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 [11]:
```
batchsize = 75
rate = 0.15
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 self.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(testset, batch_size=batchsize,
                                          shuffle=False, 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 0x0000019D7A45EE10
>
torch.cuda.current_device() = 0
```

In [18]:
```
model = resnet.resnet18(output_size=trainset.outputdim)
model.to(device)
#optimizer = optim.SGD(model.parameters(), lr=rate)
#optimizer = optim.Adam(model.parameters(), lr=rate)
optimizer = optim.SGD(model.parameters(), lr=rate, momentum=0.95)

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 [19]:
```
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()
```

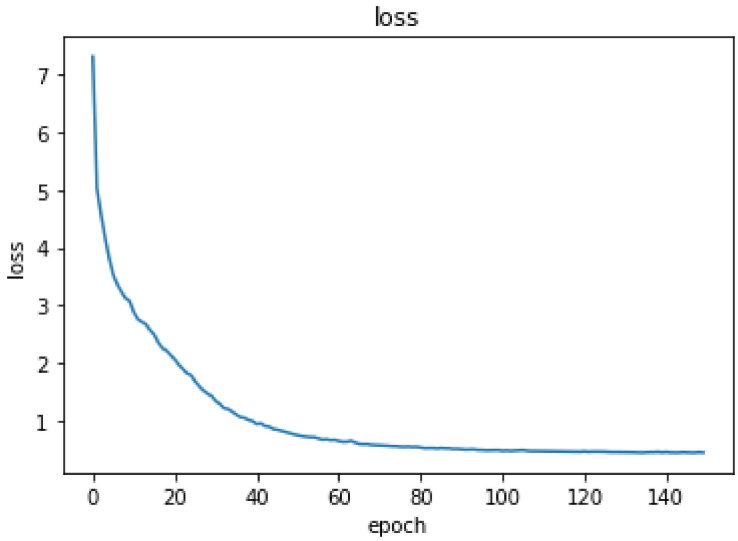
```
    plt.plot(np.arange(0, len(test_losses)/samples, 1.0/samples), test_losses)
    plt.title("test_loss")
    plt.xlabel("epoch")
    plt.ylabel("test_losses")
    plt.show()
    model.eval()
```

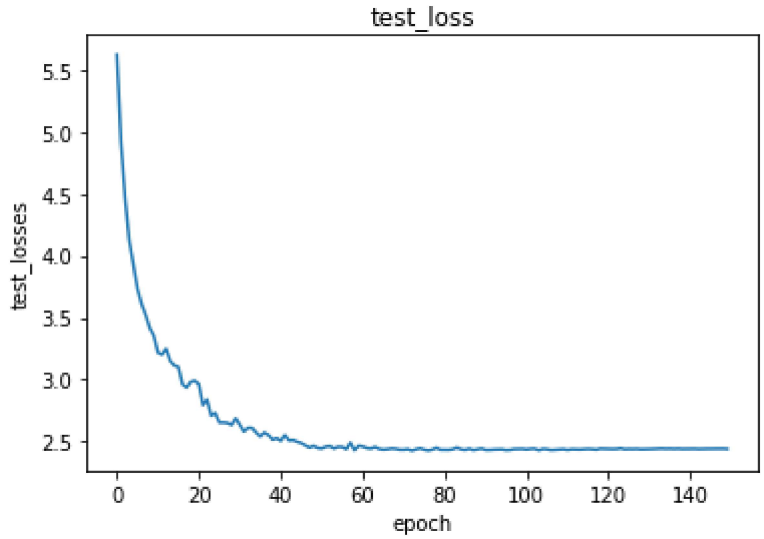
In [20]:
```
train(model, optimizer, criterion, epochs, trainloader, testloader)
```

```
[1,    60] loss: 7.306 test_loss: 5.625
[2,    60] loss: 5.029 test_loss: 4.915
[3,    60] loss: 4.577 test_loss: 4.466
[4,    60] loss: 4.170 test_loss: 4.125
[5,    60] loss: 3.817 test_loss: 3.933
[6,    60] loss: 3.526 test_loss: 3.733
[7,    60] loss: 3.362 test_loss: 3.611
[8,    60] loss: 3.235 test_loss: 3.522
[9,    60] loss: 3.123 test_loss: 3.416
[10,    60] loss: 3.076 test_loss: 3.353
[11,    60] loss: 2.882 test_loss: 3.217
[12,    60] loss: 2.762 test_loss: 3.201
[13,    60] loss: 2.713 test_loss: 3.248
[14,    60] loss: 2.673 test_loss: 3.150
[15,    60] loss: 2.566 test_loss: 3.113
[16,    60] loss: 2.495 test_loss: 3.103
[17,    60] loss: 2.355 test_loss: 2.961
[18,    60] loss: 2.260 test_loss: 2.934
[19,    60] loss: 2.215 test_loss: 2.981
[20,    60] loss: 2.139 test_loss: 2.991
[21,    60] loss: 2.064 test_loss: 2.962
[22,    60] loss: 1.977 test_loss: 2.791
[23,    60] loss: 1.897 test_loss: 2.838
[24,    60] loss: 1.826 test_loss: 2.707
[25,    60] loss: 1.792 test_loss: 2.729
[26,    60] loss: 1.682 test_loss: 2.653
[27,    60] loss: 1.598 test_loss: 2.652
[28,    60] loss: 1.527 test_loss: 2.649
[29,    60] loss: 1.473 test_loss: 2.633
[30,    60] loss: 1.433 test_loss: 2.686
[31,    60] loss: 1.346 test_loss: 2.634
[32,    60] loss: 1.294 test_loss: 2.579
[33,    60] loss: 1.224 test_loss: 2.609
[34,    60] loss: 1.206 test_loss: 2.608
[35,    60] loss: 1.164 test_loss: 2.571
[36,    60] loss: 1.110 test_loss: 2.540
[37,    60] loss: 1.067 test_loss: 2.572
[38,    60] loss: 1.054 test_loss: 2.549
[39,    60] loss: 1.019 test_loss: 2.514
[40,    60] loss: 0.998 test_loss: 2.526
[41,    60] loss: 0.947 test_loss: 2.504
[42,    60] loss: 0.960 test_loss: 2.550
[43,    60] loss: 0.913 test_loss: 2.507
[44,    60] loss: 0.903 test_loss: 2.510
[45,    60] loss: 0.863 test_loss: 2.497
[46,    60] loss: 0.849 test_loss: 2.486
[47,    60] loss: 0.829 test_loss: 2.468
[48,    60] loss: 0.810 test_loss: 2.450
[49,    60] loss: 0.791 test_loss: 2.464
[50,    60] loss: 0.770 test_loss: 2.447
[51,    60] loss: 0.753 test_loss: 2.442
[52,    60] loss: 0.738 test_loss: 2.458
[53,    60] loss: 0.730 test_loss: 2.462
[54,    60] loss: 0.721 test_loss: 2.442
[55,    60] loss: 0.721 test_loss: 2.456
[56,    60] loss: 0.698 test_loss: 2.455
[57,    60] loss: 0.677 test_loss: 2.437
```

```
[58,    60] loss: 0.684 test_loss: 2.490
[59,    60] loss: 0.668 test_loss: 2.429
[60,    60] loss: 0.673 test_loss: 2.465
[61,    60] loss: 0.653 test_loss: 2.459
[62,    60] loss: 0.639 test_loss: 2.446
[63,    60] loss: 0.639 test_loss: 2.442
[64,    60] loss: 0.657 test_loss: 2.455
[65,    60] loss: 0.628 test_loss: 2.440
[66,    60] loss: 0.603 test_loss: 2.434
[67,    60] loss: 0.598 test_loss: 2.437
[68,    60] loss: 0.599 test_loss: 2.442
[69,    60] loss: 0.587 test_loss: 2.440
[70,    60] loss: 0.583 test_loss: 2.433
[71,    60] loss: 0.581 test_loss: 2.433
[72,    60] loss: 0.573 test_loss: 2.438
[73,    60] loss: 0.571 test_loss: 2.425
[74,    60] loss: 0.563 test_loss: 2.436
[75,    60] loss: 0.563 test_loss: 2.444
[76,    60] loss: 0.555 test_loss: 2.434
[77,    60] loss: 0.552 test_loss: 2.426
[78,    60] loss: 0.555 test_loss: 2.434
[79,    60] loss: 0.550 test_loss: 2.448
[80,    60] loss: 0.552 test_loss: 2.433
[81,    60] loss: 0.539 test_loss: 2.432
[82,    60] loss: 0.529 test_loss: 2.432
[83,    60] loss: 0.532 test_loss: 2.437
[84,    60] loss: 0.531 test_loss: 2.450
[85,    60] loss: 0.523 test_loss: 2.436
[86,    60] loss: 0.532 test_loss: 2.431
[87,    60] loss: 0.525 test_loss: 2.442
[88,    60] loss: 0.524 test_loss: 2.430
[89,    60] loss: 0.514 test_loss: 2.436
[90,    60] loss: 0.516 test_loss: 2.443
[91,    60] loss: 0.510 test_loss: 2.431
[92,    60] loss: 0.504 test_loss: 2.431
[93,    60] loss: 0.507 test_loss: 2.434
[94,    60] loss: 0.508 test_loss: 2.435
[95,    60] loss: 0.497 test_loss: 2.436
[96,    60] loss: 0.497 test_loss: 2.430
[97,    60] loss: 0.491 test_loss: 2.431
[98,    60] loss: 0.492 test_loss: 2.438
[99,    60] loss: 0.492 test_loss: 2.439
[100,    60] loss: 0.495 test_loss: 2.439
[101,    60] loss: 0.483 test_loss: 2.435
[102,    60] loss: 0.489 test_loss: 2.440
[103,    60] loss: 0.482 test_loss: 2.441
[104,    60] loss: 0.485 test_loss: 2.428
[105,    60] loss: 0.491 test_loss: 2.439
[106,    60] loss: 0.495 test_loss: 2.437
[107,    60] loss: 0.484 test_loss: 2.430
[108,    60] loss: 0.479 test_loss: 2.433
[109,    60] loss: 0.481 test_loss: 2.437
[110,    60] loss: 0.479 test_loss: 2.437
[111,    60] loss: 0.478 test_loss: 2.431
[112,    60] loss: 0.472 test_loss: 2.437
[113,    60] loss: 0.475 test_loss: 2.435
[114,    60] loss: 0.472 test_loss: 2.434
```

```
[115,    60] loss: 0.469 test_loss: 2.436
[116,    60] loss: 0.468 test_loss: 2.439
[117,    60] loss: 0.474 test_loss: 2.437
[118,    60] loss: 0.470 test_loss: 2.434
[119,    60] loss: 0.466 test_loss: 2.441
[120,    60] loss: 0.467 test_loss: 2.440
[121,    60] loss: 0.474 test_loss: 2.440
[122,    60] loss: 0.465 test_loss: 2.438
[123,    60] loss: 0.470 test_loss: 2.439
[124,    60] loss: 0.469 test_loss: 2.444
[125,    60] loss: 0.469 test_loss: 2.437
[126,    60] loss: 0.468 test_loss: 2.439
[127,    60] loss: 0.462 test_loss: 2.438
[128,    60] loss: 0.462 test_loss: 2.439
[129,    60] loss: 0.465 test_loss: 2.436
[130,    60] loss: 0.461 test_loss: 2.437
[131,    60] loss: 0.461 test_loss: 2.437
[132,    60] loss: 0.459 test_loss: 2.439
[133,    60] loss: 0.457 test_loss: 2.441
[134,    60] loss: 0.457 test_loss: 2.442
[135,    60] loss: 0.452 test_loss: 2.440
[136,    60] loss: 0.457 test_loss: 2.441
[137,    60] loss: 0.460 test_loss: 2.439
[138,    60] loss: 0.459 test_loss: 2.441
[139,    60] loss: 0.463 test_loss: 2.439
[140,    60] loss: 0.455 test_loss: 2.440
[141,    60] loss: 0.463 test_loss: 2.440
[142,    60] loss: 0.454 test_loss: 2.440
[143,    60] loss: 0.454 test_loss: 2.439
[144,    60] loss: 0.452 test_loss: 2.440
[145,    60] loss: 0.459 test_loss: 2.440
[146,    60] loss: 0.454 test_loss: 2.441
[147,    60] loss: 0.452 test_loss: 2.440
[148,    60] loss: 0.452 test_loss: 2.440
[149,    60] loss: 0.459 test_loss: 2.441
[150,    60] loss: 0.453 test_loss: 2.439
Finished Training
```





In [21]:
```
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()))
```

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

In [ ]:
```
model = resnet.resnet101(output_size=trainset.outputdim)
model.load_state_dict(torch.load("res101b75r0.15e150.statedict"))
# model.to(device)
model.eval()
```

In [26]:
```
running_loss = 0
criterion = nn.MSELoss()
start = 4501
for i, data in enumerate(testloader, 0):
    # get the inputs
    inputs, labels = data

    # forward + backward + optimize
    outputs = model(inputs)

    for o in outputs:
        arr = o.detach().to("cpu").numpy().reshape((564,3)) + testset.mean

        np.savetxt("res50predictions/face" + str(start) + ".txt", arr, fmt="%.
9f")
        start += 1

    loss = criterion(outputs, labels)

    # print statistics
    running_loss += loss.item() * len(outputs)

print(running_loss / 500 * trainset.outputdim)
```

```
4014.2447912693024
```

In [ ]:
```
print (len(testloader))
```

In [ ]:
```
output = model(testset.transform(plt.imread("real/IMG_3368_cropped.png")[:,:,:
3]).unsqueeze(0))
output = output.detach().to("cpu").numpy().reshape((564,3)) + testset.mean
np.savetxt("res101predictions/facecustom.txt", output, fmt="%.9f")
```