hw3_cnn_for_ieng-Alexnet

May 14, 2018

In [7]:
```
import math
import timeit
import numpy as np
import matplotlib.pyplot as plt

import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
import torch.nn.functional as F
from torch.autograd import Variable
import torch.backends.cudnn as cudnn
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
```

In [8]:
```
# Process data
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Load Train data and split into train set and valid set
num_train = 50000
indices = list(range(num_train))
train_index, valid_index = indices[10000:], indices[:10000]
train_sampler=SubsetRandomSampler(train_index)
valid_sampler=SubsetRandomSampler(valid_index)
train_dataset = torchvision.datasets.CIFAR10(root='/datasets/CIFAR-10', train=True,
                                             download=False, transform=transform)
valid_dataset = torchvision.datasets.CIFAR10(root='/datasets/CIFAR-10', train=True,
                                             download=False, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset,
                batch_size=32, sampler=train_sampler,
                num_workers=4, pin_memory=True)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                batch_size=10, sampler=valid_sampler,
                num_workers=4, pin_memory=True)
```

```
        # Load test data
        testset = torchvision.datasets.CIFAR10(root='/datasets/CIFAR-10', train=False,
                                               download=False, transform=transform)
        test_loader = torch.utils.data.DataLoader(testset,
                      batch_size=10,pin_memory=True,
                      shuffle=True, num_workers=4)

        # Define Class
        classes = ('plane', 'car', 'bird', 'cat',
                   'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

In [9]:
```
        # Save data in list for plot
        save_train_loss=[]
        save_valid_loss=[]
        save_test_loss=[]
        save_train_acc=[]
        save_valid_acc=[]
        save_test_acc=[]
```

In [13]:
```
        def main(iteration = 10, gpu_usage = False, opti = 'default',
                 Xavier = False, batch_norm = False):
            # detect GPU
            if gpu_usage:
                use_cuda  = torch.cuda.is_available()
            else:
                use_cuda = False
            # define network
            class Net(nn.Module):
                def __init__(self):
                    super(Net, self).__init__()
                    if batch_norm:
                        # batch normalization
                        self.conv1 = nn.Conv2d(3, 96, 3, stride=1, padding=1)
                        self.conv1_bn = nn.BatchNorm2d(96)
                        self.pool1 = nn.MaxPool2d(2, 2)
                        self.conv2 = nn.Conv2d(96, 256, 3, stride=1, padding=1)
                        self.conv2_bn = nn.BatchNorm2d(256)
                        self.pool2 = nn.MaxPool2d(2, 2)
                        self.conv3 = nn.Conv2d(256, 384, 3, stride=1, padding=1)
                        self.conv3_bn = nn.BatchNorm2d(384)
                        self.conv4 = nn.Conv2d(384, 384, 3, stride=1, padding=1)
                        self.conv4_bn = nn.BatchNorm2d(384)
                        self.conv5 = nn.Conv2d(384, 256, 3, stride=1, padding=1)
                        self.conv5_bn = nn.BatchNorm2d(256)
                        self.pool5 = nn.MaxPool2d(2, 2)
                        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
                        self.drop1 = nn.Dropout2d(0.2)
```

```
            self.fc2 = nn.Linear(4096, 4096)
            self.drop2 = nn.Dropout2d(0.2)
            self.fc3 = nn.Linear(4096, 10)
        else:
            # no batch normalization
            self.conv1 = nn.Conv2d(3, 96, 3, stride=1, padding=1)
            self.pool1 = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(96, 256, 3, stride=1, padding=1)
            self.pool2 = nn.MaxPool2d(2, 2)
            self.conv3 = nn.Conv2d(256, 384, 3, stride=1, padding=1)
            self.conv4 = nn.Conv2d(384, 384, 3, stride=1, padding=1)
            self.conv5 = nn.Conv2d(384, 256, 3, stride=1, padding=1)
            self.pool5 = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(256 * 4 * 4, 4096)
            self.drop1 = nn.Dropout2d(0.2)
            self.fc2 = nn.Linear(4096, 4096)
            self.drop2 = nn.Dropout2d(0.2)
            self.fc3 = nn.Linear(4096, 10)

        # Xavier init, need to check formula
        # Xavier Initialize for the whole network
        if Xavier:
            for m in self.modules():
                if isinstance(m, nn.Conv2d):
                    n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                    m.weight.data.normal_(0, math.sqrt(2. / n))
                elif isinstance(m, nn.BatchNorm2d):
                    m.weight.data.fill_(1)
                    m.bias.data.zero_()

    def forward(self, x):
        if batch_norm:
            # batch normalization
            x = self.pool1(F.relu(self.conv1_bn(self.conv1(x))))
            x = self.pool2(F.relu(self.conv2_bn(self.conv2(x))))
            x = F.relu(self.conv3_bn(self.conv3(x)))
            x = F.relu(self.conv4_bn(self.conv4(x)))
            x = self.pool5(F.relu(self.conv5_bn(self.conv5(x))))
            x = x.view(-1, 256 * 4 * 4)
            x = self.drop1(F.relu(self.fc1(x)))
            x = self.drop2(F.relu(self.fc2(x)))
            x = self.fc3(x)
        else:
            # no batch normalization
            x = self.pool1(F.relu(self.conv1(x)))
            x = self.pool2(F.relu(self.conv2(x)))
            x = F.relu(self.conv3(x))
            x = F.relu(self.conv4(x))
```

```
                x = self.pool5(F.relu(self.conv5(x)))
                x = x.view(-1, 256 * 4 * 4)
                x = self.drop1(F.relu(self.fc1(x)))
                x = self.drop2(F.relu(self.fc2(x)))
                x = self.fc3(x)
            return x

    net = Net()
#   net.apply(weights_init)

    ## using GPU
    if use_cuda:
        net.cuda()
        net = torch.nn.DataParallel(net, device_ids=range(torch.cuda.device_count()))
        cudnn.benchmark = True

    ## loss function
    criterion = nn.CrossEntropyLoss()
    if (opti == "Adam"):
        optimizer = optim.Adam(net.parameters(), lr = 0.001)
    else:
        optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, nesterov=True)

    ## starting training testinig validating
    start = timeit.default_timer()
    print('epoch, mini-batch\ttrain_loss\tvalid_loss\ttest_loss\ttrain_acc\tvalid_acc\t
    for epoch in range(iteration):  # loop over the dataset multiple times
        train_running_loss = 0.0
        for i, data in enumerate(train_loader, 0):
            # get the inputs
            inputs, labels = data
            # detect GPU
            if use_cuda:
                inputs, labels = inputs.cuda(), 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
            train_running_loss += loss.data[0]
```

```
if i % 501 == 500:    # print every 1 mini-batches
    train_loss = train_running_loss / 500
    #print('train_loss = '+str(train_loss))
    train_running_loss = 0.0
    # Train
    correct = 0
    total = 0
    for data in train_loader:
        images, labels = data
        # detect GPU
        if use_cuda:
            images, labels = images.cuda(), labels.cuda()
        outputs = net(Variable(images))
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
    train_acc = 100 * correct / total
    #print('train_acc = '+str(train_acc))
    # Valid
    correct = 0
    total = 0
    valid_loss = 0
    for data in valid_loader:
        images, labels = data
        # detect GPU
        if use_cuda:
            images, labels = images.cuda(), labels.cuda()
        v_images, v_labels = Variable(images), Variable(labels)
        outputs = net(v_images)
        valid_loss += criterion(outputs, v_labels).data[0]
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
    valid_loss /= 1000
    valid_acc = 100 * correct / total
    # Test
    correct = 0
    total = 0
    test_loss = 0
    for data in test_loader:
        images, labels = data
        # detect GPU
        if use_cuda:
            images, labels = images.cuda(), labels.cuda()
        v_images, v_labels = Variable(images), Variable(labels)
        outputs = net(v_images)
        test_loss += criterion(outputs, v_labels).data[0]
```

```
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum()
            test_loss /= 1000
            test_acc = 100 * correct / total
            # Save data for plot
            save_train_loss.append(train_loss)
            save_valid_loss.append(valid_loss)
            save_test_loss.append(test_loss)
            save_train_acc.append(train_acc)
            save_valid_acc.append(valid_acc)
            save_test_acc.append(test_acc)
            print('[%d, %5d]\t\t%.3f\t\t%.3f\t\t%.3f\t\t%.3f%%\t\t%.3f%%\t\t%.3f%%'
    print('Finished Training')
    stop = timeit.default_timer()
    print ('Time it takes: %.3f second' %(stop - start))
```

In [14]:
```
# iteration: howmany iteration you want. default = 10
# gpu_usage: use GPU or not. default = false
# opti: type of optimizer. default: SGD
main(iteration = 100, gpu_usage = True, opti = 'SGD', Xavier = True, batch_norm = True)
```

```
epoch, mini-batch        train_loss        valid_loss        test_loss        train_acc        v
[1,   501]                 1.542               1.334               1.338               54.227%
[1,  1002]                 1.194               1.157               1.165               62.138%
[2,   501]                 0.953               0.913               0.923               71.957%
[2,  1002]                 0.864               0.889               0.906               73.793%
[3,   501]                 0.714               0.852               0.897               76.500%
[3,  1002]                 0.714               0.771               0.810               79.815%
[4,   501]                 0.578               0.740               0.747               81.880%
[4,  1002]                 0.590               0.747               0.767               82.448%
[5,   501]                 0.471               0.758               0.766               84.290%
[5,  1002]                 0.489               0.664               0.676               86.972%
[6,   501]                 0.361               0.729               0.717               87.892%
[6,  1002]                 0.408               0.679               0.671               89.278%
[7,   501]                 0.286               0.695               0.727               91.263%
[7,  1002]                 0.320               0.698               0.719               91.302%
[8,   501]                 0.226               0.715               0.746               92.463%
[8,  1002]                 0.250               0.710               0.727               94.740%
[9,   501]                 0.174               0.756               0.783               94.203%
[9,  1002]                 0.203               0.702               0.737               95.647%
[10,   501]                0.127               0.768               0.822               95.422
[10,  1002]                0.144               0.716               0.734               96.207
[11,   501]                0.107               0.806               0.836               96.500
[11,  1002]                0.117               0.776               0.821               96.858
[12,   501]                0.081               0.781               0.794               98.020
[12,  1002]                0.095               0.801               0.857               97.670
[13,   501]                0.064               0.833               0.848               97.525
```

[13, 1002]	0.082	0.826	0.844	98.078
[14, 501]	0.051	0.886	0.950	98.040
[14, 1002]	0.061	0.923	0.952	97.862
[15, 501]	0.048	0.867	0.902	98.640
[15, 1002]	0.062	0.850	0.868	98.070
[16, 501]	0.045	0.843	0.863	99.188
[16, 1002]	0.038	0.838	0.879	99.203
[17, 501]	0.029	0.973	1.032	98.835
[17, 1002]	0.036	0.943	1.004	98.618
[18, 501]	0.032	0.867	0.925	99.237
[18, 1002]	0.038	0.888	0.951	99.040
[19, 501]	0.023	0.871	0.901	99.595
[19, 1002]	0.016	0.920	0.945	99.545
[20, 501]	0.022	0.921	0.977	99.297
[20, 1002]	0.019	0.895	0.958	99.465
[21, 501]	0.015	0.920	0.976	99.690
[21, 1002]	0.027	0.879	0.960	99.388
[22, 501]	0.027	0.882	0.931	99.595
[22, 1002]	0.014	0.920	1.024	99.740
[23, 501]	0.015	0.972	1.033	99.655
[23, 1002]	0.019	0.927	0.983	99.540
[24, 501]	0.023	0.978	1.039	99.218
[24, 1002]	0.018	0.941	1.025	99.535
[25, 501]	0.019	0.948	1.007	99.493
[25, 1002]	0.022	1.013	1.062	99.320
[26, 501]	0.014	0.933	0.975	99.713
[26, 1002]	0.013	0.996	1.061	99.510
[27, 501]	0.015	0.934	0.966	99.677
[27, 1002]	0.014	1.041	1.074	99.493
[28, 501]	0.018	0.983	1.041	99.547
[28, 1002]	0.012	0.961	1.000	99.740
[29, 501]	0.015	0.984	1.041	99.570
[29, 1002]	0.017	0.943	1.015	99.630
[30, 501]	0.012	0.981	1.062	99.767
[30, 1002]	0.012	0.950	1.011	99.805
[31, 501]	0.013	0.986	1.008	99.700
[31, 1002]	0.016	0.964	1.030	99.600
[32, 501]	0.011	0.992	1.046	99.567
[32, 1002]	0.008	1.015	1.073	99.755
[33, 501]	0.013	1.016	1.049	99.588
[33, 1002]	0.012	0.990	1.062	99.627
[34, 501]	0.012	0.994	1.053	99.662
[34, 1002]	0.013	1.020	1.076	99.608
[35, 501]	0.010	1.022	1.025	99.685
[35, 1002]	0.010	1.007	1.038	99.797
[36, 501]	0.006	1.002	1.033	99.892
[36, 1002]	0.006	1.044	1.065	99.793
[37, 501]	0.005	1.028	1.092	99.778

```
[37,  1002]                0.005                1.061                1.103                99.853
[38,   501]                0.007                1.039                1.104                99.820
[38,  1002]                0.008                1.047                1.101                99.832
[39,   501]                0.005                1.118                1.212                99.665
[39,  1002]                0.009                1.038                1.088                99.850
[40,   501]                0.007                1.182                1.254                99.233
[40,  1002]                0.011                1.086                1.160                99.800


Process Process-1127:
Process Process-1126:
Process Process-1128:
Process Process-1125:
```

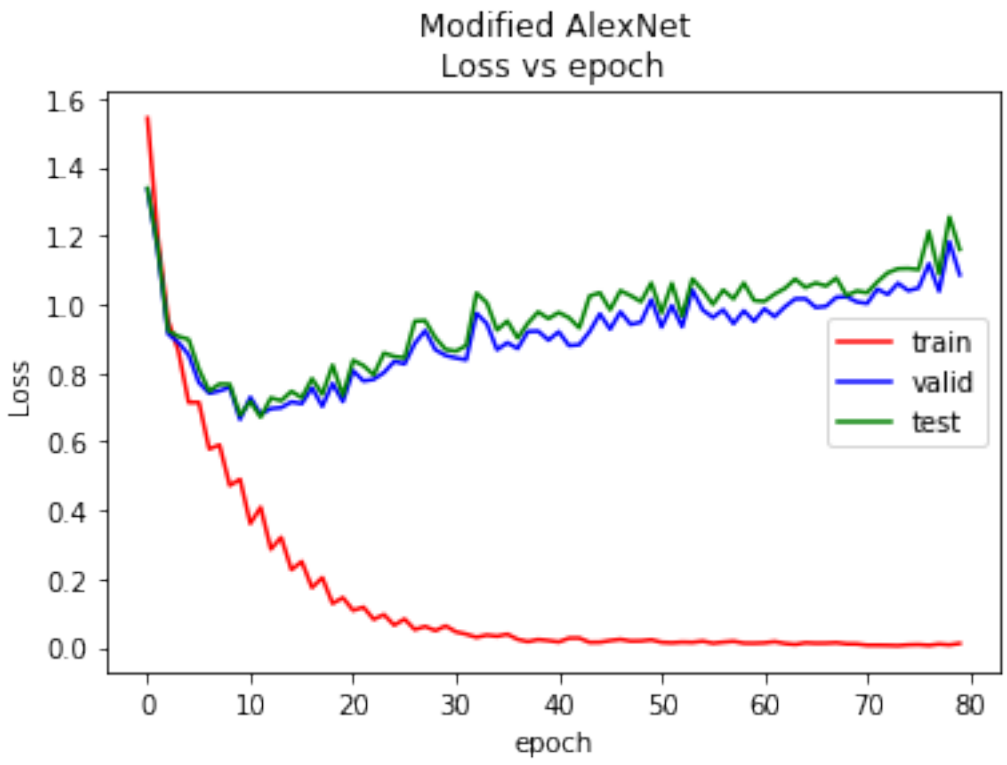
```
    KeyboardInterruptTraceback (most recent call last)

    <ipython-input-14-9eb1a9fc7a7f> in <module>()
      2 # gpu_usage: use GPU or not. default = false
      3 # opti: type of optimizer. default: SGD
----> 4 main(iteration = 100, gpu_usage = True, opti = 'SGD', Xavier = True, batch_norm = Tr


    <ipython-input-13-8f4b7d1c7234> in main(iteration, gpu_usage, opti, Xavier, batch_norm)
    118             outputs = net(inputs)
    119             loss = criterion(outputs, labels)
--> 120             loss.backward()
    121             optimizer.step()
    122


    /opt/conda/lib/python3.6/site-packages/torch/autograd/variable.py in backward(self, grad
    165                 Variable.
    166         """
--> 167         torch.autograd.backward(self, gradient, retain_graph, create_graph, retain_v
    168
    169     def register_hook(self, hook):


    /opt/conda/lib/python3.6/site-packages/torch/autograd/__init__.py in backward(variables,
     97
     98     Variable._execution_engine.run_backward(
---> 99         variables, grad_variables, retain_graph)
    100
    101
```

```
        KeyboardInterrupt:


Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 258, in _bootstrap
    self.run()
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 258, in _bootstrap
    self.run()
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 258, in _bootstrap
    self.run()
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 258, in _bootstrap
    self.run()
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/opt/conda/lib/python3.6/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 50, in _wor
    r = index_queue.get()
  File "/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 50, in _wor
    r = index_queue.get()
  File "/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 50, in _wor
    r = index_queue.get()
  File "/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 55, in _wor
    samples = collate_fn([dataset[i] for i in batch_indices])
  File "/opt/conda/lib/python3.6/multiprocessing/queues.py", line 334, in get
    with self._rlock:
  File "/opt/conda/lib/python3.6/multiprocessing/queues.py", line 334, in get
    with self._rlock:
  File "/opt/conda/lib/python3.6/multiprocessing/queues.py", line 335, in get
    res = self._reader.recv_bytes()
  File "/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 55, in <lis
    samples = collate_fn([dataset[i] for i in batch_indices])
  File "/opt/conda/lib/python3.6/multiprocessing/synchronize.py", line 96, in __enter__
    return self._semlock.__enter__()
  File "/opt/conda/lib/python3.6/multiprocessing/synchronize.py", line 96, in __enter__
    return self._semlock.__enter__()
  File "/opt/conda/lib/python3.6/multiprocessing/connection.py", line 216, in recv_bytes
    buf = self._recv_bytes(maxlength)
  File "/opt/conda/lib/python3.6/site-packages/torchvision-0.2.0-py3.6.egg/torchvision/datasets/
```

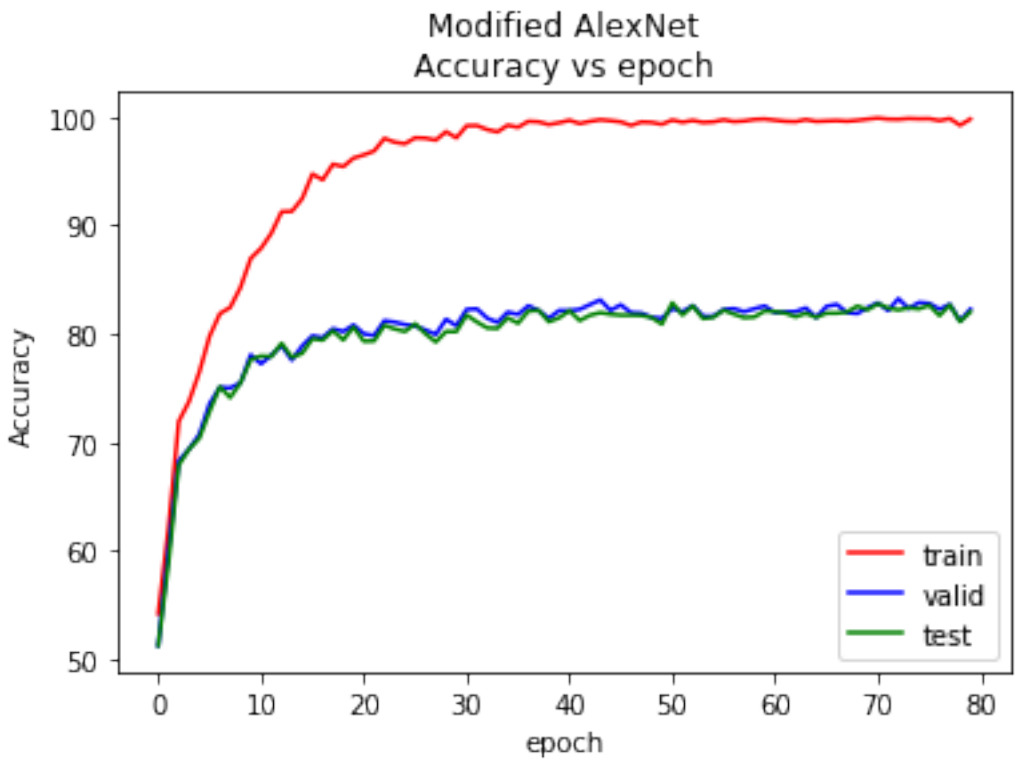
```
    img = Image.fromarray(img)
KeyboardInterrupt
KeyboardInterrupt
  File "/opt/conda/lib/python3.6/multiprocessing/connection.py", line 407, in _recv_bytes
    buf = self._recv(4)
  File "/opt/conda/lib/python3.6/site-packages/PIL/Image.py", line 2446, in fromarray
    obj = obj.tobytes()
  File "/opt/conda/lib/python3.6/multiprocessing/connection.py", line 379, in _recv
    chunk = read(handle, remaining)
KeyboardInterrupt
KeyboardInterrupt
```

In [18]:
```
def plot_accuracy(trainacc,validacc,testacc):
    plt.figure()
    plt.title('Modified AlexNet\nAccuracy vs epoch')
    plt.ylabel('Accuracy')
    plt.xlabel('epoch')
    plt.plot(trainacc,'red')
    plt.plot(validacc,'blue')
    plt.plot(testacc,'green')
    plt.legend(['train','valid','test'])
    plt.savefig('HW3_accuracy_model_3_32_batch_001')
    plt.show()

def plot_loss(trainloss,validloss,testloss):
    plt.figure()
    plt.title('Modified AlexNet\nLoss vs epoch')
    plt.ylabel('Loss')
    plt.xlabel('epoch')
    plt.plot(trainloss,'red')
    plt.plot(validloss,'blue')
    plt.plot(testloss,'green')
    plt.legend(['train','valid','test'])
    plt.savefig('HW3_loss_model_3_32_batch_001')
    plt.show()
```

In [19]:
```
plot_loss(save_train_loss,save_valid_loss,save_test_loss)
```



In [20]: `plot_accuracy(save_train_acc,save_valid_acc,save_test_acc)`


Modified AlexNet
Accuracy vs epoch
Accuracy
epoch
100
90
80
70
60
50
0
10
20
30
40
50
60
70
80
train
valid
test