]:
```
# optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, momentum=0.9) -- part 2 
criterion = nn.CrossEntropyLoss()

# optimizer = optim.SGD(model.parameters(),lr=LEARNING_RATE)
# optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, momentum=0.9)
# optimizer = optim.ASGD(model.parameters(),  lr=LEARNING_RATE) # remove the alpha
# optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
# optimizer = optim.Adadelta(model.parameters(), lr=1.0, rho=0.9, eps=1e-06, weight_de
# optimizer = optim.Adagrad(model.parameters(), lr=0.01, lr_decay=0, weight_decay=0, i
# optimizer = optim.Adamax(model.parameters(), lr=0.002, betas=(0.9, 0.999), eps=1e-08
# optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99, eps=1e-08, weight
# optimizer = optim.ASGD(model.parameters(), lr=0.01, lambd=0.0001, alpha=0.75, t0=100
optimizer = optim.Rprop(model.parameters(), lr=0.01, etas=(0.5, 1.2), step_sizes=(1e-0


timer2  = []
train_loss_over_epochs2 = []
train_accuracy_over_epochs2 = []
plt.ioff()
fig = plt.figure()
```

3.6 Part 2.5 - train the model

In [205]:
```
#Begin Train
for epoch in range(NUM_EPOCHS):
  start = time.time()
  train_epoch_loss,train_epoch_acc = train(model, optimizer, criterion, epoch+1, NUM_
  end = time.time()
  ti = end - start
  timer2.append(ti)
  train_loss_over_epochs2.append(train_epoch_loss)
  train_accuracy_over_epochs2.append(train_epoch_acc)
```

```python
    print("Finished Training")
    print("-"*10)

    optimer_train_loss_over_epochs_opts2.append(train_loss_over_epochs2)
    optimer_train_accuracy_over_epochs_opts2.append(train_accuracy_over_epochs2)
    optimer_time_cost2.append(timer2)
```

```
TRAINING Epoch 1/40 Loss 0.0919 Accuracy 0.0030
TRAINING Epoch 2/40 Loss 0.0784 Accuracy 0.0093
TRAINING Epoch 3/40 Loss 0.1210 Accuracy 0.0090
TRAINING Epoch 4/40 Loss 0.1626 Accuracy 0.0093
TRAINING Epoch 5/40 Loss 0.2677 Accuracy 0.0083
TRAINING Epoch 6/40 Loss 0.2011 Accuracy 0.0090
TRAINING Epoch 7/40 Loss 0.3129 Accuracy 0.0050
TRAINING Epoch 8/40 Loss 0.3288 Accuracy 0.0077
TRAINING Epoch 9/40 Loss 0.3664 Accuracy 0.0073
TRAINING Epoch 10/40 Loss 1.1886 Accuracy 0.0050
TRAINING Epoch 11/40 Loss 1.0636 Accuracy 0.0073
TRAINING Epoch 12/40 Loss 2.1457 Accuracy 0.0083
TRAINING Epoch 13/40 Loss 2.4675 Accuracy 0.0063
TRAINING Epoch 14/40 Loss 4.0406 Accuracy 0.0097
TRAINING Epoch 15/40 Loss 3.7310 Accuracy 0.0090
TRAINING Epoch 16/40 Loss 4.8876 Accuracy 0.0090
TRAINING Epoch 17/40 Loss 4.1615 Accuracy 0.0117
TRAINING Epoch 18/40 Loss 4.8271 Accuracy 0.0100
TRAINING Epoch 19/40 Loss 6.1380 Accuracy 0.0047
TRAINING Epoch 20/40 Loss 5.9884 Accuracy 0.0067
TRAINING Epoch 21/40 Loss 6.4026 Accuracy 0.0103
TRAINING Epoch 22/40 Loss 7.3444 Accuracy 0.0063
TRAINING Epoch 23/40 Loss 5.9234 Accuracy 0.0057
TRAINING Epoch 24/40 Loss 5.0320 Accuracy 0.0083
TRAINING Epoch 25/40 Loss 5.1664 Accuracy 0.0080
TRAINING Epoch 26/40 Loss 7.2598 Accuracy 0.0057
TRAINING Epoch 27/40 Loss 4.2791 Accuracy 0.0100
TRAINING Epoch 28/40 Loss 4.9331 Accuracy 0.0093
TRAINING Epoch 29/40 Loss 4.8941 Accuracy 0.0077
TRAINING Epoch 30/40 Loss 5.9331 Accuracy 0.0093
TRAINING Epoch 31/40 Loss 2.8656 Accuracy 0.0067
TRAINING Epoch 32/40 Loss 4.1043 Accuracy 0.0090
TRAINING Epoch 33/40 Loss 5.0496 Accuracy 0.0057
TRAINING Epoch 34/40 Loss 3.1052 Accuracy 0.0057
TRAINING Epoch 35/40 Loss 5.9857 Accuracy 0.0093
TRAINING Epoch 36/40 Loss 4.8002 Accuracy 0.0083
TRAINING Epoch 37/40 Loss 3.7178 Accuracy 0.0070
TRAINING Epoch 38/40 Loss 4.2180 Accuracy 0.0090
TRAINING Epoch 39/40 Loss 3.5416 Accuracy 0.0063
TRAINING Epoch 40/40 Loss 3.3093 Accuracy 0.0090
Finished Training
```

3.7 Part 2.6 - Visualizing single optimizer train prediction result

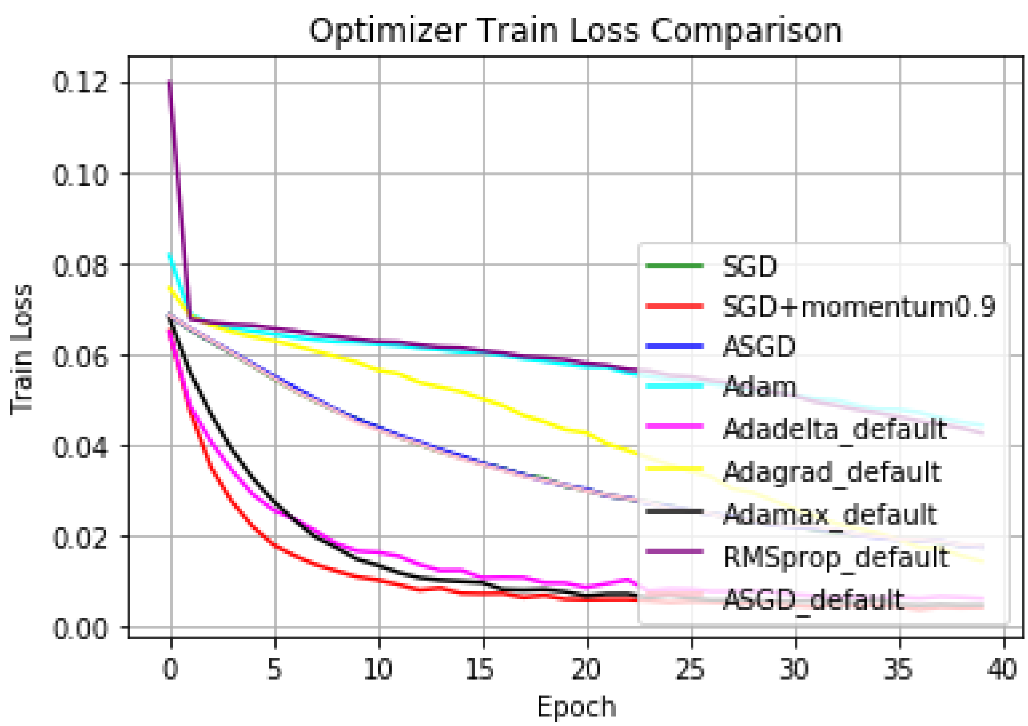
In [206]:
```
# Plot train loss over epochs and val set accuracy over epochs
# Nothing to change here
# --------------
print(NUM_EPOCHS,len(timer2))
plt.subplot(3, 1, 1)
plt.ylabel('time sec')
plt.plot(np.arange(NUM_EPOCHS), timer2, 'g-')
plt.title(str(optimer_name2[9]) + ' train time,loss and accuracy')
plt.xticks(np.arange(NUM_EPOCHS, dtype=int))
plt.grid(True)

plt.subplot(3, 1, 2)
plt.ylabel('train loss')
plt.plot(np.arange(NUM_EPOCHS), train_loss_over_epochs2, 'k-')
plt.xticks(np.arange(NUM_EPOCHS, dtype=int))
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(np.arange(NUM_EPOCHS), train_accuracy_over_epochs2, 'b-')
plt.ylabel('train accuracy')
plt.xlabel('Epochs')
plt.xticks(np.arange(NUM_EPOCHS, dtype=int))
plt.grid(True)
plt.savefig('opt_res_part2/' + str(optimer_name2[9])+"-plot.png")
plt.close(fig)
print(str(optimer_name2[9]) + ' finished training')
# --------------
print(len(optimer_train_loss_over_epochs_opts2),len(optimer_train_accuracy_over_epoch
```

```
40 40
Rprop_default finished training
10 10 10
```

3.8 Part 2.7 - Visualizing total optimizers train prediction result

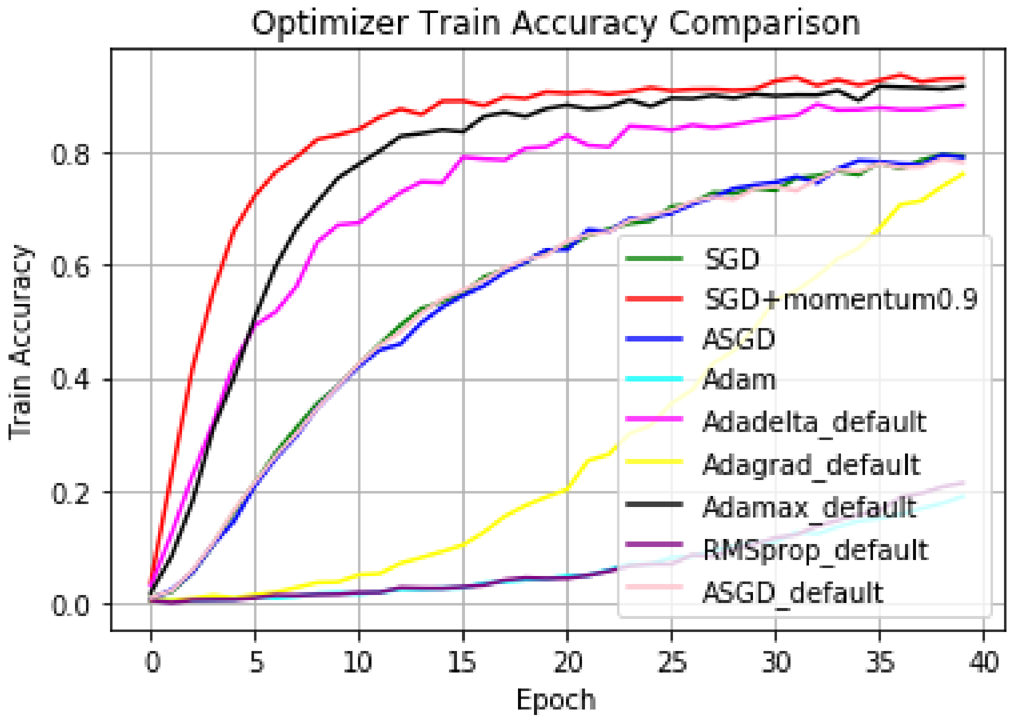
In [218]:
```
if(len(optimer_train_loss_over_epochs_opts2) == 10 and
   len(optimer_train_accuracy_over_epochs_opts2) == 10 and
   len(optimer_time_cost2) == 10):
   t = np.arange(0, NUM_EPOCHS, 1) # equals epoch number
   plt.title('Optimizer Train Loss Comparison')
   plt.ylabel('Train Loss')
   plt.xlabel('Epoch')
```

```
    plt.plot(t,optimer_train_loss_over_epochs_opts2[0],label=str(optimer_name2[0]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[1],label=str(optimer_name2[1]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[2],label=str(optimer_name2[2]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[3],label=str(optimer_name2[3]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[4],label=str(optimer_name2[4]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[5],label=str(optimer_name2[5]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[6],label=str(optimer_name2[6]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[7],label=str(optimer_name2[7]),co
    plt.plot(t,optimer_train_loss_over_epochs_opts2[8],label=str(optimer_name2[8]),co
#     plt.plot(t,optimer_train_loss_over_epochs_opts2[9],label=str(optimer_name2[9]),
    plt.grid(True)
    plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(
                str(optimer_name2[8])], loc='lower right')
    plt.show()
```



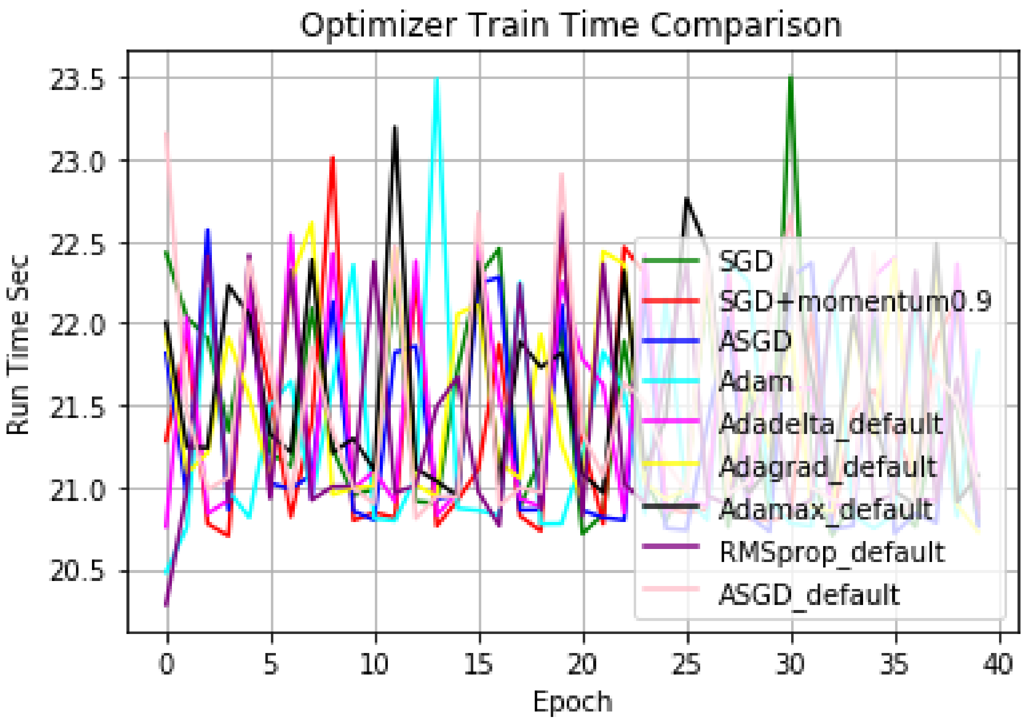

```
In [240]: if(len(optimer_train_loss_over_epochs_opts2) == 10 and
              len(optimer_train_accuracy_over_epochs_opts2) == 10 and
              len(optimer_time_cost2) == 10):
              t = np.arange(0, NUM_EPOCHS, 1) # equals epoch number
              plt.title('Optimizer Train Accuracy Comparison')
              plt.ylabel('Train Accuracy')
              plt.xlabel('Epoch')
              plt.plot(t,optimer_train_accuracy_over_epochs_opts2[0],label=str(optimer_name2[0]
```

```
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[1],label=str(optimer_name2[1]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[2],label=str(optimer_name2[2]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[3],label=str(optimer_name2[3]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[4],label=str(optimer_name2[4]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[5],label=str(optimer_name2[5]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[6],label=str(optimer_name2[6]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[7],label=str(optimer_name2[7]
    plt.plot(t,optimer_train_accuracy_over_epochs_opts2[8],label=str(optimer_name2[8]
#   plt.plot(t,optimer_train_accuracy_over_epochs_opts2[9],label=str(optimer_name2[9
    plt.grid(True)
    plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(
                str(optimer_name2[8])], loc='lower right')
    plt.show()
```




```
In [244]: if(len(optimer_train_loss_over_epochs_opts2) == 10 and
    len(optimer_train_accuracy_over_epochs_opts2) == 10 and
    len(optimer_time_cost2) == 10):
    t = np.arange(0, NUM_EPOCHS, 1) # equals epoch number
    plt.title('Optimizer Train Time Comparison')
    plt.ylabel('Run Time Sec')
    plt.xlabel('Epoch')
    plt.plot(t,optimer_time_cost2[0],label=str(optimer_name2[0]),color='green')
    plt.plot(t,optimer_time_cost2[1],label=str(optimer_name2[1]),color='red')
```

```
    plt.plot(t,optimer_time_cost2[2],label=str(optimer_name2[2]),color='blue')
    plt.plot(t,optimer_time_cost2[3],label=str(optimer_name2[3]),color='cyan')
    plt.plot(t,optimer_time_cost2[4],label=str(optimer_name2[4]),color='magenta')
    plt.plot(t,optimer_time_cost2[5],label=str(optimer_name2[5]),color='yellow')
    plt.plot(t,optimer_time_cost2[6],label=str(optimer_name2[6]),color='black')
    plt.plot(t,optimer_time_cost2[7],label=str(optimer_name2[7]),color='purple')
    plt.plot(t,optimer_time_cost2[8],label=str(optimer_name2[8]),color='pink')
#  plt.plot(t,optimer_time_cost2[9],label=str(optimer_name2[9]),color='brown')
    plt.grid(True)
    plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(o
                str(optimer_name2[8])], loc='lower right')
    plt.show()
```




```
In [250]: if(len(optimer_train_loss_over_epochs_opts2) == 10 and
    len(optimer_train_accuracy_over_epochs_opts2) == 10 and
    len(optimer_time_cost2) == 10):
    optimer_total_time_cost2 = []
    for i in range(10):
        total = 0
        for j in range (NUM_EPOCHS):
            print(optimer_time_cost2[i][j])
            total += optimer_time_cost2[i][j]
        optimer_total_time_cost2.append(total)
```

```python
plt.title('Optimizer Train Total Run Time Comparison')
plt.ylabel('Train Loss')
plt.xlabel('Epoch')
plt.scatter(0,optimer_total_time_cost2[0],label=str(optimer_name2[0]),color='green
plt.scatter(1,optimer_total_time_cost2[1],label=str(optimer_name2[1]),color='red')
plt.scatter(2,optimer_total_time_cost2[2],label=str(optimer_name2[2]),color='blue
plt.scatter(3,optimer_total_time_cost2[3],label=str(optimer_name2[3]),color='cyan
plt.scatter(4,optimer_total_time_cost2[4],label=str(optimer_name2[4]),color='mager
plt.scatter(5,optimer_total_time_cost2[5],label=str(optimer_name2[5]),color='yello
plt.scatter(6,optimer_total_time_cost2[6],label=str(optimer_name2[6]),color='black
plt.scatter(7,optimer_total_time_cost2[7],label=str(optimer_name2[7]),color='purpl
plt.scatter(8,optimer_total_time_cost2[8],label=str(optimer_name2[8]),color='pink
plt.scatter(9,optimer_total_time_cost2[9],label=str(optimer_name2[9]),color='brown
plt.grid(True)
plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
           str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(c
           str(optimer_name2[8]),str(optimer_name2[9])], loc=2)
plt.show()
```

```
22.42923069000244
22.03575849533081
21.90809464454651
21.33849549293518
22.12070894241333
21.207977056503296
21.12524652481079
22.091578722000122
21.231019020080566
20.94936752319336
20.988755226135254
22.303212881088257
20.912145853042603
20.901576042175293
21.715555667877197
22.2813458442688
22.454292058944702
20.88107180595398
21.178807735443115
21.97369122505188
20.71423649787903
20.834086894989014
21.88960337638855
21.055163860321045
20.789132595062256
21.03568172454834
22.253254175186157
20.757803678512573
21.624095916748047
```

20.846429109573364
23.50133728981018
21.080498695373535
20.702409505844116
20.92083191871643
22.024969577789307
20.883948802947998
20.7589910030365
21.449717044830322
21.496236085891724
20.78956699371338
21.28217053413391
21.924538373947144
20.776846170425415
20.700535774230957
22.22884750366211
21.56350564956665
20.81874656677246
21.394007921218872
23.011247873306274
20.795247554779053
20.84532332420349
20.812267780303955
22.275070190429688
20.762842178344727
20.919060468673706
21.108945846557617
21.872865676879883
20.82339859008789
20.731842517852783
22.555508375167847
21.28514075279236
20.776172161102295
22.461990356445312
22.303091526031494
20.868398427963257
20.84310746192932
20.866524934768677
22.318110942840576
20.80787682533264
20.793549060821533
20.830275297164917
22.05786418914795
20.795043468475342
21.45679783821106
21.592525959014893
21.334539890289307
21.039289712905884

21.8958899974823
22.145450830459595
20.812574863433838
21.816141605377197
20.95058846473694
22.567543983459473
20.859676361083984
22.20030379295349
21.023101568222046
20.987590551376343
21.077537775039673
22.13083291053772
20.85323166847229
20.798567533493042
21.821053504943848
21.85932230949402
20.928206205368042
20.948158025741577
22.241189002990723
22.280003309249878
20.85981583595276
20.861351251602173
22.113306283950806
20.85755205154419
20.813919067382812
20.797508001327515
22.195573329925537
20.749218702316284
20.737070560455322
21.38177990913391
22.034016847610474
20.821397304534912
20.728224515914917
22.282161951065063
22.36711025238037
20.742109775543213
20.76522731781006
22.276553869247437
20.718058109283447
20.83864402770996
20.780874252319336
22.248981952667236
20.760797262191772
20.47953724861145
20.75412678718567
22.204808950424194
20.97880721092224
20.81403923034668

21.496278285980225
21.644554376602173
20.928886890411377
21.62219786643982
22.355279207229614
20.806708097457886
20.794236421585083
21.059670209884644
23.489551782608032
20.869853496551514
20.857051849365234
20.827840089797974
22.250791549682617
20.77502465248108
20.776650428771973
21.190448999404907
21.82620930671692
21.58103585243225
20.837517499923706
22.14835238456726
20.930064916610718
20.798590660095215
22.37905216217041
22.250690460205078
20.922422647476196
20.777000904083252
20.760912656784058
22.25983452796936
20.82355546951294
20.747795343399048
20.830416202545166
22.12186574935913
21.66226863861084
20.820285320281982
21.82592010498047
20.758229732513428
22.038937091827393
20.83705997467041
20.91964292526245
22.221453428268433
20.954660892486572
22.53923988342285
20.92583727836609
22.424652099609375
20.988992929458618
21.097344636917114
20.911219358444214
22.37939763069153

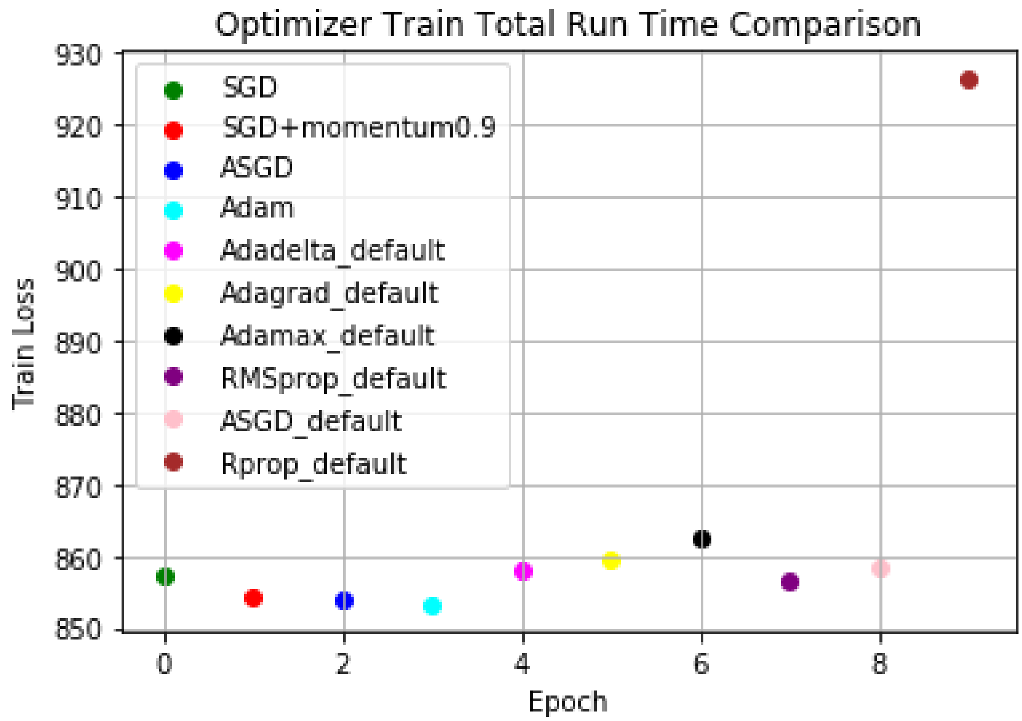
20.827630281448364
20.968465328216553
22.504855155944824
21.551945209503174
20.928365230560303
20.879690885543823
22.253753662109375
21.781203985214233
21.623628854751587
20.84201717376709
22.41798710823059
20.848080158233643
20.920857429504395
20.88617467880249
22.289814472198486
20.932315587997437
21.69871163368225
21.603018522262573
21.602075815200806
20.878568649291992
20.911405563354492
22.284514904022217
22.398690938949585
20.92375946044922
20.919060468673706
22.359917402267456
20.898556232452393
21.93619203567505
21.08270573616028
21.208574771881104
21.91174340248108
21.551580905914307
21.04027485847473
22.2557156085968
22.61259651184082
20.953967809677124
20.99371314048767
21.007105588912964
22.450904846191406
21.015406608581543
20.940542697906494
22.056631088256836
22.111185550689697
21.14973735809326
21.052252769470215
21.931382179260254
21.2577006816864
20.917197704315186

22.43786334991455
22.358187913894653
21.02865195274353
20.925820350646973
20.981807947158813
22.33217740058899
20.92373752593994
21.662782907485962
21.239054203033447
21.94881534576416
20.87835168838501
20.850823402404785
22.021872758865356
21.312347650527954
22.390477418899536
20.897218465805054
22.362406730651855
20.891057014465332
20.724937915802002
22.00249195098877
21.243712425231934
21.23816466331482
22.2292423248291
22.049875259399414
21.32120633125305
21.210060834884644
22.387352466583252
21.208637714385986
21.295859575271606
21.104896783828735
23.195470333099365
21.113304615020752
21.03507709503174
20.944205284118652
22.374748706817627
21.01891541481018
21.887879133224487
21.73018193244934
21.819302797317505
21.07967448234558
20.96636128425598
22.317509174346924
21.11035132408142
21.416374444961548
22.759358882904053
22.44571852684021
21.190085411071777
20.966177225112915

21.112340450286865
22.343995094299316
21.693760871887207
21.00046157836914
22.04397201538086
21.417667865753174
20.973711013793945
20.89094066619873
22.48668599128723
20.910603046417236
21.073868989944458
20.281720638275146
20.97692894935608
22.408581972122192
20.90725541114807
22.413835763931274
20.931402444839478
22.31988286972046
20.92158532142639
21.00292444229126
21.004684686660767
22.375543355941772
20.965242624282837
21.009610652923584
21.491018056869507
21.672693967819214
20.96899938583374
20.765011072158813
22.23760437965393
20.86014175415039
22.67426633834839
20.82118010520935
22.36017370223999
21.020330667495728
20.90294098854065
21.735311269760132
22.353575944900513
20.94617748260498
20.891746759414673
21.434842109680176
21.66800045967102
20.85370969772339
20.923628330230713
22.204758167266846
22.454827070236206
20.906435012817383
20.95933699607849
22.317883014678955

20.910624265670776
21.661466121673584
20.872251510620117
23.155891180038452
21.464565992355347
20.984652280807495
21.05830216407776
22.380713939666748
21.765608310699463
20.92652440071106
21.852323293685913
21.336247205734253
20.993821620941162
20.896556854248047
22.469802618026733
20.808311939239502
20.942216873168945
20.939860582351685
22.67508888244629
20.882913827896118
20.9955472946167
20.965774297714233
22.90931987762451
21.327720642089844
21.063002347946167
21.63566541671753
21.519047737121582
21.151105642318726
20.971673727035522
22.40164017677307
20.95595073699951
21.032946348190308
22.07201862335205
22.659844160079956
20.996187448501587
20.92403531074524
20.893766403198242
22.43104887008667
20.80897808074951
20.90363121032715
21.684406518936157
21.526373147964478
21.001543521881104
23.55530309677124
22.607478141784668
22.763533353805542
24.091240882873535
22.927775859832764

23.86912512779236
22.671830654144287
23.79189395904541
23.39036512374878
22.658430337905884
24.062125205993652
22.49925184249878
22.569798946380615
22.88583540916443
23.702516078948975
22.701101064682007
22.52009868621826
23.883825063705444
24.20812225341797
22.488388299942017
23.067634344100952
24.20920753479004
22.77599573135376
22.696553707122803
23.886849880218506
22.529088973999023
22.668351411819458
23.174968719482422
23.391643047332764
22.67641806602478
22.65504789352417
24.94974112510681
23.111512184143066
22.578299283981323
23.609617948532104
23.086546421051025
22.41592264175415
22.644678354263306
23.98894476890564
22.623257637023926



3.9 Part 2.8 - test the model

```
In [0]: def test(model, criterion, repeats=2):

          test_loss = 0.0
          test_acc = 0.0

          start = time
          for itr in range(repeats):
            for batch_idx, (images, labels) in enumerate(dataloaders['train']):
              #move to GPU
              images, labels = images.cuda(), labels.cuda()

              #forward
              outputs = model.forward(images)

              loss = criterion(outputs, labels)

              _, preds = torch.max(outputs.data, 1)

              test_loss += loss.item()
              test_acc += torch.sum(preds == labels).item()
```

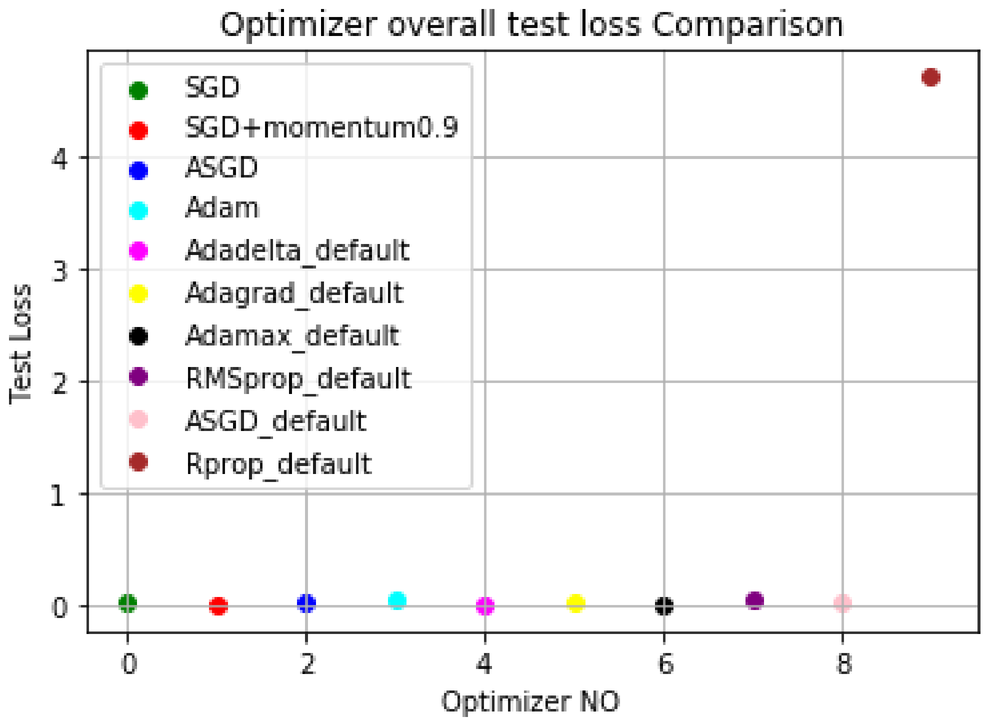
```
        test_loss /= (dataset_sizes['test']*repeats)
        test_acc /= (dataset_sizes['test']*repeats)

        print('Test Loss: %.4f Test Accuracy %.4f' % (test_loss, test_acc))
        return test_loss, test_acc
```

```
In [212]: start = time.time()
          test_loss, test_acc = test(model, criterion)
          end = time.time()
          test_timer = end - start
          optimizer_test_loss.append(test_loss)
          optimizer_test_accuracy.append(test_acc)
          optimizer_time_cost3.append(test_timer)
          print(len(optimizer_test_loss),len(optimizer_test_accuracy),len(optimizer_time_cost3
```

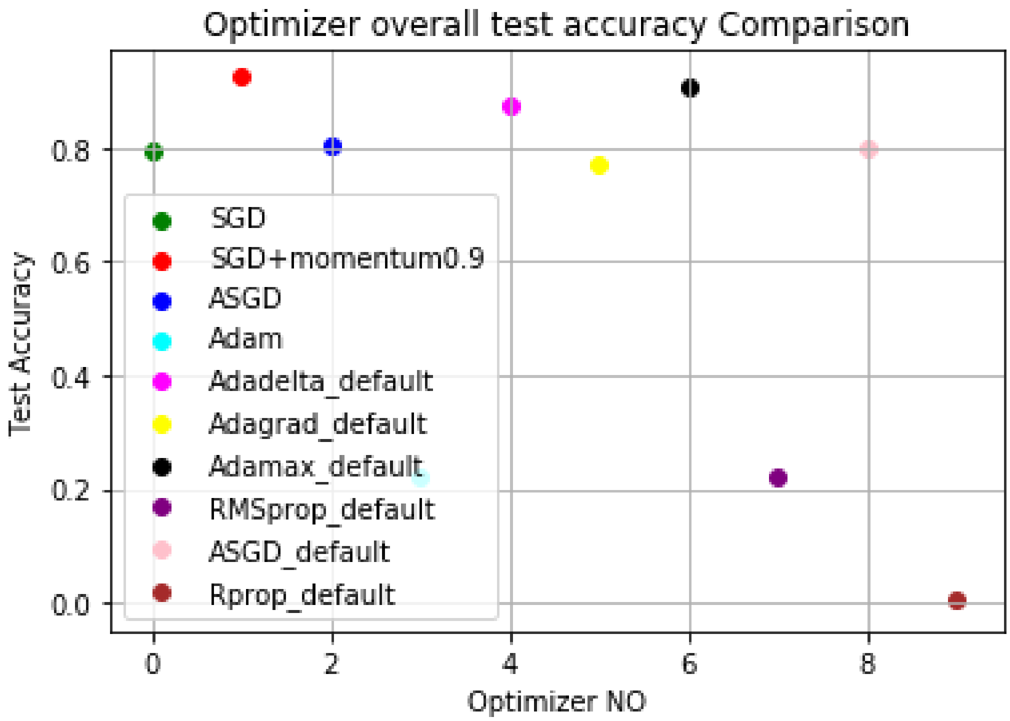
```
Test Loss: 4.7098 Test Accuracy 0.0073
10 10 10
```

3.10 Part 2.9 - Visualizing total optimizers test prediction result

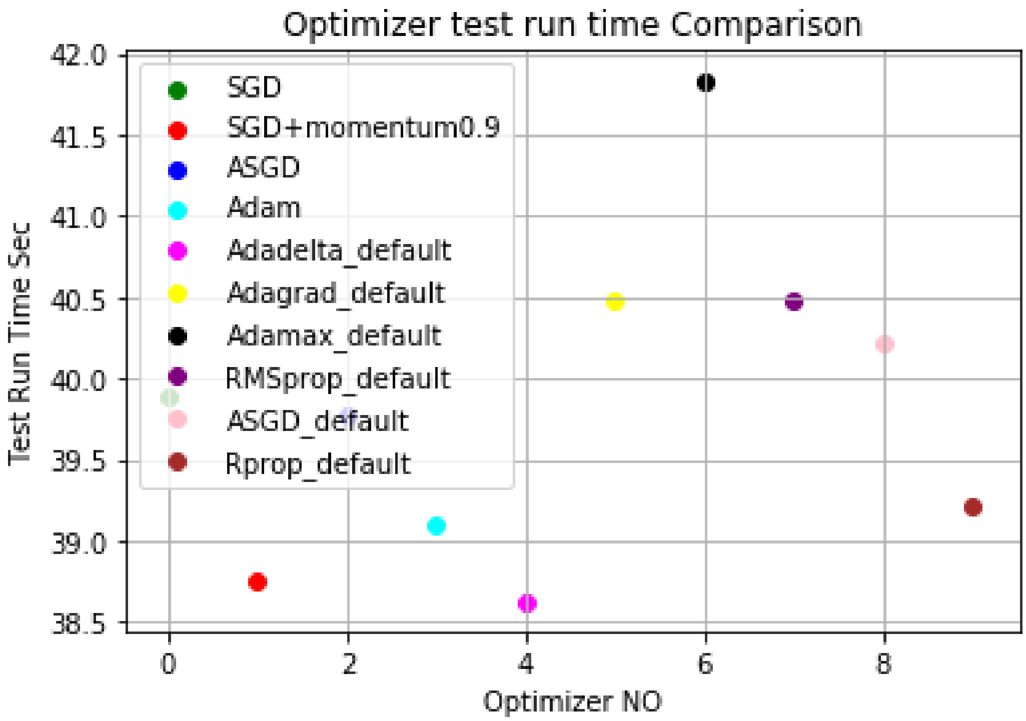
```
In [223]: if(len(optimizer_test_loss) == 10 and
             len(optimizer_test_accuracy) == 10 and
             len(optimizer_time_cost3) == 10):
             plt.title('Optimizer overall test loss Comparison')
             plt.ylabel('Test Loss')
             plt.xlabel('Optimizer NO')
             plt.scatter(0,optimizer_test_loss[0],color='green')
             plt.scatter(1,optimizer_test_loss[1],color='red')
             plt.scatter(2,optimizer_test_loss[2],color='blue')
             plt.scatter(3,optimizer_test_loss[3],color='cyan')
             plt.scatter(4,optimizer_test_loss[4],color='magenta')
             plt.scatter(5,optimizer_test_loss[5],color='yellow')
             plt.scatter(6,optimizer_test_loss[6],color='black')
             plt.scatter(7,optimizer_test_loss[7],color='purple')
             plt.scatter(8,optimizer_test_loss[8],color='pink')
             plt.scatter(9,optimizer_test_loss[9],color='brown')
             plt.grid(True)
             plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                        str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(
                        str(optimer_name2[8]),str(optimer_name2[9])], loc=2)
             plt.show()
```




```
In [230]: if(len(optimizer_test_loss) == 10 and
              len(optimizer_test_accuracy) == 10 and
              len(optimizer_time_cost3) == 10):
              plt.title('Optimizer overall test accuracy Comparison')
              plt.ylabel('Test Accuracy')
              plt.xlabel('Optimizer NO')
              plt.scatter(0,optimizer_test_accuracy[0],color='green')
              plt.scatter(1,optimizer_test_accuracy[1],color='red')
              plt.scatter(2,optimizer_test_accuracy[2],color='blue')
              plt.scatter(3,optimizer_test_accuracy[3],color='cyan')
              plt.scatter(4,optimizer_test_accuracy[4],color='magenta')
              plt.scatter(5,optimizer_test_accuracy[5],color='yellow')
              plt.scatter(6,optimizer_test_accuracy[6],color='black')
              plt.scatter(7,optimizer_test_accuracy[7],color='purple')
              plt.scatter(8,optimizer_test_accuracy[8],color='pink')
              plt.scatter(9,optimizer_test_accuracy[9],color='brown')
              plt.grid(True)
              plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                         str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(
                         str(optimer_name2[8]),str(optimer_name2[9])], loc=3)
              plt.show()
```




```
In [237]: if(len(optimizer_test_loss) == 10 and
             len(optimizer_test_accuracy) == 10 and
             len(optimizer_time_cost3) == 10):
             plt.title('Optimizer test run time Comparison')
             plt.ylabel('Test Run Time Sec')
             plt.xlabel('Optimizer NO')
             plt.scatter(0,optimizer_time_cost3[0],label= str(optimer_name2[0]),color='green')
             plt.scatter(1,optimizer_time_cost3[1],color='red')
             plt.scatter(2,optimizer_time_cost3[2],color='blue')
             plt.scatter(3,optimizer_time_cost3[3],color='cyan')
             plt.scatter(4,optimizer_time_cost3[4],color='magenta')
             plt.scatter(5,optimizer_time_cost3[5],color='yellow')
             plt.scatter(6,optimizer_time_cost3[6],color='black')
             plt.scatter(7,optimizer_time_cost3[7],color='purple')
             plt.scatter(8,optimizer_time_cost3[8],color='pink')
             plt.scatter(9,optimizer_time_cost3[9],color='brown')
             plt.grid(True)
             plt.legend([str(optimer_name2[0]),str(optimer_name2[1]),str(optimer_name2[2]),str
                         str(optimer_name2[4]),str(optimer_name2[5]),str(optimer_name2[6]),str(
                         str(optimer_name2[8]),str(optimer_name2[9])], loc=0)
             plt.show()
```



3.11 Part 2.10 - Visualizing the model image predictions

```
In [0]: def imshow(inp, title=None):
            """Imshow for Tensor."""
            inp = inp.numpy().transpose((1, 2, 0))
            inp = np.clip(inp, 0, 1)
            plt.imshow(inp)
            if title is not None:
                plt.title(title)
            plt.pause(1)  # pause a bit so that plots are updated

        def visualize_model(model, num_images=8):
            images_so_far = 0
            fig = plt.figure()

            for batch_idx, (images, labels) in enumerate(dataloaders['test']):
                #move to GPU
                images, labels = images.cuda(), labels.cuda()

                outputs = model(images)

                _, preds = torch.max(outputs.data, 1)
```

```
            for j in range(images.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('class: {} predicted: {}'.format(class_names[labels.data[j]],

                imshow(images.cpu().data[j])

                if images_so_far == num_images:
                    return
```

```
In [0]: visualize_model(model)
```