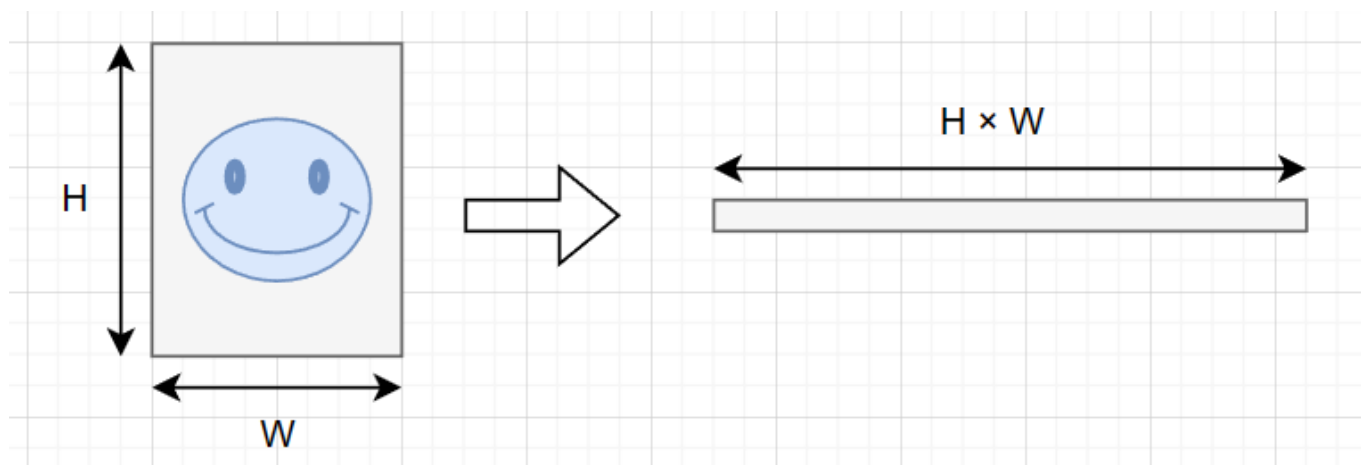
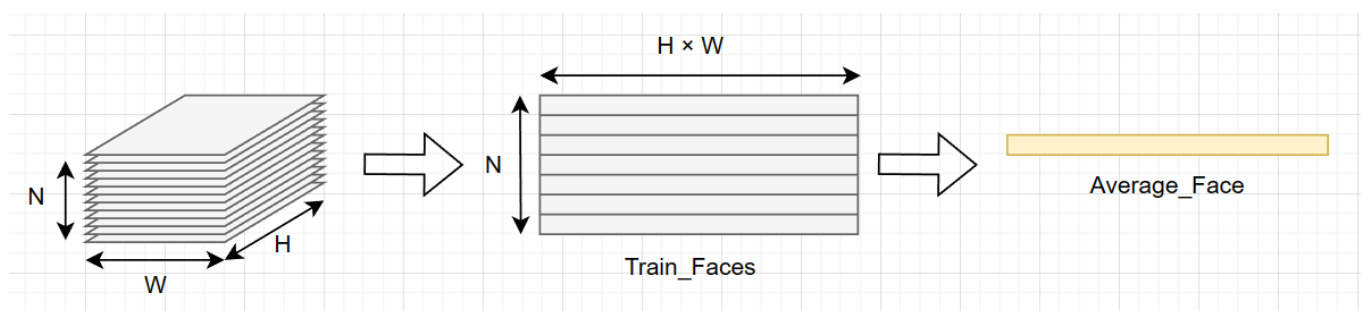


处理图像

一张 (H,W) 大小的图像可以按像素点展开为 $(1, HW)$ 大小数组。



我们将训练集中所有图像展开到一个 (N, HW) 大小的数组中;
然后求一个 $(1, HW)$ 大小的平均脸



参考代码:

```
# Suppose imgs is a (N, W, H) np array
img2D = imgs.reshape(imgs.shape[0], -1) # Flatten imgs to (N, WH)
img2D = img2D.astype("float32") # Set elem type to float

averageFace = np.mean(img2D, axis=0) # Calculate the average face
```

求特征脸 (Eigen Faces)

首先要声明, 上节中**每个像素点(列)不是特征, 每个脸(行)才是特征**.

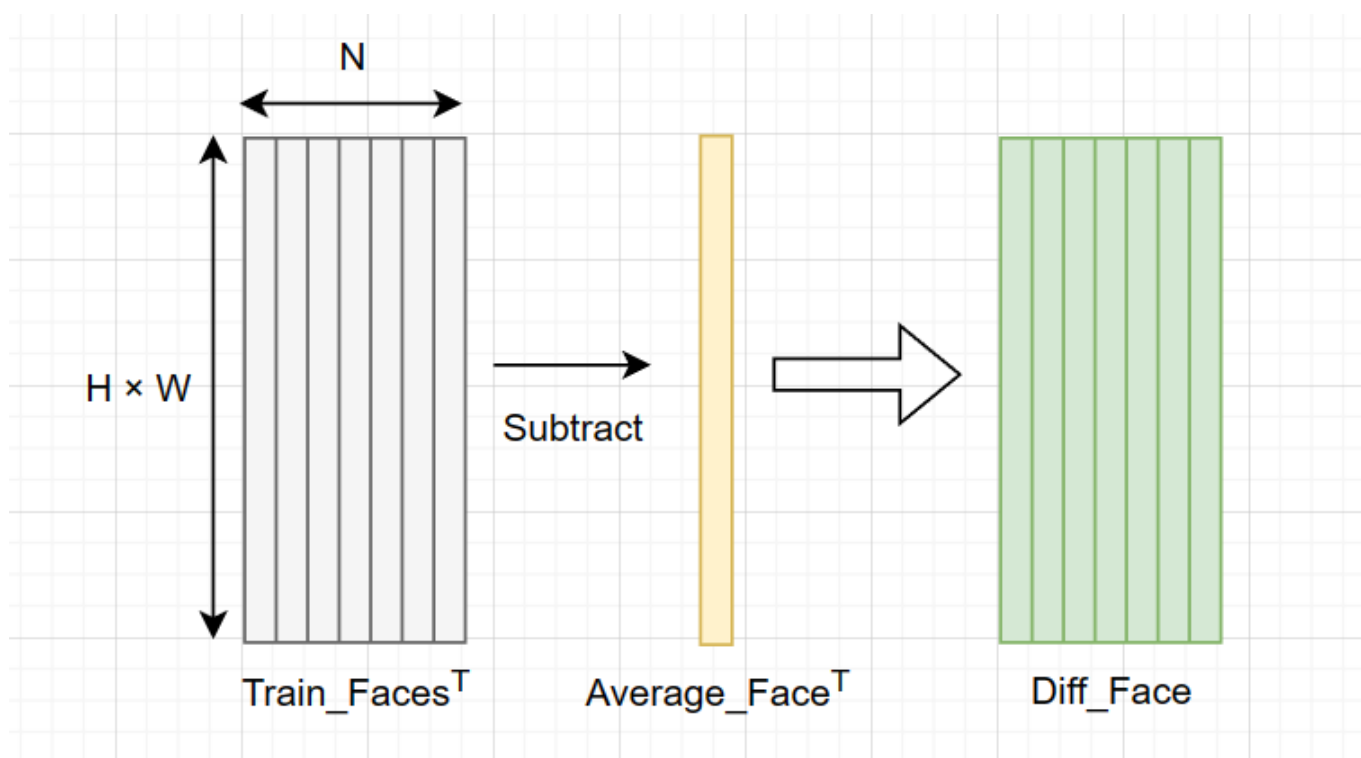
用不恰当的话说, 我们要做的是找到最有特征的一组脸, 把每个脸当成一个维度, 构成一个高维空间;
然后把训练集中的所有脸 (不同人不同角度的照片) 映射到这个空间中, 形成一些点.

当有一张新的脸 (测试集) 出现, 我们把新的脸映射到空间中, 寻找与前面训练集里距离最近的点.

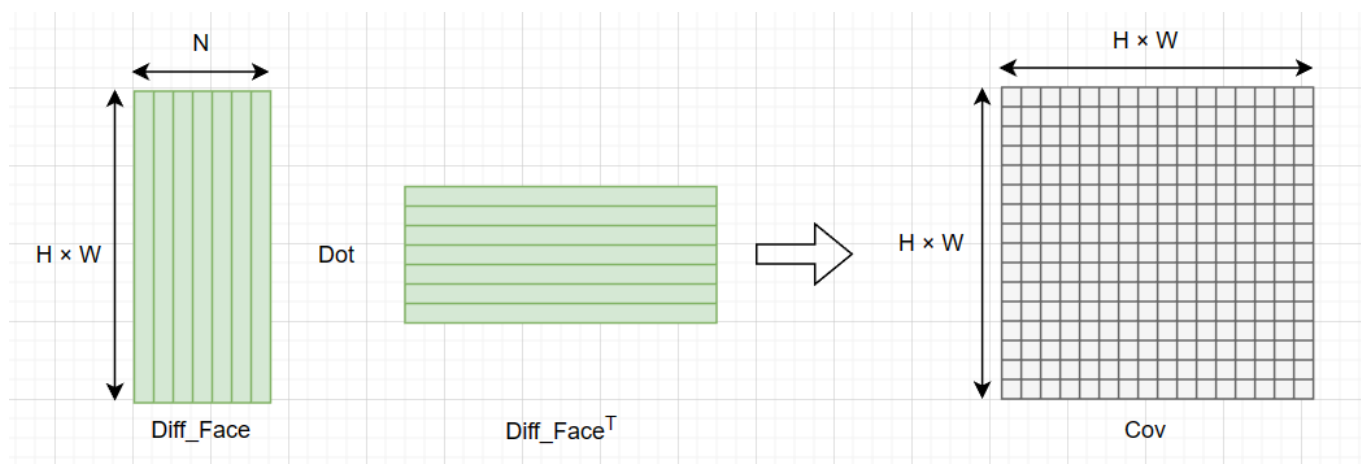
我们接下来就是要用 PCA (主成分分析) 求特征脸.

求每个特征 (每张脸) 的协方差矩阵

先将每张脸减去平均脸得到每张脸的差值 Diff_Face .



将 Diff_Face 与 Diff_Face 的转置点乘得到一个 (HW, HW) 大小的矩阵, 再对矩阵中每个元素乘以 $\frac{1}{HW - 1}$ 就得到了协方差矩阵 Cov .

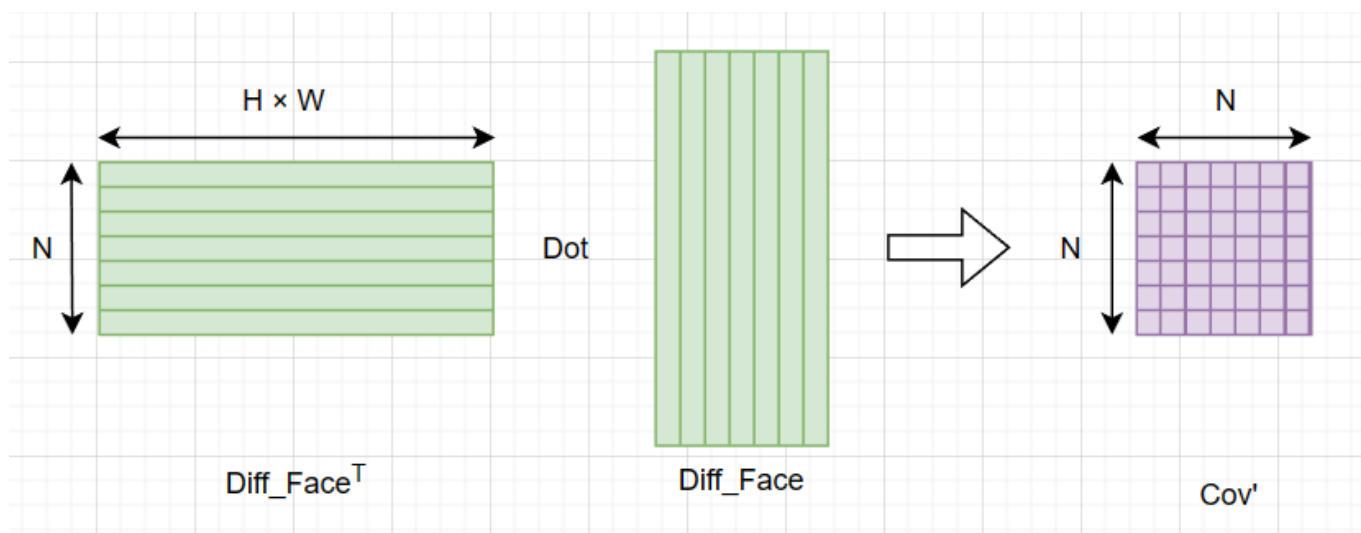


按照 PCA 的方法, 接下来我们需要求 Cov 的特征值和特征向量.

但是这里有个问题, 就是我们手里的这个矩阵的维数太高了 (维数是 HW , 即每张图片的像素点个数); 直接求特征向量运算时间不可接受.

如果用来 PCA 分析的图片数量 (N , 也就是特征数量) 超过了像素数 (HW , 也就是行数), 那么可以选择不采用下面的方法.

用 Diff_Face 的转置点乘自己, 再对矩阵中每个元素乘以 $\frac{1}{HW - 1}$ 就得到了一个 (N, N) 的协方差矩阵 Cov' .



可以证明, Cov' 的特征值与 Cov 的特征值相同;
 Cov 的特征向量矩阵等于 Cov' 点乘 Diff_Face^T .

这里不要太过纠结于应该横着来还是竖着来, 可以参考一下我下面给出的代码.

最终我们得到的每个向量应该有 HW 个维度.

参考代码:

```
# Suppose that images2D is a (N, HW) np array, representing N images and each has
# HW pixels.
# Suppose that averageFace is a (1, HW) np array.

# Calculate the covariance matrix of the result mat
covMat = np.cov(images2D) # (N, N) matrix

# Calculate the eigenvalues and eigenvectors of the covariance matrix
# eigenValues is a (N, N) mat, eigenVecs is an 1D array with N elems
eigenValues, eigenVecs = np.linalg.eig(covMat)

# Calculate the engine faces of origin images
# Now eigenVecs is a (N, WH) mat
eigenVecs = np.dot(eigenVecs, images2D - averageFace)
```

PCA

上面求出了一维向量 eigenValues , 也就是 Cov 的特征值. 我们选取最大的前 K 个特征值, 找到他们的对应特征向量, 将这些向量正交化, 选为特征脸.

正交化后的向量组成了一个 (K, WH) 大小的矩阵, 表示构成了 K 维特征脸空间.

参考代码:

```
# Get the indexes of the K largest eigenvalues
indexes = np.argsort(eigenValues)[::-1][:K]

# Get the K largest eigenvalues
eigenValues = eigenValues[indexes].copy()
# Get the K largest eigenvectors, i.e., the engine faces
eigenVecs = eigenVecs[indexes].copy() # (N, HW) mat

# Orthogonalize the engine faces
orth_eigenVecs, _ = np.linalg.qr(eigenVecs.T)
orth_eigenVecs = orth_eigenVecs.T # (K, HW) mat
```

训练样本

假设有 N 张图片作为训练集, 存放在一个 (N, HW) 大小的数组 `train_imgs` 中; 还有一个 $(N, 1)$ 大小的数组 `labels` 表示每张图片属于第几组(或者说第几个人).

`averageFace` 为 $(1, HW)$ 大小的平均脸, 求法上文已给出.

`orth_eigenVecs` 为 (K, HW) 大小的数组, 是特征脸, 求法上文已给出.

现在我们来训练样本.

```
# reshape face_imgs to [N, (H * W)]
train_imgs = train_imgs.reshape(train_imgs.shape[0], -1)
# reshape averageFace to [1, (H * W)]
averageFace = averageFace.reshape(1, -1)
# Subtract averageFace from face_imgs
train_imgs -= averageFace
# Calculate the engine values of the train faces
engineValues = np.dot(train_imgs, orth_eigenVecs.T) # [N, K]
labels = labels.reshape(-1, 1)
trainedData = np.concatenate((engineValues, labels), axis=1)
```

`trainedData` 是一个 $(N, K + 1)$ 大小的数组.

前 K 列存放了每张图片映射到特征脸空间后在每个 $k \in K$ 维度下的投影值. (或者我们可以说, 每行表示一个点, 前 K 列表示出了这个点在我们的 K 维特征脸空间中的坐标)

最后一列是传入的数据标签, 也就是说这个图像属于第一个人还是第十个人.

预测

我们有 N 个训练样本, 上面最后算出的 `trainedData` 标出每个点的坐标.

那我们接下来要做的显然是:

给出一个待预测图像, 用同样的方法算出其在特征脸空间中的坐标, 然后和 `trainedData` 中的 N 个坐标比较, 找到距离最近的那个点.

那个点的标签, 应该就是待预测图像的标签.

假设 `testFace` 为待预测图像, 大小为 $(1, HW)$.

`trainedData` 是一个 $(N, K + 1)$ 大小的数组, 表示 N 个已训练的图像在特征脸空间中的坐标, 同时每个点的标签存放在 $(1, N)$ 大小的数组 `labels` 中.

`averageFace` 为 $(1, HW)$ 大小的平均脸, 求法上文已给出.

`orth_eigenVecs` 为 (K, HW) 大小的数组, 是特征脸, 求法上文已给出.

```
# reshape testFace to [1 x (H * W)]
testFace = testFace.reshape(1, -1)
# Subtract averageFace from testFace
testFace -= averageFace
# Calculate the engine value of the test face
testEngineValue = np.dot(testFace, orth_engineVecs.T)
# Calculate the distance between the test face and the train faces
distances = np.linalg.norm(engineValues - testEngineValue, axis=1)
# Get the index of the train face with the minimum distance
index = np.argmin(distances)
# Get the label of the train face with the minimum distance
label = labels[index]
```

`label` 就是最终结果.