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]

        # 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
        y = self.labels[idx]
        x = plt.imread(self.image_prefix + str(idx + 1) + self.image_suffix)

        sample = (x,y)
        sample = (self.transform(sample[0]), sample[1])

        return sample
```

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 0x0000024D22AED7F0
>
torch.cuda.current_device() = 0
```

In [6]:
```
model = resnet.resnet34(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],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_losses)
    plt.title("test_loss")
    plt.xlabel("epoch")
    plt.ylabel("test_losses")
    plt.show()
    model.eval()
```

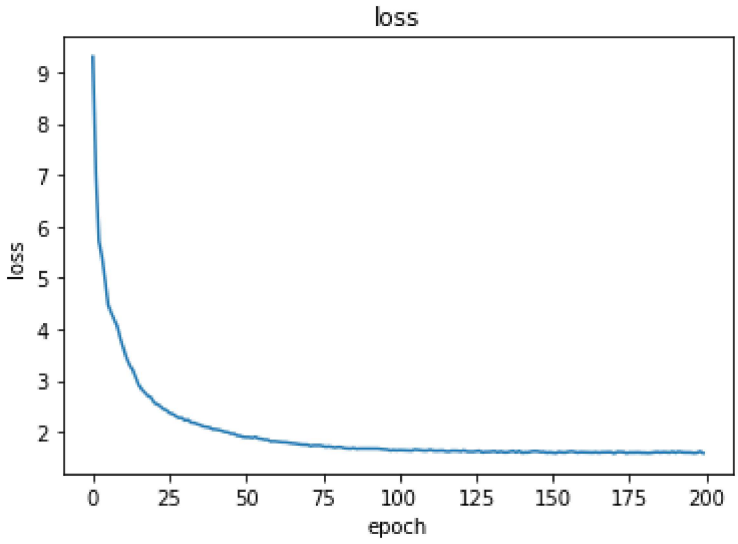
In [8]:
```
train(model, optimizer, criterion, epochs, trainloader, testloader)
```

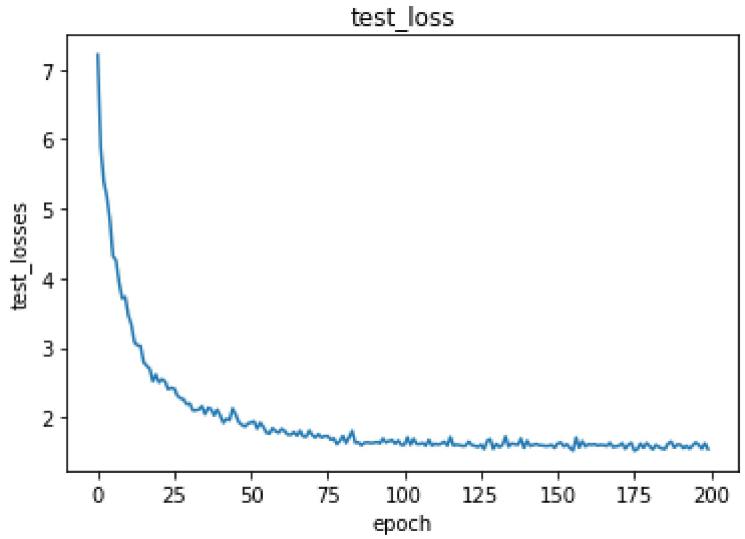
```
[1,    60] loss: 9.306 test_loss: 7.216
[2,    60] loss: 7.020 test_loss: 5.891
[3,    60] loss: 5.709 test_loss: 5.392
[4,    60] loss: 5.402 test_loss: 5.171
[5,    60] loss: 4.956 test_loss: 4.837
[6,    60] loss: 4.471 test_loss: 4.317
[7,    60] loss: 4.319 test_loss: 4.257
[8,    60] loss: 4.184 test_loss: 3.938
[9,    60] loss: 4.048 test_loss: 3.707
[10,    60] loss: 3.812 test_loss: 3.733
[11,    60] loss: 3.623 test_loss: 3.466
[12,    60] loss: 3.435 test_loss: 3.334
[13,    60] loss: 3.300 test_loss: 3.079
[14,    60] loss: 3.209 test_loss: 3.034
[15,    60] loss: 3.056 test_loss: 3.028
[16,    60] loss: 2.909 test_loss: 2.792
[17,    60] loss: 2.825 test_loss: 2.740
[18,    60] loss: 2.765 test_loss: 2.699
[19,    60] loss: 2.699 test_loss: 2.518
[20,    60] loss: 2.668 test_loss: 2.615
[21,    60] loss: 2.569 test_loss: 2.502
[22,    60] loss: 2.535 test_loss: 2.548
[23,    60] loss: 2.500 test_loss: 2.515
[24,    60] loss: 2.445 test_loss: 2.402
[25,    60] loss: 2.420 test_loss: 2.425
[26,    60] loss: 2.367 test_loss: 2.413
[27,    60] loss: 2.348 test_loss: 2.310
[28,    60] loss: 2.309 test_loss: 2.277
[29,    60] loss: 2.272 test_loss: 2.259
[30,    60] loss: 2.274 test_loss: 2.189
[31,    60] loss: 2.222 test_loss: 2.199
[32,    60] loss: 2.234 test_loss: 2.101
[33,    60] loss: 2.180 test_loss: 2.101
[34,    60] loss: 2.177 test_loss: 2.114
[35,    60] loss: 2.151 test_loss: 2.159
[36,    60] loss: 2.136 test_loss: 2.046
[37,    60] loss: 2.104 test_loss: 2.136
[38,    60] loss: 2.095 test_loss: 2.119
[39,    60] loss: 2.086 test_loss: 2.029
[40,    60] loss: 2.050 test_loss: 2.112
[41,    60] loss: 2.047 test_loss: 2.024
[42,    60] loss: 2.038 test_loss: 1.920
[43,    60] loss: 2.021 test_loss: 1.980
[44,    60] loss: 1.998 test_loss: 1.960
[45,    60] loss: 1.993 test_loss: 2.128
[46,    60] loss: 1.964 test_loss: 2.040
[47,    60] loss: 1.965 test_loss: 1.930
[48,    60] loss: 1.927 test_loss: 1.893
[49,    60] loss: 1.923 test_loss: 1.867
[50,    60] loss: 1.899 test_loss: 1.910
[51,    60] loss: 1.894 test_loss: 1.928
[52,    60] loss: 1.894 test_loss: 1.941
[53,    60] loss: 1.881 test_loss: 1.836
[54,    60] loss: 1.901 test_loss: 1.923
[55,    60] loss: 1.869 test_loss: 1.863
[56,    60] loss: 1.862 test_loss: 1.780
[57,    60] loss: 1.846 test_loss: 1.767
```

```
[58,    60] loss: 1.846 test_loss: 1.843
[59,    60] loss: 1.810 test_loss: 1.802
[60,    60] loss: 1.816 test_loss: 1.779
[61,    60] loss: 1.809 test_loss: 1.833
[62,    60] loss: 1.803 test_loss: 1.801
[63,    60] loss: 1.796 test_loss: 1.750
[64,    60] loss: 1.795 test_loss: 1.750
[65,    60] loss: 1.784 test_loss: 1.786
[66,    60] loss: 1.775 test_loss: 1.735
[67,    60] loss: 1.771 test_loss: 1.806
[68,    60] loss: 1.759 test_loss: 1.725
[69,    60] loss: 1.751 test_loss: 1.719
[70,    60] loss: 1.748 test_loss: 1.807
[71,    60] loss: 1.747 test_loss: 1.744
[72,    60] loss: 1.724 test_loss: 1.708
[73,    60] loss: 1.734 test_loss: 1.756
[74,    60] loss: 1.732 test_loss: 1.709
[75,    60] loss: 1.735 test_loss: 1.729
[76,    60] loss: 1.721 test_loss: 1.730
[77,    60] loss: 1.707 test_loss: 1.680
[78,    60] loss: 1.716 test_loss: 1.695
[79,    60] loss: 1.704 test_loss: 1.613
[80,    60] loss: 1.693 test_loss: 1.661
[81,    60] loss: 1.705 test_loss: 1.731
[82,    60] loss: 1.701 test_loss: 1.629
[83,    60] loss: 1.681 test_loss: 1.712
[84,    60] loss: 1.676 test_loss: 1.804
[85,    60] loss: 1.671 test_loss: 1.635
[86,    60] loss: 1.685 test_loss: 1.640
[87,    60] loss: 1.666 test_loss: 1.597
[88,    60] loss: 1.673 test_loss: 1.634
[89,    60] loss: 1.672 test_loss: 1.639
[90,    60] loss: 1.671 test_loss: 1.629
[91,    60] loss: 1.669 test_loss: 1.633
[92,    60] loss: 1.672 test_loss: 1.646
[93,    60] loss: 1.669 test_loss: 1.626
[94,    60] loss: 1.672 test_loss: 1.693
[95,    60] loss: 1.663 test_loss: 1.638
[96,    60] loss: 1.660 test_loss: 1.660
[97,    60] loss: 1.653 test_loss: 1.671
[98,    60] loss: 1.635 test_loss: 1.622
[99,    60] loss: 1.645 test_loss: 1.666
[100,    60] loss: 1.637 test_loss: 1.601
[101,    60] loss: 1.643 test_loss: 1.605
[102,    60] loss: 1.641 test_loss: 1.712
[103,    60] loss: 1.639 test_loss: 1.605
[104,    60] loss: 1.637 test_loss: 1.691
[105,    60] loss: 1.632 test_loss: 1.616
[106,    60] loss: 1.654 test_loss: 1.617
[107,    60] loss: 1.646 test_loss: 1.629
[108,    60] loss: 1.639 test_loss: 1.592
[109,    60] loss: 1.633 test_loss: 1.675
[110,    60] loss: 1.634 test_loss: 1.593
[111,    60] loss: 1.651 test_loss: 1.621
[112,    60] loss: 1.633 test_loss: 1.606
[113,    60] loss: 1.629 test_loss: 1.621
[114,    60] loss: 1.633 test_loss: 1.648
```

```
[115,    60] loss: 1.635 test_loss: 1.591
[116,    60] loss: 1.621 test_loss: 1.716
[117,    60] loss: 1.615 test_loss: 1.593
[118,    60] loss: 1.631 test_loss: 1.608
[119,    60] loss: 1.630 test_loss: 1.599
[120,    60] loss: 1.629 test_loss: 1.588
[121,    60] loss: 1.618 test_loss: 1.592
[122,    60] loss: 1.639 test_loss: 1.650
[123,    60] loss: 1.614 test_loss: 1.599
[124,    60] loss: 1.611 test_loss: 1.590
[125,    60] loss: 1.612 test_loss: 1.574
[126,    60] loss: 1.621 test_loss: 1.606
[127,    60] loss: 1.630 test_loss: 1.545
[128,    60] loss: 1.599 test_loss: 1.663
[129,    60] loss: 1.610 test_loss: 1.685
[130,    60] loss: 1.609 test_loss: 1.547
[131,    60] loss: 1.615 test_loss: 1.618
[132,    60] loss: 1.622 test_loss: 1.573
[133,    60] loss: 1.601 test_loss: 1.606
[134,    60] loss: 1.618 test_loss: 1.725
[135,    60] loss: 1.601 test_loss: 1.579
[136,    60] loss: 1.600 test_loss: 1.624
[137,    60] loss: 1.615 test_loss: 1.614
[138,    60] loss: 1.626 test_loss: 1.604
[139,    60] loss: 1.599 test_loss: 1.695
[140,    60] loss: 1.627 test_loss: 1.561
[141,    60] loss: 1.591 test_loss: 1.665
[142,    60] loss: 1.602 test_loss: 1.589
[143,    60] loss: 1.599 test_loss: 1.604
[144,    60] loss: 1.617 test_loss: 1.616
[145,    60] loss: 1.623 test_loss: 1.598
[146,    60] loss: 1.619 test_loss: 1.596
[147,    60] loss: 1.606 test_loss: 1.587
[148,    60] loss: 1.610 test_loss: 1.589
[149,    60] loss: 1.593 test_loss: 1.603
[150,    60] loss: 1.601 test_loss: 1.561
[151,    60] loss: 1.592 test_loss: 1.624
[152,    60] loss: 1.586 test_loss: 1.638
[153,    60] loss: 1.610 test_loss: 1.582
[154,    60] loss: 1.599 test_loss: 1.612
[155,    60] loss: 1.596 test_loss: 1.561
[156,    60] loss: 1.612 test_loss: 1.521
[157,    60] loss: 1.619 test_loss: 1.708
[158,    60] loss: 1.610 test_loss: 1.565
[159,    60] loss: 1.595 test_loss: 1.659
[160,    60] loss: 1.605 test_loss: 1.581
[161,    60] loss: 1.602 test_loss: 1.613
[162,    60] loss: 1.602 test_loss: 1.604
[163,    60] loss: 1.600 test_loss: 1.613
[164,    60] loss: 1.596 test_loss: 1.594
[165,    60] loss: 1.604 test_loss: 1.603
[166,    60] loss: 1.593 test_loss: 1.599
[167,    60] loss: 1.602 test_loss: 1.587
[168,    60] loss: 1.601 test_loss: 1.632
[169,    60] loss: 1.605 test_loss: 1.574
[170,    60] loss: 1.604 test_loss: 1.584
[171,    60] loss: 1.578 test_loss: 1.591
```

```
[172,    60] loss: 1.610 test_loss: 1.628
[173,    60] loss: 1.608 test_loss: 1.549
[174,    60] loss: 1.606 test_loss: 1.603
[175,    60] loss: 1.583 test_loss: 1.633
[176,    60] loss: 1.597 test_loss: 1.521
[177,    60] loss: 1.595 test_loss: 1.542
[178,    60] loss: 1.599 test_loss: 1.623
[179,    60] loss: 1.599 test_loss: 1.562
[180,    60] loss: 1.591 test_loss: 1.630
[181,    60] loss: 1.592 test_loss: 1.583
[182,    60] loss: 1.586 test_loss: 1.538
[183,    60] loss: 1.590 test_loss: 1.602
[184,    60] loss: 1.600 test_loss: 1.573
[185,    60] loss: 1.608 test_loss: 1.550
[186,    60] loss: 1.596 test_loss: 1.544
[187,    60] loss: 1.605 test_loss: 1.621
[188,    60] loss: 1.603 test_loss: 1.652
[189,    60] loss: 1.595 test_loss: 1.566
[190,    60] loss: 1.613 test_loss: 1.608
[191,    60] loss: 1.594 test_loss: 1.605
[192,    60] loss: 1.615 test_loss: 1.558
[193,    60] loss: 1.602 test_loss: 1.589
[194,    60] loss: 1.597 test_loss: 1.551
[195,    60] loss: 1.598 test_loss: 1.597
[196,    60] loss: 1.597 test_loss: 1.638
[197,    60] loss: 1.586 test_loss: 1.603
[198,    60] loss: 1.592 test_loss: 1.548
[199,    60] loss: 1.619 test_loss: 1.627
[200,    60] loss: 1.584 test_loss: 1.543
Finished Training
```





In [9]:
```
torch.save(model.state_dict(), "res34b" + str(batchsize) + "r" + str(rate) +
"e" + str(epochs) + ".statedict")
```

In [10]:
```
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([-0.2974,  2.1401,  1.9206,  ...,  1.9475, -2.6860,  0.9224], device
='cuda:0')
tensor([ 0.0596,  0.2830,  0.4403,  ...,  2.5278, -5.4042,  1.1915], device
='cuda:0')
52.23364
```