hw4实验报告

52205903005 娄泽华

**hw4要求：**

Final Project: 人脸识别

数据：CMU Machine Learning Faces

<http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html>

描述：20人，每人32张脸图(含表情)

任务1：使用机器学习进行人脸分类识别，给出识别准确率

任务2：使用聚类或分类算法发现表情相似的脸图

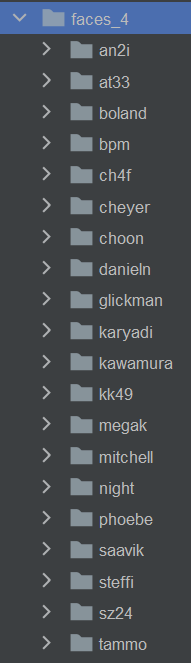
源码+实验报告

交给助教

Deadline: 学期末考试前

# 一、数据集简介

此数据集包含600多张人脸图像（本应20\*32=640张，实际为624张）。



数据集格式：

照片中的人的姿势(straight, left, right, up)、表情(neutral, happy, sad, angry)、眼部 (wearing sunglasses or not)各不相同。20个人，每个人约4\*4\*4=32张图片。每张图片的大小为：宽×高（32\*30），即30行32列。

# 二、实验思路

使用VGG网络实现人脸分类识别。先读入数据并解析、转换为标准的DataLoader对象，设计FaceVGG网络，输出线性层的20维向量作为，经验风险由交叉熵损失构成。

聚类算法采用K-Means算法，训练完成后，绘制出预测结果的平方误差损失与聚类中心个数k的关系图，并选取合适的k预测最终的类别标签（事实上从k大于等于20开始，平方误差开始收敛，详见实验结果，因此以n\_clusters=20为超参数训练并预测了聚类结果）。

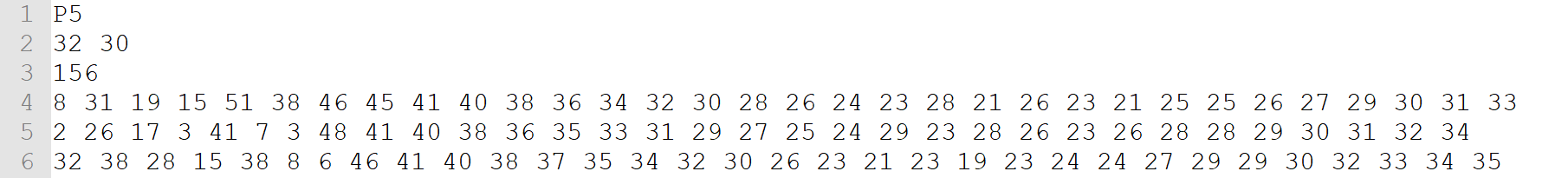
# 三、代码实现

## 3.1 预处理

读入数据并解析

def preprocess(faces\_groups\_dir):  
 X = []  
 y\_person = []  
 y\_facial\_expression = []  
 faces\_groups = os.listdir(faces\_groups\_dir)  
 for faces\_group in faces\_groups:  
 faces\_group\_path = os.path.join(faces\_groups\_dir, faces\_group)  
 person\_faces = os.listdir(faces\_group\_path)  
 for person\_face in person\_faces:  
 # image = pyplot.imread(os.path.join(faces\_group\_path, person\_face))  
 image = read\_pgm(os.path.join(faces\_group\_path, person\_face), '<')  
 X.append(image)  
 y\_person.append(y\_person\_dict\_reverse[faces\_group])  
 facial\_expression = person\_face.split('\_')[2]  
 y\_facial\_expression.append(y\_facial\_expression\_dict\_reverse[facial\_expression])  
 X = np.asarray(X, dtype=np.float32)  
 # X = X / np.max(X)  
 y\_person = np.asanyarray(y\_person, dtype=np.int32)  
 y\_facial\_expression = np.asanyarray(y\_facial\_expression, dtype=np.int32)  
 return X, y\_person, y\_facial\_expression

注意原始数据存放在.pgm文件中，存放格式为：



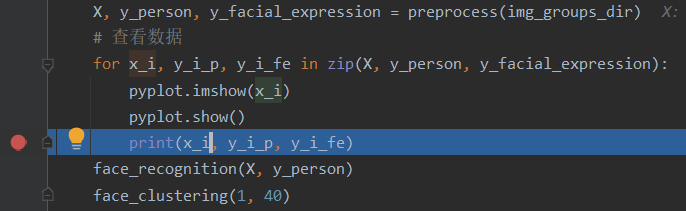
第1行：文件格式P5。

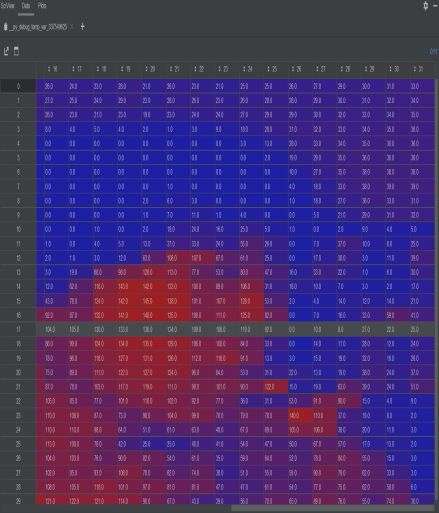
第2行：32为宽度（列数），30为高度（行数），共960个值。

第3行：最大像素值156，取值范围[0,255]。

第4-33行：960个像素值，30行，每行32个值，空格分隔。

查看X[0]如下：





因此读入函数为：

def read\_pgm(filename, byteorder='>'):  
 *"""Return image data from a raw PGM file as numpy array.  
  
 Format specification: http://netpbm.sourceforge.net/doc/pgm.html  
  
 """* with open(filename, 'rb') as f:  
 buffer = f.read()  
 try:  
 header, width, height, maxval = re.search(  
 b"(P5.\*[\r\n]"  
 b"(\d+)\s(\d+)[\r\n]"  
 b"(\d+).\*[\r\n])", buffer).groups()  
 except AttributeError:  
 raise ValueError("Not a raw PGM file: '%s'" % filename)  
 return np.frombuffer(buffer,  
 dtype='u1' if int(maxval) < 256 else byteorder + 'u2',  
 count=int(width) \* int(height),  
 offset=len(header)  
 ).reshape((int(height), int(width)))

选用交叉熵损失，选用随机梯度下降优化器，步长0.01：

criterion = nn.CrossEntropyLoss()  
opt = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight\_decay=5e-4)

## 3.2 设计FaceVGG

class FaceVGG(nn.Module):  
 def \_\_init\_\_(self):  
 super(FaceVGG, self).\_\_init\_\_()  
 self.features = self.\_make\_layers([8, 8, 'M', 16, 16, 'M', 16, 16, 'M'])  
 self.classifier = nn.Linear(192, 20)  
  
 def forward(self, x):  
 out = self.features(x)  
 out = out.view(out.size(0), -1)  
 out = self.classifier(out)  
 return out  
  
 def \_make\_layers(self, cfg):  
 layers = []  
 in\_channels = 1  
 for x in cfg:  
 if x == 'M':  
 layers += [nn.MaxPool2d(kernel\_size=2, stride=2)]  
 else:  
 layers += [nn.Conv2d(in\_channels, x, kernel\_size=3, padding=1),  
 nn.BatchNorm2d(x),  
 nn.ReLU(inplace=True)]  
 in\_channels = x # rotate  
 layers += [nn.AvgPool2d(kernel\_size=1, stride=1)]  
 return nn.Sequential(\*layers)

设计层次如下：

OrderedDict([('features', Sequential(

(0): Conv2d(1, 8, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(8, 8, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(4): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

(6): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(7): Conv2d(8, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(9): ReLU(inplace=True)

(10): Conv2d(16, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(12): ReLU(inplace=True)

(13): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(14): Conv2d(16, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(15): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(16): ReLU(inplace=True)

(17): Conv2d(16, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(18): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(19): ReLU(inplace=True)

(20): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(21): AvgPool2d(kernel\_size=1, stride=1, padding=0)

)), ('classifier', Linear(in\_features=192, out\_features=20, bias=True))])

## 3.3训练、预测

### 3.3.1 人脸识别

训练测试集划分比例为0.75:0.25，总数据624张图，训练集468张图，测试集156张图；采用交叉熵损失，随机梯度下降优化，并额外设置了训练退出条件（100%预测正确时跳出）：

def face\_recognition(X, y):  
 train\_dataloader, test\_dataloader = get\_dataloader(X, y, test\_size=0.25)  
 global model  
 model = FaceVGG()  
 lr = 0.01  
 criterion = nn.CrossEntropyLoss()  
 opt = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight\_decay=5e-4)  
 # print(model.features)  
 for epoch in range(EPOCHS):  
 model.train()  
 train\_correct = 0  
 train\_total = 0  
 # train  
 for batch\_idx, (inputs, targets) in enumerate(train\_dataloader):  
 opt.zero\_grad()  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 loss.backward()  
 opt.step()  
 \_, pre = outputs.max(1)  
 train\_correct += pre.eq(targets).sum().item()  
 train\_total += targets.size(0)  
  
 if (batch\_idx + 1) % 5 == 0 or batch\_idx + 1 >= len(train\_dataloader):  
 print(  
 'Epoch {:2d}: Process: [{:3d}/{:3d} ({:.1f}%)]\tAccuracy: [{:3d}/{:3d} ({:.1f}%)]\tMean Loss: {:.5f}'.format(  
 epoch + 1, batch\_idx \* BATCH\_SIZE + len(inputs), len(train\_dataloader.dataset),  
 100.0 \* (batch\_idx \* BATCH\_SIZE + len(inputs)) / len(train\_dataloader.dataset), train\_correct,  
 train\_total, 100.0 \* train\_correct / train\_total, loss.item()))  
 # test  
 model.eval()  
 test\_loss = 0  
 test\_correct = 0  
 test\_total = 0  
 with torch.no\_grad():  
 for inputs, targets in test\_dataloader:  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 test\_loss += loss.item()  
 \_, pre = outputs.max(1)  
 test\_correct += pre.eq(targets).sum().item()  
 test\_total += targets.size(0)  
 test\_loss /= len(test\_dataloader.dataset)  
 print(  
 '\nTest Set: Epoch {:2d}: Accuracy: [{:3d}/{:3d} ({:.1f}%)]\tMean Loss: {:.5f}\n'.format(  
 epoch + 1, test\_correct, test\_total, 100.0 \* test\_correct / test\_total, test\_loss))  
 if test\_correct >= test\_total:  
 print('Face Recognition training task breaks with an 100% accuracy!\n')  
 break

### 3.3.2 聚类

用K-Means算法聚类：设置k\_min=1, k\_max=40，遍历k值找出最适合此任务的超参数k0，并以k0作为超参数训练模型，输出预测结果。

def face\_clustering(k\_min, k\_max):  
 k\_min = max(1, int(k\_min))  
 k\_max = max(1, k\_min, int(k\_max))  
  
 features = model(torch.Tensor(X).unsqueeze(1)).detach().numpy()  
 SSE = []  
 for c\_i in range(k\_min, k\_max):  
 clf = KMeans(n\_clusters=c\_i)  
 clf.fit(features)  
 SSE.append(clf.inertia\_)  
 pyplot.plot([i for i in range(k\_min, k\_max)], SSE)  
 pyplot.xlabel('Sum of Squared Error')  
 pyplot.ylabel('k(number of clusters)')  
 pyplot.show()  
  
 # k = 20 开始收敛  
 clf = KMeans(n\_clusters=20)  
 clusters\_predicted = clf.fit(features).predict(features)  
 print("clusters\_predicted: \n", clusters\_predicted)

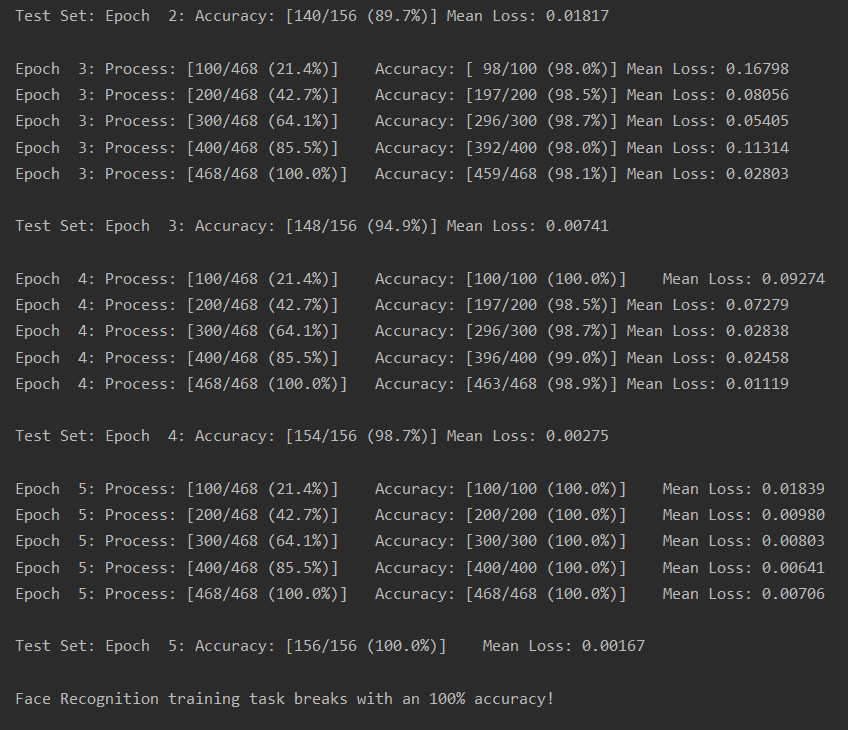
函数调用：

face\_clustering(1, 40)

# 四、实验结果

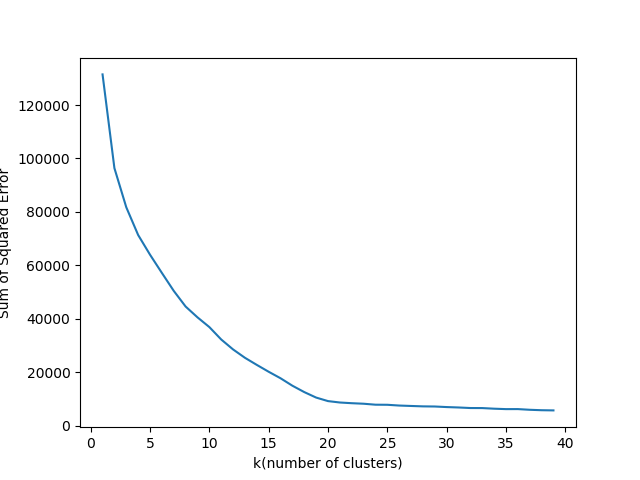
## 4.1 人脸识别

epoch设置为10个，最高准确率可以达到Accuracy: [156/156 (100.0%)]，最低精度Accuracy: [155/156 (99.4%)]。

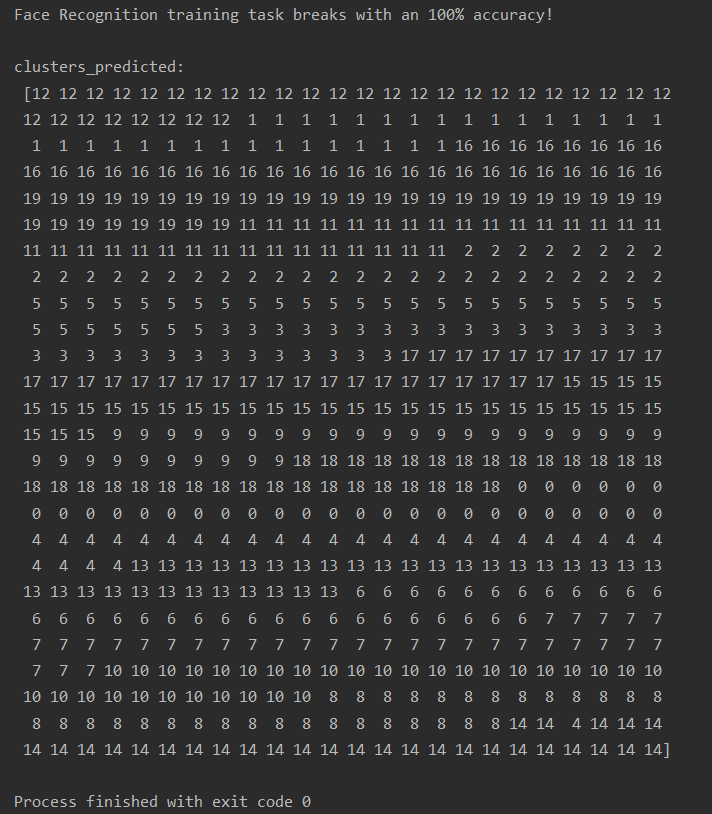


## 4.2 聚类

平方误差随着超参数k的增大而收敛，当k=20时，基本收敛，与数据集本身20个人的预设相同。



聚类预测结果：



# 五、实验心得

本次实验中，我在编程过程中了解了人脸识别的常用方法，并优化了神经网络的构建方式。此次实验的关键点主要在于对数据集特征的理解上，此外，在聚类时发现误差随着k值的增大逐渐收敛，但在等于实际数据集的聚类个数时，收敛速度减缓，甚至几乎不再减小。因此需要综合考虑计算复杂度和收敛情况，选取最合适的k值。