Resnet 18 on faces

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import random
import resnet
```

In [2]:
```
batchsize = 75
rate = 0.1
epochs = 150
lr_decay = 0.84
lr_stride = 5
```

In [3]:
```
class FaceDataset(torch.utils.data.Dataset):

    def __init__(self, transform, train=True):
        self.image_prefix = "face_renders/face"
        self.image_suffix = ".jpg"
        self.vertex_prefix = "processed_faces/face"
        self.vertex_suffix = ".txt"
        self.count = 5000
        self.trainn = 4500

        self.train = train
        self.transform = transform

        shape = np.loadtxt(self.vertex_prefix + str(1) + self.vertex_suffix).s
hape
        tmp = np.zeros((self.count, shape[0], shape[1]))
        for i in range(self.count):
            tmp[i] = np.loadtxt(self.vertex_prefix + str(i + 1) + self.vertex_
suffix)

        self.mean = np.mean(tmp, axis=0)
        self.outputdim = shape[0] * shape[1]
        self.labels = [torch.from_numpy((lab - self.mean).reshape(self.outputd
im)).float() for lab in tmp]

        self.images = [plt.imread(self.image_prefix + str(i + 1) + self.image_
suffix) for i in range(self.count)]

        # simple version for working with CWD


    def __len__(self):
        if self.train:
            return self.trainn
        else:
            return self.count - self.trainn

    def __getitem__(self, idx):
        if not train:
            idx += self.trainn
        return (self.transform(self.images[idx]), self.labels[idx])
```

In [4]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = FaceDataset(transform, train=True)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batchsize,
                                          shuffle=True, num_workers=0)

testset = FaceDataset(transform, train=False)
testloader = torch.utils.data.DataLoader(trainset, batch_size=batchsize,
                                          shuffle=True, num_workers=0)
```

In [5]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("torch.cuda.is_available()   =", torch.cuda.is_available())
print("torch.cuda.device_count()   =", torch.cuda.device_count())
print("torch.cuda.device('cuda')   =", torch.cuda.device(0))
print("torch.cuda.current_device() =", torch.cuda.current_device())

def to_device(data, device):
    if isinstance(data, (list, tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)

trainloader = DeviceDataLoader(trainloader, device)
testloader = DeviceDataLoader(testloader, device)
```

```
torch.cuda.is_available()   = True
torch.cuda.device_count()   = 1
torch.cuda.device('cuda')   = <torch.cuda.device object at 0x0000026F2D14FA20
>
torch.cuda.current_device() = 0
```

In [6]:
```
model = resnet.resnet50(output_size=trainset.outputdim)
model.to(device)
optimizer = optim.SGD(model.parameters(), lr=rate)

criterion = nn.MSELoss()

def adjust_learning_rate(optimizer, epoch, decay, stride):
    lr = rate * (decay ** (epoch // stride))
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
```

In [7]:
```
def train(model, optimizer, criterion, epochs, trainloader, testloader):
    model.train()
    samples = 1
    losses = []
#     test_losses = []
    k = len(trainloader)// samples

    for epoch in range(epochs):  # loop over the dataset multiple times
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs
            inputs, labels = data

            # zero the parameter gradients
            optimizer.zero_grad()

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

            # print statistics
            running_loss += loss.item()
            if i % k == k - 1:

                losses.append(running_loss / k)

#                 testloss = 0
#                 total = 0
#                 iterations = 0
#                 with torch.no_grad():
#                     for data in testloader:
#                         images, labels = data
#                         outputs = model(images)
#                         testloss += criterion(outputs, labels)
#                         total += labels.size(0)
#                         iterations += 1
#                         if total > 200:
#                             break
#                 test_losses.append(testloss / iterations)

                print('[%d, %5d] loss: %.3f test_loss: %.3f' %(epoch + 1, i +
1,losses[-1], 0)) #test_losses[-1]

                running_loss = 0.0

        adjust_learning_rate(optimizer, epoch+1, lr_decay, lr_stride)

    print('Finished Training')
    plt.plot(np.arange(0, len(losses)/samples, 1.0/samples), losses)
    plt.title("loss")
    plt.xlabel("epoch")
    plt.ylabel("loss")
    plt.show()
```

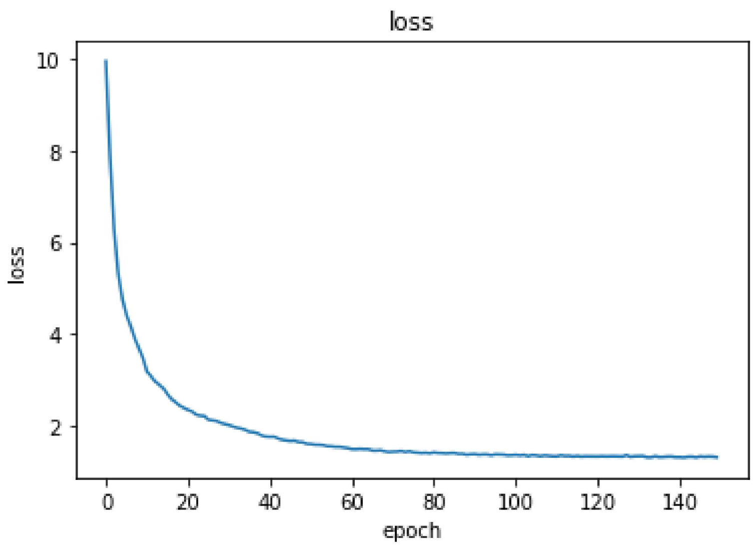
```
#     plt.plot(np.arange(0, len(test_losses)/samples, 1.0/samples), test_losse
s)
#     plt.title("test_loss")
#     plt.xlabel("epoch")
#     plt.ylabel("test_losses")
#     plt.show()
    model.eval()
```

In [8]:
```
train(model, optimizer, criterion, epochs, trainloader, testloader)
```

```
[1,    60] loss: 9.961 test_loss: 0.000
[2,    60] loss: 7.874 test_loss: 0.000
[3,    60] loss: 6.283 test_loss: 0.000
[4,    60] loss: 5.314 test_loss: 0.000
[5,    60] loss: 4.763 test_loss: 0.000
[6,    60] loss: 4.415 test_loss: 0.000
[7,    60] loss: 4.176 test_loss: 0.000
[8,    60] loss: 3.906 test_loss: 0.000
[9,    60] loss: 3.690 test_loss: 0.000
[10,    60] loss: 3.475 test_loss: 0.000
[11,    60] loss: 3.176 test_loss: 0.000
[12,    60] loss: 3.073 test_loss: 0.000
[13,    60] loss: 2.959 test_loss: 0.000
[14,    60] loss: 2.887 test_loss: 0.000
[15,    60] loss: 2.810 test_loss: 0.000
[16,    60] loss: 2.686 test_loss: 0.000
[17,    60] loss: 2.572 test_loss: 0.000
[18,    60] loss: 2.508 test_loss: 0.000
[19,    60] loss: 2.429 test_loss: 0.000
[20,    60] loss: 2.382 test_loss: 0.000
[21,    60] loss: 2.338 test_loss: 0.000
[22,    60] loss: 2.305 test_loss: 0.000
[23,    60] loss: 2.233 test_loss: 0.000
[24,    60] loss: 2.211 test_loss: 0.000
[25,    60] loss: 2.204 test_loss: 0.000
[26,    60] loss: 2.123 test_loss: 0.000
[27,    60] loss: 2.107 test_loss: 0.000
[28,    60] loss: 2.095 test_loss: 0.000
[29,    60] loss: 2.053 test_loss: 0.000
[30,    60] loss: 2.031 test_loss: 0.000
[31,    60] loss: 2.005 test_loss: 0.000
[32,    60] loss: 1.976 test_loss: 0.000
[33,    60] loss: 1.946 test_loss: 0.000
[34,    60] loss: 1.930 test_loss: 0.000
[35,    60] loss: 1.899 test_loss: 0.000
[36,    60] loss: 1.859 test_loss: 0.000
[37,    60] loss: 1.849 test_loss: 0.000
[38,    60] loss: 1.825 test_loss: 0.000
[39,    60] loss: 1.777 test_loss: 0.000
[40,    60] loss: 1.754 test_loss: 0.000
[41,    60] loss: 1.748 test_loss: 0.000
[42,    60] loss: 1.752 test_loss: 0.000
[43,    60] loss: 1.712 test_loss: 0.000
[44,    60] loss: 1.683 test_loss: 0.000
[45,    60] loss: 1.672 test_loss: 0.000
[46,    60] loss: 1.657 test_loss: 0.000
[47,    60] loss: 1.666 test_loss: 0.000
[48,    60] loss: 1.633 test_loss: 0.000
[49,    60] loss: 1.632 test_loss: 0.000
[50,    60] loss: 1.595 test_loss: 0.000
[51,    60] loss: 1.591 test_loss: 0.000
[52,    60] loss: 1.575 test_loss: 0.000
[53,    60] loss: 1.575 test_loss: 0.000
[54,    60] loss: 1.567 test_loss: 0.000
[55,    60] loss: 1.543 test_loss: 0.000
[56,    60] loss: 1.540 test_loss: 0.000
[57,    60] loss: 1.528 test_loss: 0.000
```

```
[58,    60] loss: 1.529 test_loss: 0.000
[59,    60] loss: 1.515 test_loss: 0.000
[60,    60] loss: 1.502 test_loss: 0.000
[61,    60] loss: 1.479 test_loss: 0.000
[62,    60] loss: 1.476 test_loss: 0.000
[63,    60] loss: 1.485 test_loss: 0.000
[64,    60] loss: 1.476 test_loss: 0.000
[65,    60] loss: 1.479 test_loss: 0.000
[66,    60] loss: 1.449 test_loss: 0.000
[67,    60] loss: 1.448 test_loss: 0.000
[68,    60] loss: 1.456 test_loss: 0.000
[69,    60] loss: 1.431 test_loss: 0.000
[70,    60] loss: 1.417 test_loss: 0.000
[71,    60] loss: 1.424 test_loss: 0.000
[72,    60] loss: 1.425 test_loss: 0.000
[73,    60] loss: 1.441 test_loss: 0.000
[74,    60] loss: 1.416 test_loss: 0.000
[75,    60] loss: 1.433 test_loss: 0.000
[76,    60] loss: 1.410 test_loss: 0.000
[77,    60] loss: 1.402 test_loss: 0.000
[78,    60] loss: 1.393 test_loss: 0.000
[79,    60] loss: 1.400 test_loss: 0.000
[80,    60] loss: 1.387 test_loss: 0.000
[81,    60] loss: 1.408 test_loss: 0.000
[82,    60] loss: 1.395 test_loss: 0.000
[83,    60] loss: 1.393 test_loss: 0.000
[84,    60] loss: 1.384 test_loss: 0.000
[85,    60] loss: 1.393 test_loss: 0.000
[86,    60] loss: 1.396 test_loss: 0.000
[87,    60] loss: 1.374 test_loss: 0.000
[88,    60] loss: 1.377 test_loss: 0.000
[89,    60] loss: 1.355 test_loss: 0.000
[90,    60] loss: 1.366 test_loss: 0.000
[91,    60] loss: 1.370 test_loss: 0.000
[92,    60] loss: 1.355 test_loss: 0.000
[93,    60] loss: 1.363 test_loss: 0.000
[94,    60] loss: 1.363 test_loss: 0.000
[95,    60] loss: 1.343 test_loss: 0.000
[96,    60] loss: 1.365 test_loss: 0.000
[97,    60] loss: 1.361 test_loss: 0.000
[98,    60] loss: 1.358 test_loss: 0.000
[99,    60] loss: 1.342 test_loss: 0.000
[100,    60] loss: 1.346 test_loss: 0.000
[101,    60] loss: 1.355 test_loss: 0.000
[102,    60] loss: 1.341 test_loss: 0.000
[103,    60] loss: 1.354 test_loss: 0.000
[104,    60] loss: 1.325 test_loss: 0.000
[105,    60] loss: 1.350 test_loss: 0.000
[106,    60] loss: 1.341 test_loss: 0.000
[107,    60] loss: 1.325 test_loss: 0.000
[108,    60] loss: 1.341 test_loss: 0.000
[109,    60] loss: 1.332 test_loss: 0.000
[110,    60] loss: 1.324 test_loss: 0.000
[111,    60] loss: 1.321 test_loss: 0.000
[112,    60] loss: 1.344 test_loss: 0.000
[113,    60] loss: 1.329 test_loss: 0.000
[114,    60] loss: 1.332 test_loss: 0.000
```

```
[115,    60] loss: 1.317 test_loss: 0.000
[116,    60] loss: 1.330 test_loss: 0.000
[117,    60] loss: 1.316 test_loss: 0.000
[118,    60] loss: 1.323 test_loss: 0.000
[119,    60] loss: 1.321 test_loss: 0.000
[120,    60] loss: 1.317 test_loss: 0.000
[121,    60] loss: 1.319 test_loss: 0.000
[122,    60] loss: 1.324 test_loss: 0.000
[123,    60] loss: 1.320 test_loss: 0.000
[124,    60] loss: 1.323 test_loss: 0.000
[125,    60] loss: 1.324 test_loss: 0.000
[126,    60] loss: 1.319 test_loss: 0.000
[127,    60] loss: 1.317 test_loss: 0.000
[128,    60] loss: 1.348 test_loss: 0.000
[129,    60] loss: 1.311 test_loss: 0.000
[130,    60] loss: 1.323 test_loss: 0.000
[131,    60] loss: 1.327 test_loss: 0.000
[132,    60] loss: 1.329 test_loss: 0.000
[133,    60] loss: 1.299 test_loss: 0.000
[134,    60] loss: 1.299 test_loss: 0.000
[135,    60] loss: 1.325 test_loss: 0.000
[136,    60] loss: 1.304 test_loss: 0.000
[137,    60] loss: 1.307 test_loss: 0.000
[138,    60] loss: 1.316 test_loss: 0.000
[139,    60] loss: 1.317 test_loss: 0.000
[140,    60] loss: 1.303 test_loss: 0.000
[141,    60] loss: 1.303 test_loss: 0.000
[142,    60] loss: 1.295 test_loss: 0.000
[143,    60] loss: 1.312 test_loss: 0.000
[144,    60] loss: 1.314 test_loss: 0.000
[145,    60] loss: 1.297 test_loss: 0.000
[146,    60] loss: 1.320 test_loss: 0.000
[147,    60] loss: 1.310 test_loss: 0.000
[148,    60] loss: 1.318 test_loss: 0.000
[149,    60] loss: 1.315 test_loss: 0.000
[150,    60] loss: 1.302 test_loss: 0.000
Finished Training
```



In [10]:
```
torch.save(model.state_dict(), "res50b" + str(batchsize) + "r" + str(rate) +
"e" + str(epochs) + ".statedict")
```

In [11]:
```
with torch.no_grad():
    d = next(testloader.__iter__())
    images, labels = d
    outputs = model(images)

    print(trainset.mean)
    print(outputs[0])
    print(labels[0])
    print(np.linalg.norm((labels[0] - outputs[0]).to('cpu').numpy()))
```

```
[[-57.42849171  43.50585649  81.09827847]
 [-57.10330025  40.76217867  80.77729131]
 [-56.4013172   36.99098729  79.87123442]
 ...
 [ 54.11570628 -43.99206555  66.38817782]
 [ 55.6071899  -46.20434955  60.50756982]
 [ 56.74632444 -47.9137754   53.69290953]]
tensor([-1.9255, -1.2657, -1.7704,  ...,  1.6352, -0.1930,  0.1472], device
='cuda:0')
tensor([-2.2897, -0.7128, -3.1278,  ...,  1.4948, -1.4139,  1.3726], device
='cuda:0')
46.01323
```

In [12]:
```
np.mean(np.std([l.numpy() for l in trainset.labels], axis=0))
```

Out[12]:
```
3.0160196
```