Resnet 18 on faces

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import resnet
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 = 200
lr_decay = 0.85
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 0x00000272CB0349E8
>
torch.cuda.current_device() = 0
```

In [6]:
```
model = resnet.resnet18(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
```

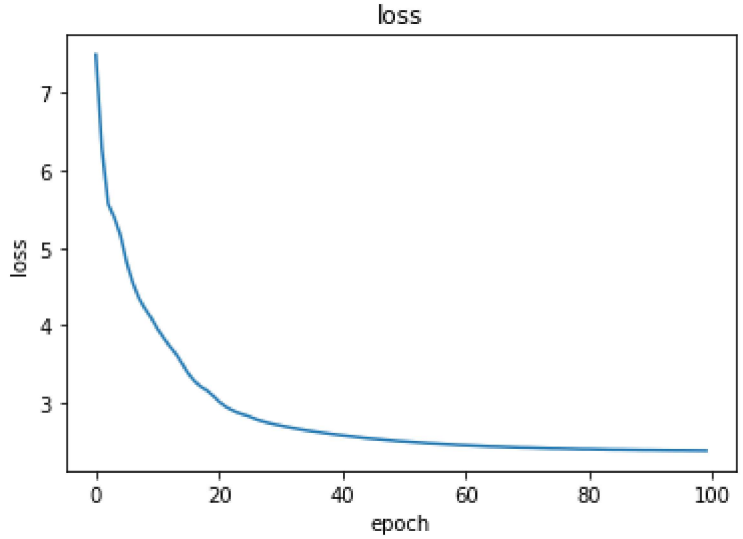
In [9]:
```
def train(model, optimizer, criterion, epochs, trainloader, testloader):
    model.train()
    samples = 1
    losses = []
    test_losses = []
    k = len(trainloader)// samples

    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()
```

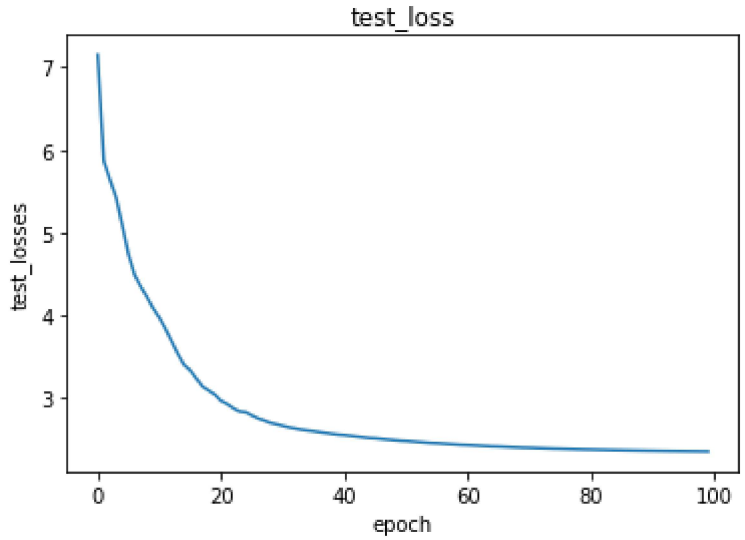
```
    plt.plot(np.arange(0, len(test_losses)/samples, 1.0/samples), test_losses)
    plt.title("test_loss")
    plt.xlabel("epoch")
    plt.ylabel("test_losses")
    plt.show()
```

In [10]:
```
train(model, optimizer, criterion, epochs, trainloader, testloader)
```

```
[1,    90] loss: 7.498 test_loss: 7.152
[2,    90] loss: 6.272 test_loss: 5.867
[3,    90] loss: 5.566 test_loss: 5.639
[4,    90] loss: 5.390 test_loss: 5.425
[5,    90] loss: 5.160 test_loss: 5.088
[6,    90] loss: 4.810 test_loss: 4.738
[7,    90] loss: 4.547 test_loss: 4.489
[8,    90] loss: 4.346 test_loss: 4.347
[9,    90] loss: 4.209 test_loss: 4.223
[10,    90] loss: 4.093 test_loss: 4.085
[11,    90] loss: 3.953 test_loss: 3.975
[12,    90] loss: 3.836 test_loss: 3.839
[13,    90] loss: 3.725 test_loss: 3.688
[14,    90] loss: 3.628 test_loss: 3.537
[15,    90] loss: 3.503 test_loss: 3.403
[16,    90] loss: 3.373 test_loss: 3.336
[17,    90] loss: 3.276 test_loss: 3.236
[18,    90] loss: 3.206 test_loss: 3.141
[19,    90] loss: 3.156 test_loss: 3.091
[20,    90] loss: 3.088 test_loss: 3.042
[21,    90] loss: 3.011 test_loss: 2.966
[22,    90] loss: 2.949 test_loss: 2.929
[23,    90] loss: 2.904 test_loss: 2.877
[24,    90] loss: 2.868 test_loss: 2.837
[25,    90] loss: 2.843 test_loss: 2.829
[26,    90] loss: 2.816 test_loss: 2.790
[27,    90] loss: 2.782 test_loss: 2.754
[28,    90] loss: 2.757 test_loss: 2.728
[29,    90] loss: 2.736 test_loss: 2.700
[30,    90] loss: 2.717 test_loss: 2.684
[31,    90] loss: 2.700 test_loss: 2.664
[32,    90] loss: 2.684 test_loss: 2.646
[33,    90] loss: 2.669 test_loss: 2.632
[34,    90] loss: 2.655 test_loss: 2.618
[35,    90] loss: 2.641 test_loss: 2.608
[36,    90] loss: 2.629 test_loss: 2.599
[37,    90] loss: 2.617 test_loss: 2.587
[38,    90] loss: 2.606 test_loss: 2.577
[39,    90] loss: 2.595 test_loss: 2.567
[40,    90] loss: 2.583 test_loss: 2.557
[41,    90] loss: 2.574 test_loss: 2.551
[42,    90] loss: 2.565 test_loss: 2.543
[43,    90] loss: 2.556 test_loss: 2.534
[44,    90] loss: 2.547 test_loss: 2.525
[45,    90] loss: 2.538 test_loss: 2.517
[46,    90] loss: 2.531 test_loss: 2.512
[47,    90] loss: 2.523 test_loss: 2.505
[48,    90] loss: 2.516 test_loss: 2.498
[49,    90] loss: 2.509 test_loss: 2.491
[50,    90] loss: 2.501 test_loss: 2.484
[51,    90] loss: 2.496 test_loss: 2.480
[52,    90] loss: 2.490 test_loss: 2.474
[53,    90] loss: 2.484 test_loss: 2.468
[54,    90] loss: 2.478 test_loss: 2.462
[55,    90] loss: 2.472 test_loss: 2.456
[56,    90] loss: 2.469 test_loss: 2.453
[57,    90] loss: 2.464 test_loss: 2.448
```

```
[58,    90] loss: 2.459 test_loss: 2.443
[59,    90] loss: 2.454 test_loss: 2.439
[60,    90] loss: 2.449 test_loss: 2.434
[61,    90] loss: 2.447 test_loss: 2.432
[62,    90] loss: 2.443 test_loss: 2.427
[63,    90] loss: 2.439 test_loss: 2.423
[64,    90] loss: 2.435 test_loss: 2.419
[65,    90] loss: 2.431 test_loss: 2.415
[66,    90] loss: 2.430 test_loss: 2.414
[67,    90] loss: 2.427 test_loss: 2.410
[68,    90] loss: 2.423 test_loss: 2.407
[69,    90] loss: 2.420 test_loss: 2.404
[70,    90] loss: 2.417 test_loss: 2.400
[71,    90] loss: 2.416 test_loss: 2.399
[72,    90] loss: 2.414 test_loss: 2.396
[73,    90] loss: 2.411 test_loss: 2.393
[74,    90] loss: 2.409 test_loss: 2.390
[75,    90] loss: 2.406 test_loss: 2.388
[76,    90] loss: 2.405 test_loss: 2.386
[77,    90] loss: 2.403 test_loss: 2.384
[78,    90] loss: 2.401 test_loss: 2.382
[79,    90] loss: 2.399 test_loss: 2.380
[80,    90] loss: 2.397 test_loss: 2.378
[81,    90] loss: 2.397 test_loss: 2.376
[82,    90] loss: 2.395 test_loss: 2.375
[83,    90] loss: 2.394 test_loss: 2.373
[84,    90] loss: 2.392 test_loss: 2.371
[85,    90] loss: 2.390 test_loss: 2.369
[86,    90] loss: 2.390 test_loss: 2.368
[87,    90] loss: 2.389 test_loss: 2.367
[88,    90] loss: 2.388 test_loss: 2.366
[89,    90] loss: 2.386 test_loss: 2.364
[90,    90] loss: 2.385 test_loss: 2.363
[91,    90] loss: 2.385 test_loss: 2.362
[92,    90] loss: 2.384 test_loss: 2.361
[93,    90] loss: 2.383 test_loss: 2.360
[94,    90] loss: 2.382 test_loss: 2.359
[95,    90] loss: 2.381 test_loss: 2.358
[96,    90] loss: 2.381 test_loss: 2.357
[97,    90] loss: 2.380 test_loss: 2.356
[98,    90] loss: 2.379 test_loss: 2.355
[99,    90] loss: 2.378 test_loss: 2.354
[100,    90] loss: 2.377 test_loss: 2.353
Finished Training
```





In [11]:
```
torch.save(model.state_dict(), "res18b" + str(batchsize) + "r" + str(rate) +
"e" + str(epochs) + ".statedict")
```

In [ ]:
```
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()))
```