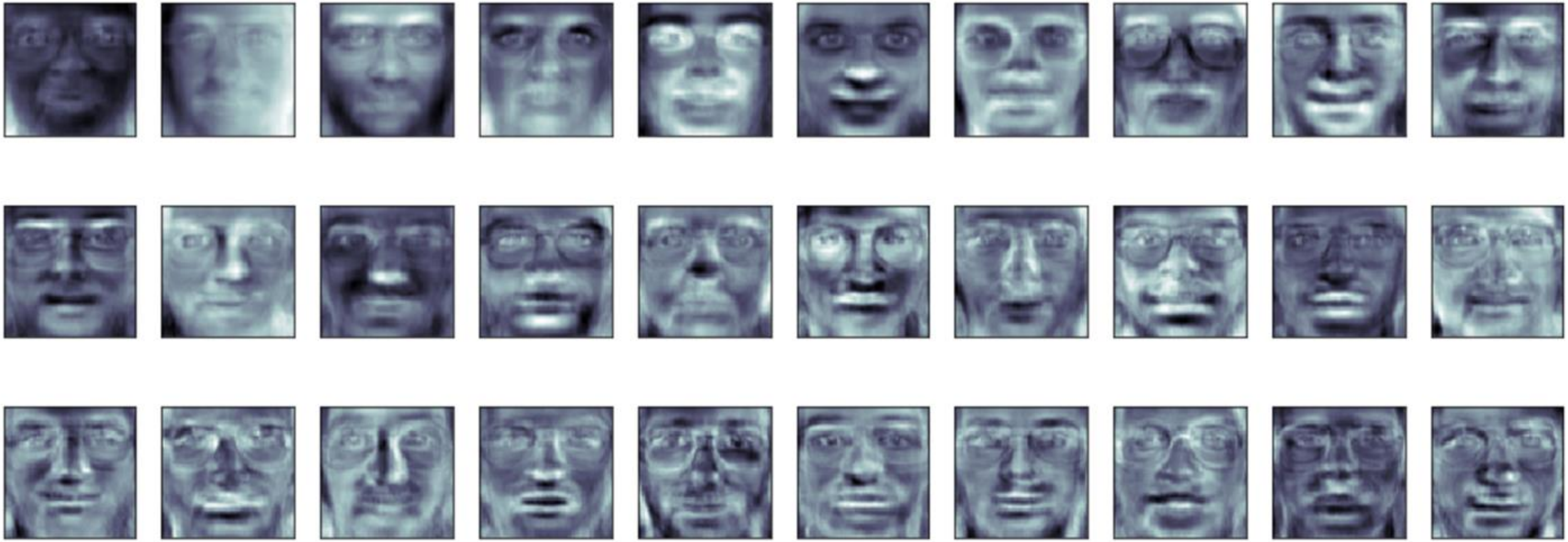
IMAGE CLASSIFICATION IN MACHINE LEARNING

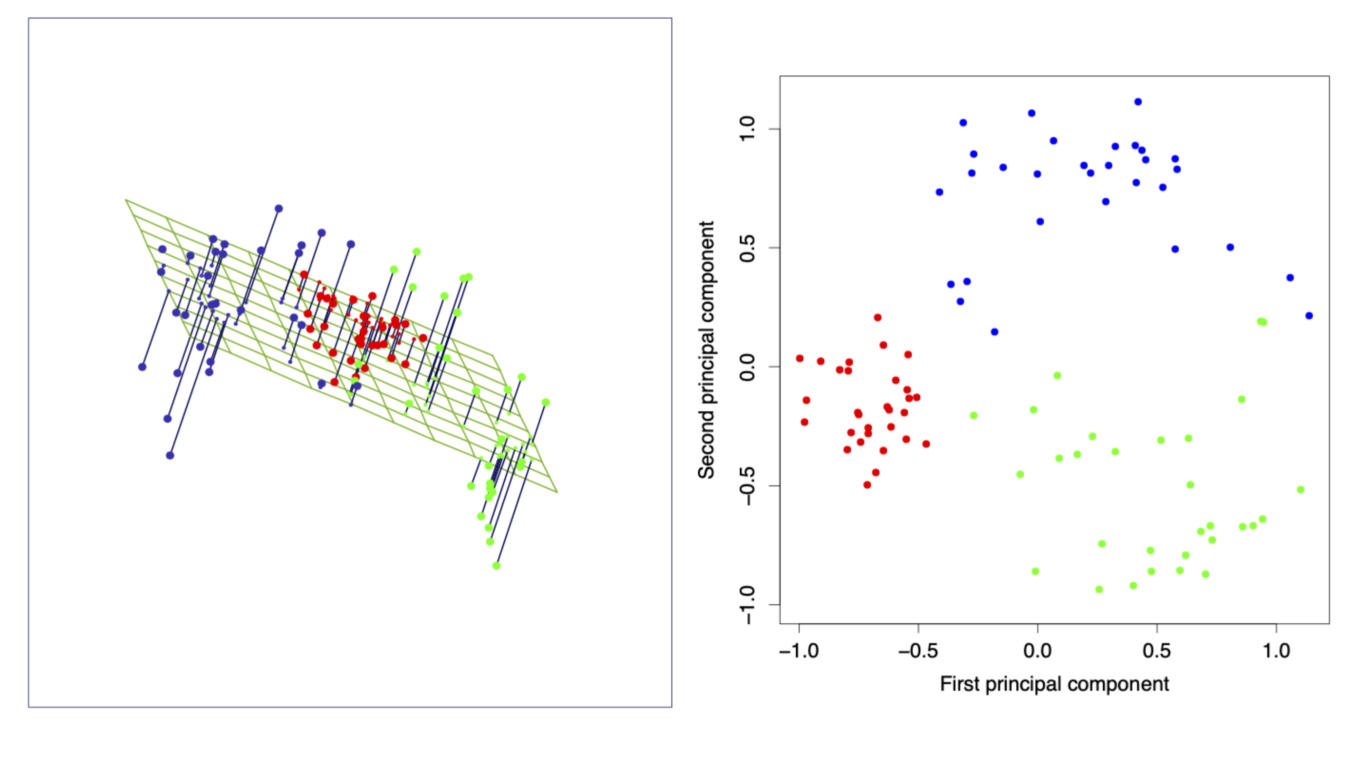
The applications of PCA in two cases

Ziyu Chen
Department of Mathematics, Applied Mathematics and Statistics
Dec 10^{th}, 2019
Capstone advisor: Danhong Song



Application of Principal Component Analysis (PCA) in Face Recognition

-the so called *eigenface*

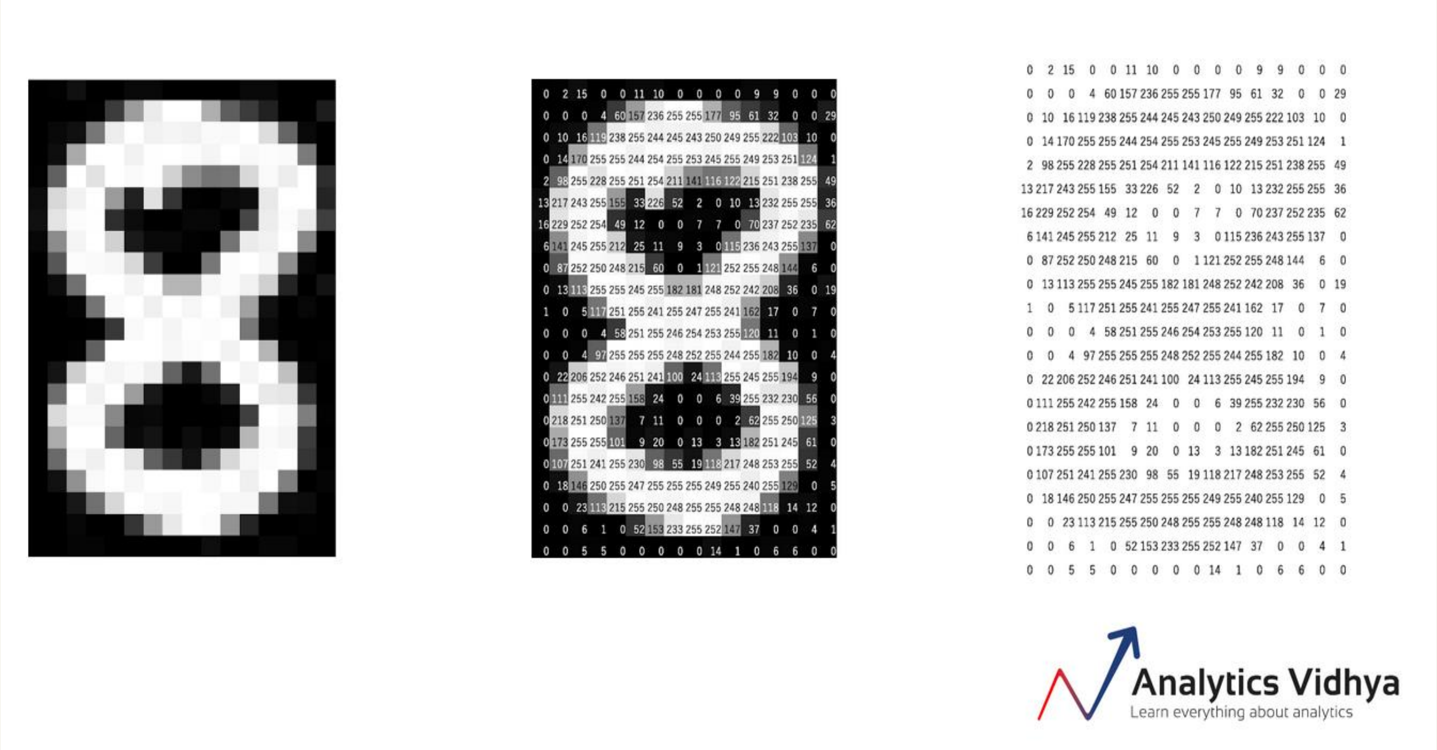


A Quick Review on PCA

- In general, PCA, as an **unsupervised** statistical machine learning method, is used to **reduce dimension** and to simplify the model. In the case of face recognition, we can accomplish the goal by PCA alone through finding *Eigenfaces*.

- According to *Wikipedia*, PCA is **a statistical procedure** that uses an **orthogonal transformation** to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of **linearly uncorrelated variables called principal components.**

Step 0: Some Background Introduction: How an image is stored in a computer?

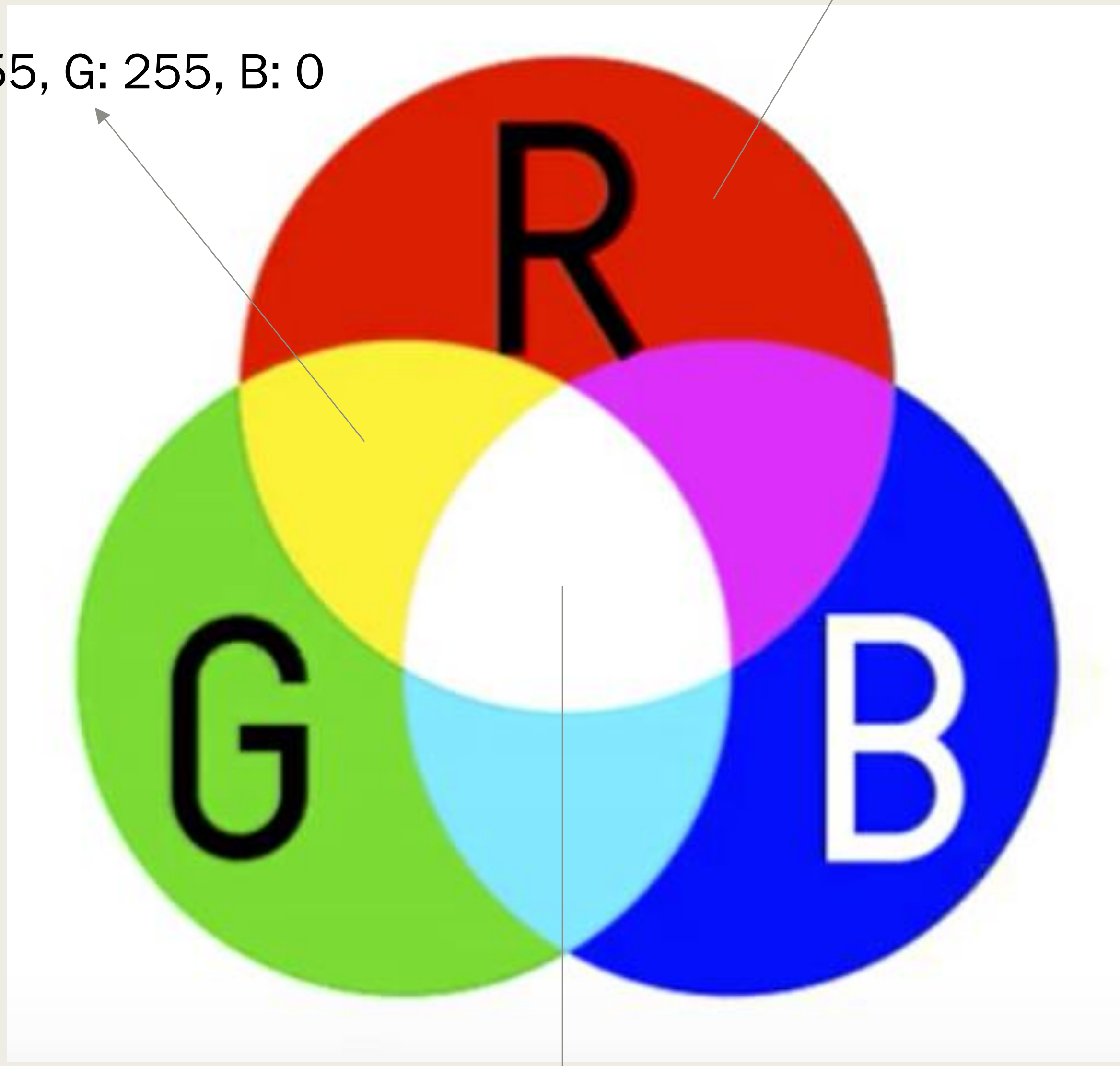


- Pixels

Each picture is composed by many small squares called "**pixels**", which is generally the color information. For a black and white image, the range for one pixel is usually **0-255**, where 0 stands for pure white and 255 stands for pure black. Grey is the middle stage. Thus, we call this grey scale.

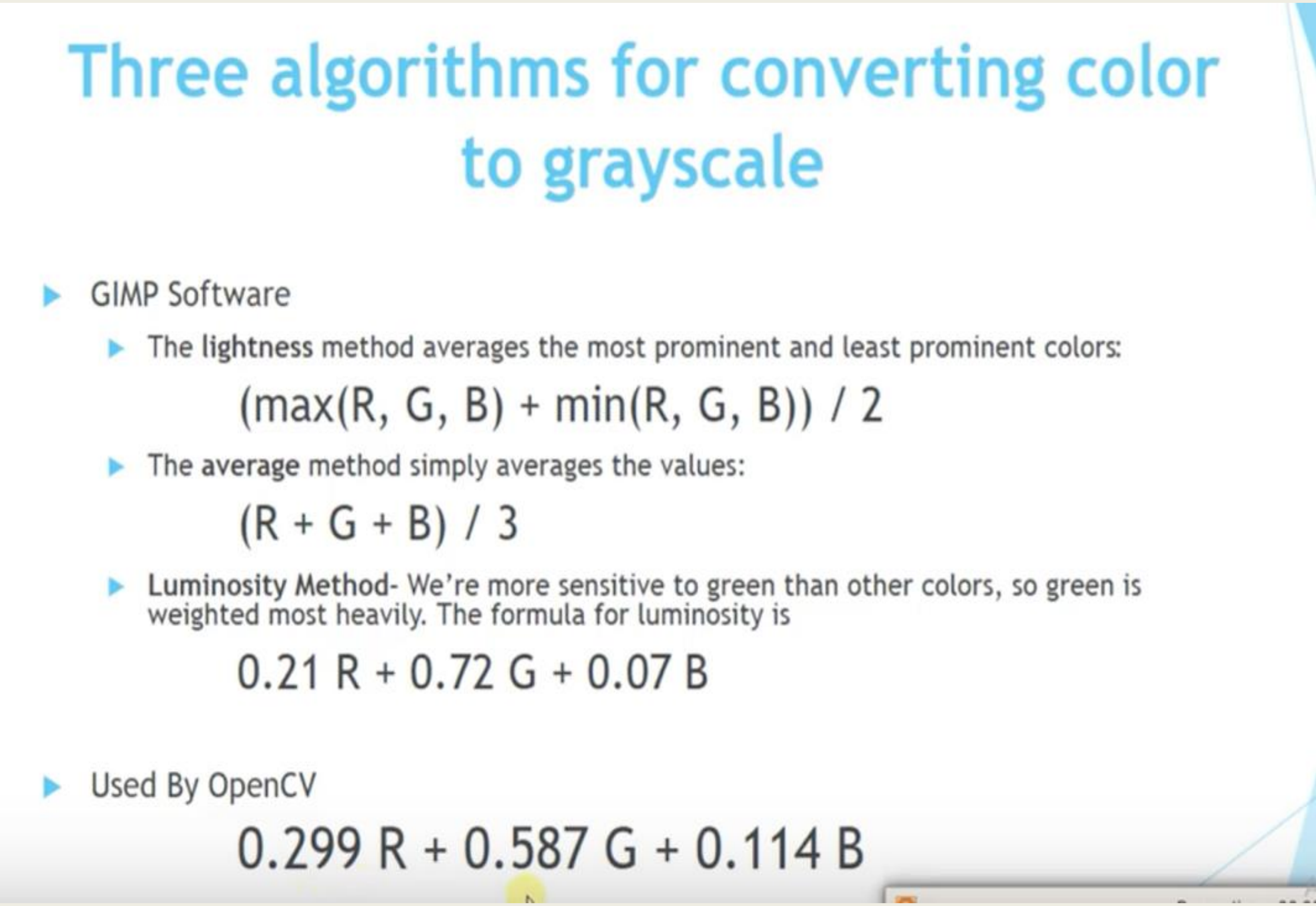
■ Color

All the different colors, considered by the computer, are composed from three base colors – red, green, and blue (**RGB**). Each of the three colors is stored in the computer as a number from 0 to 255. Now, we have three layers of color information for each pixel, if it is a colored image.



Change color to greyscale

Color information don't help identify the edges and more features. For many reasons, we will **convert color to greyscale** firstly, the information we need to deal with is only 1/3 compared to before. The simplest approach is (R+G+B) / 3.



https://www.youtube.com/watch?v=jDduLfZJMGY&list=PLgWKOWHJIDUM_cog-ujJgYoRCJ6LcAhtU

Step 1: Preprocessing of the image matrix and preparing the “single” image matrix

- Localized and scaled to a common size
- The algorithm to do so is beyond our discussion and should be considered more professionally by computer science majors.

Data Source: (known as **LFW**) Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. University of Massachusetts, Amherst, Technical Report 07-49*, October 2007.

Step 2: Basic data manipulation

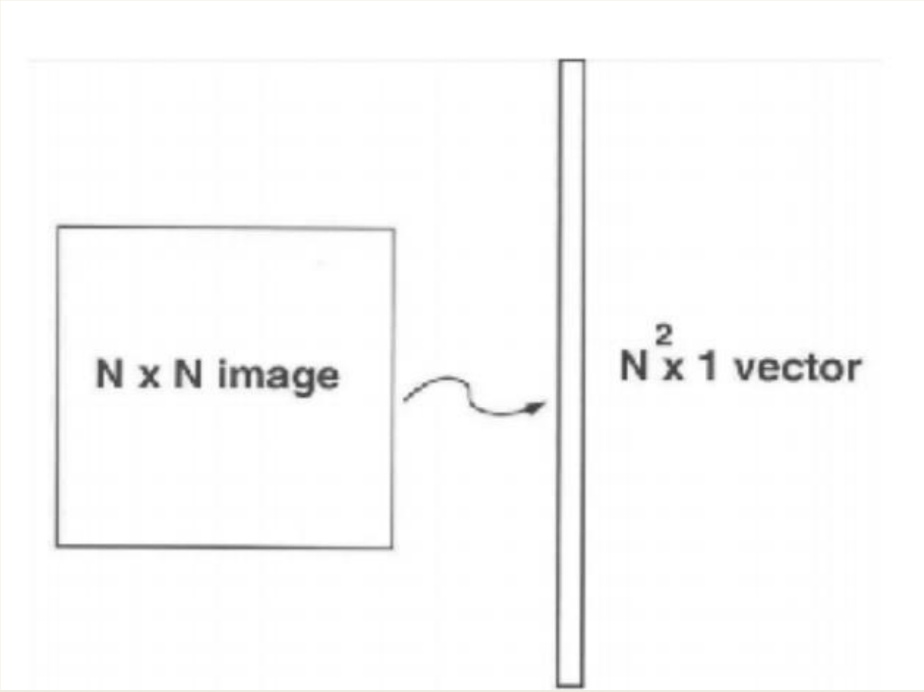
- Denote a **single image matrix** as $A = \begin{bmatrix} & \cdots & \\ & \vdots & \\ & \cdots & \end{bmatrix}^{N*N}$.

- Convert it to **a face vector $\boldsymbol{x}$** $= \begin{bmatrix} \\ \\ \\ \end{bmatrix}^{N^2*1}$

- Suppose we have T training images,

then we put all the face vectors together in **an image matrix**

$$\boldsymbol{X} = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_T] = \begin{bmatrix} & \cdots & \\ & \vdots & \\ & \cdots & \end{bmatrix}^{N^2*T}.$$



Step 3: Computing and subtracting the "mean" face

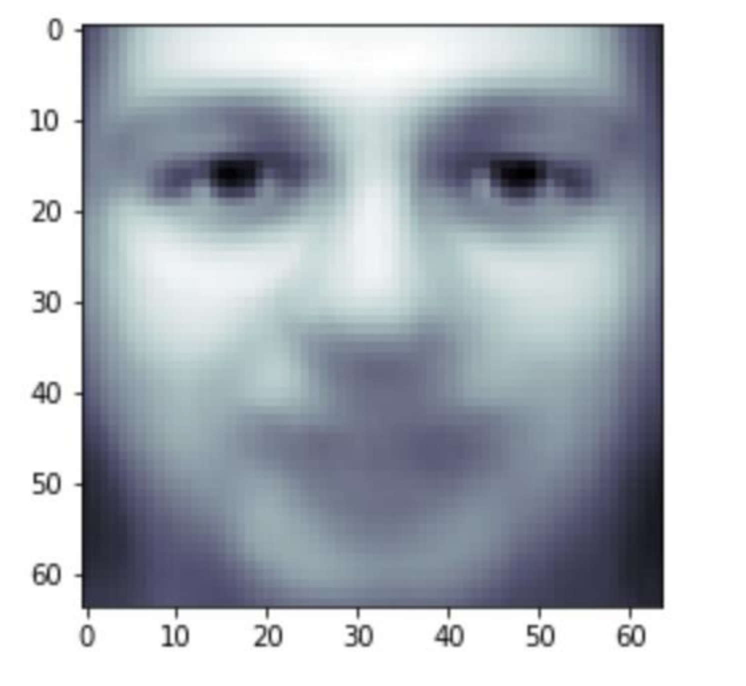
- Then we compute our mean face by simply find the "average" face:

$$\boldsymbol{\mu} = \frac{1}{T}\begin{bmatrix} \sum_{i=1}^{T} x_{i1} \\ \sum_{i=1}^{T} x_{i2} \\ \vdots \\ \sum_{i=1}^{T} x_{N^2 1} \end{bmatrix}^{N^2 * 1}$$

- We **subtract the mean face from image matrix** $\boldsymbol{X}$ to get a matrix that I call it a **normalized image matrix (NIM)**

$$\boldsymbol{M} = [\boldsymbol{m}_1 \quad \boldsymbol{m}_2 \quad \cdots \quad \boldsymbol{m}_T], where\ \boldsymbol{m}_i = \boldsymbol{x}_i - \boldsymbol{\mu}$$

*Note: We focus on the deviations of each face from the mean face.

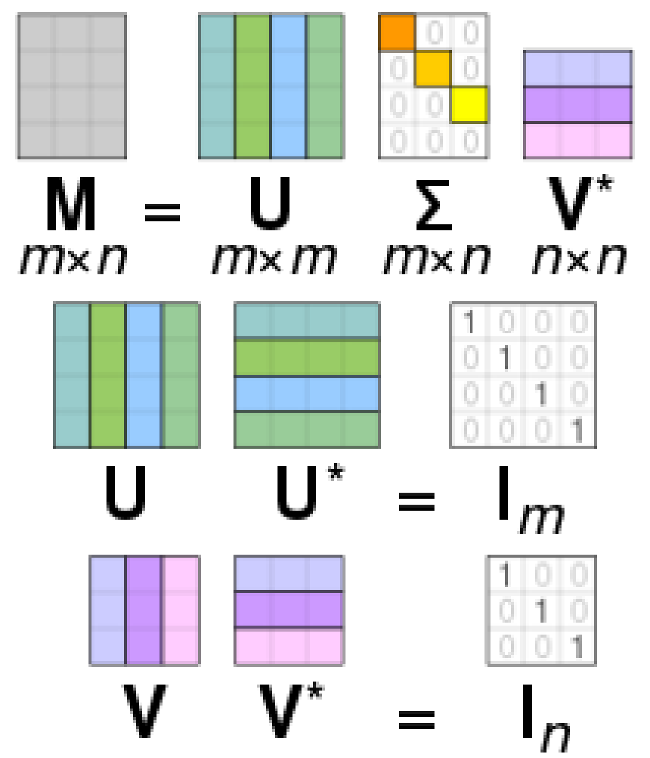


Step 4: Finding bases by finding eigenvectors of the covariance matrix of the NIM

- According to linear algebra, we know that we can find bases for our matrix M by finding eigenvectors of the covariance matrix of it.
- Each eigenvector is obtained in the direction of maximum variation.
- The covariance matrix of M is given as

$$C = MM^T = \frac{1}{T}\sum_{i=1}^{T} \boldsymbol{m}_i \boldsymbol{m}_i^{\ T} = [\quad]^{N^2 * N^2},$$

which is usually very large, i.e. when N=64, the dimension of C would be 4096*4096.



Step 5: Singular Value Decomposition of NIM

Usually, we would find the SVD of M, $\boldsymbol{M = U\Sigma V^*}$, *where U is the matrix composed by eigenvectors of covariance matrix of M, and U should have a dimension of $N^2 * N^2$.*

However, it is neither practical nor time-efficient to do such a huge computation.

Step 6: Practical approach

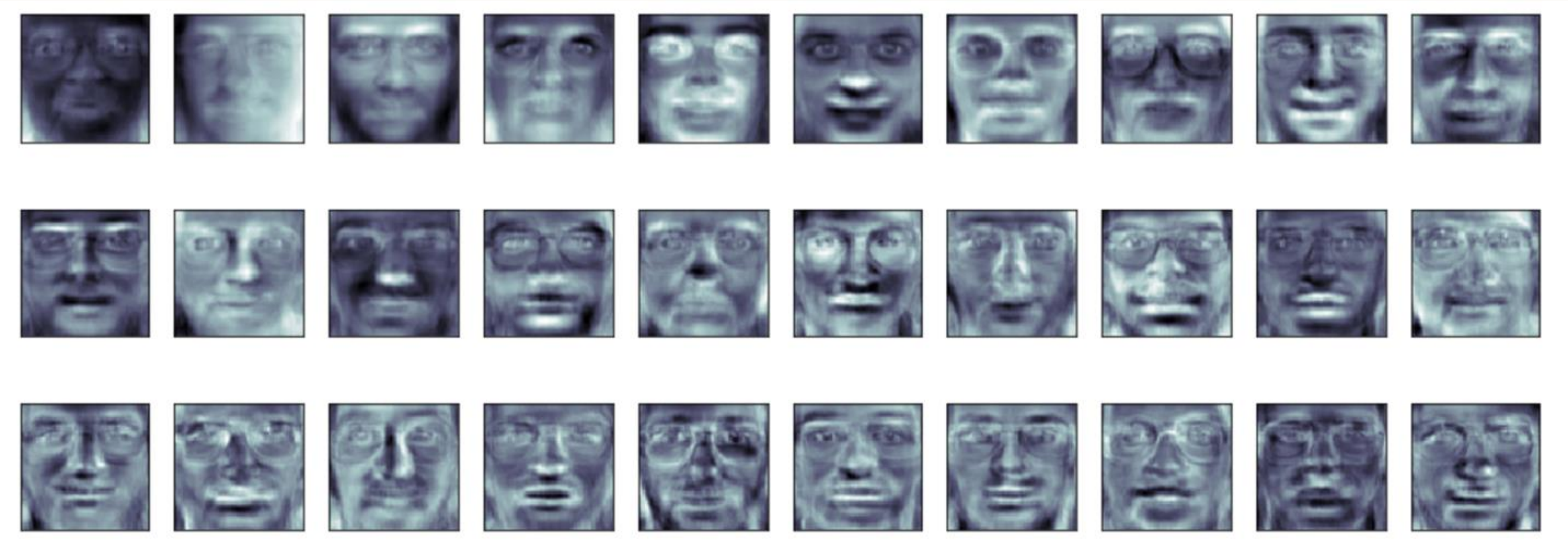
- In practice, we would consider **computing the eigenvectors of $M^T M$ instead of MM^T**, so we are dealing with a dimension of T*T instead of N^2* N^2. It is supported by **the theorem that MM^T and $M^T M$ have the same eigenvalues and their eigenvectors are related.**

- Since the first eigenvector is the direction of highest variance, to obtain 90% of the variation in the data, we only need p eigenvectors, where $p<T<<N^2$. We can achieve this goal when adding the corresponding eigenvalues until it reaches 90% of the sum of the eigenvalues.

- Then we choose the first p eigenvectors of C, putting these eigenvectors to U_p, we obtain the new bases matrix

$$U_p = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1p} \\ & & \vdots & \\ u_{N^2 1} & u_{N^2 2} & \cdots & u_{N^2 p} \end{bmatrix}^{N^2 * p} = [\boldsymbol{u}_1 \quad \boldsymbol{u}_2 \quad \cdots \quad \boldsymbol{u}_p].$$

- We call each $\boldsymbol{u}_j$ an eigenface, $1 \leq \boldsymbol{j} \leq \boldsymbol{p}$.



Step 7: Representing faces onto this basis = Representing faces in a combination of eigenfaces

- Then we can get our NIM projected on a new coordinated given by the basis. The projected NIM onto eigenspace is given as

$$M^* = U_p^T M = \begin{bmatrix} & \cdots & \\ & \vdots & \\ & \cdots & \end{bmatrix}^{p*T} = [\boldsymbol{w}_1 \quad \boldsymbol{w}_2 \quad \cdots \quad \boldsymbol{w}_T], \text{ where } \boldsymbol{w}_i = \begin{bmatrix} w_{1i} \\ w_{2i} \\ \vdots \\ w_{pi} \end{bmatrix}.$$

- $\boldsymbol{w}_i = U_p^T \cdot \boldsymbol{m}_i \; and \; then \; \boldsymbol{m}_i = U_p \boldsymbol{w}_i$
- Finally, we can express our original image matrix $x_i = \mu + \boldsymbol{m}_i = \boldsymbol{\mu} + w_{1i}\boldsymbol{u}_1 + w_{2i}\boldsymbol{u}_2 + \cdots + w_{pi}\boldsymbol{u}_p = \boldsymbol{\mu} + \sum_{j=1}^{p} w_{ji}\boldsymbol{u}_j \, , where \; w_{ji} = \boldsymbol{u}_j{}^T \boldsymbol{m}_i$.
- Your face = Mean Face + A linear Combination of Eigenfaces

Illustration(Only For Fun)

$\boldsymbol{x_i}$

Face of Ziyu Chen,
in the training data

$=$

$\boldsymbol{\mu}$

$- 0.3$

$\boldsymbol{u}_1$

$+ 0.07$

$\boldsymbol{u}_2$

$- 0.012$

$\boldsymbol{u}_3$

$+ \ldots \ldots$

Step 8: Recognizing an unknown face

Convert the input matrix A* to a face vector $\boldsymbol{x}^*$

Normalize the vector by subtracting the mean face

$$\boldsymbol{m} = \boldsymbol{x}^* - \boldsymbol{\mu}$$

Project the normalized face vector onto the eigenspace

$$\widehat{\boldsymbol{m}} = \sum_{i=1}^{p} \boldsymbol{w}_i \boldsymbol{u}_i \; (\boldsymbol{w}_i = \boldsymbol{u}_i{}^T \boldsymbol{m})$$

Obtain weight vector

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

Calculate distance between input weight vector and all the weight vectors in the training set

$$\boldsymbol{d} = \boldsymbol{min}_l \| \boldsymbol{W} - \boldsymbol{W}^l \|$$

An unknown test photo

Recognized as Ziyu

YES

If d < Threshold?

NO

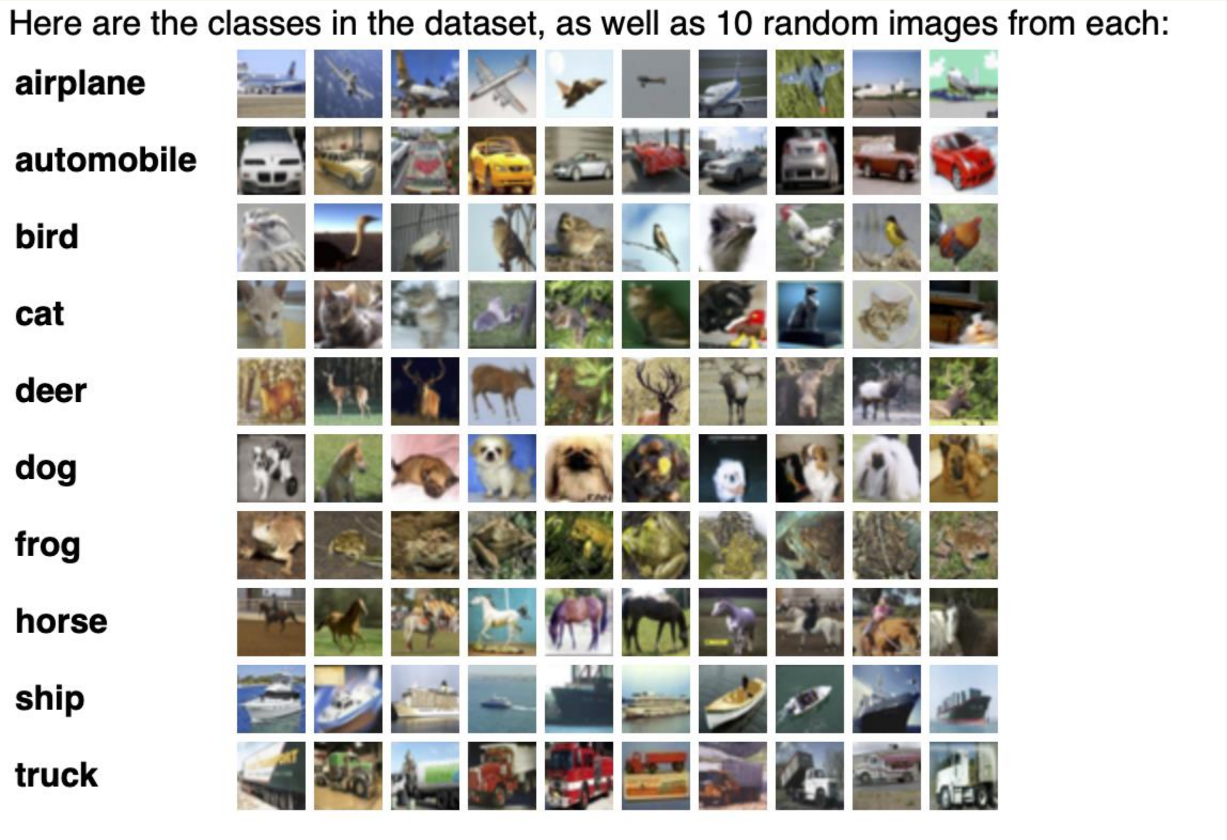
NO. Unknown person.

A face in training set.

Advanced models usually constructed with PCA

- In the **method of feature dimension reduction**, the Principal Component Analysis is the most classic and practical technology, especially in the image recognition field.
- *Linear Discriminant Analysis (LDA):* LDA is a well-known **supervised** algorithm that used in statistics, pattern recognition and machine learning to find a linear discriminant to best separate two or more classes. The method of using LDA in face recognition is usually referred as Fisher Face.

Method	Prediction Accuracy
SVM	0.84
PCA+SVM	0.94
LDA	0.99
PCA+LDA	0.97



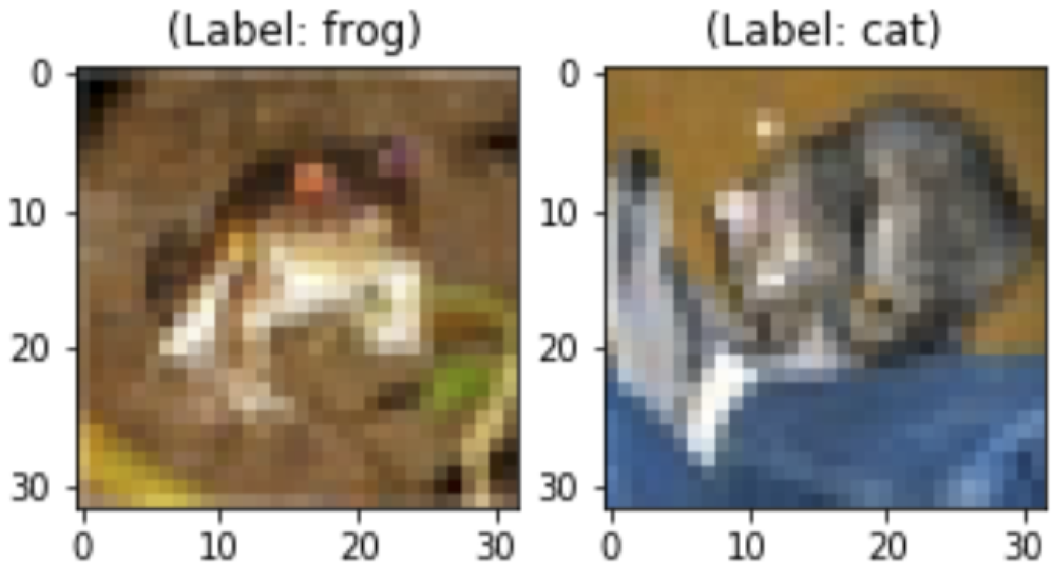
Application of Principal Component Analysis (PCA) in Object Classification

-with CIFAR 10 dataset

[9]:

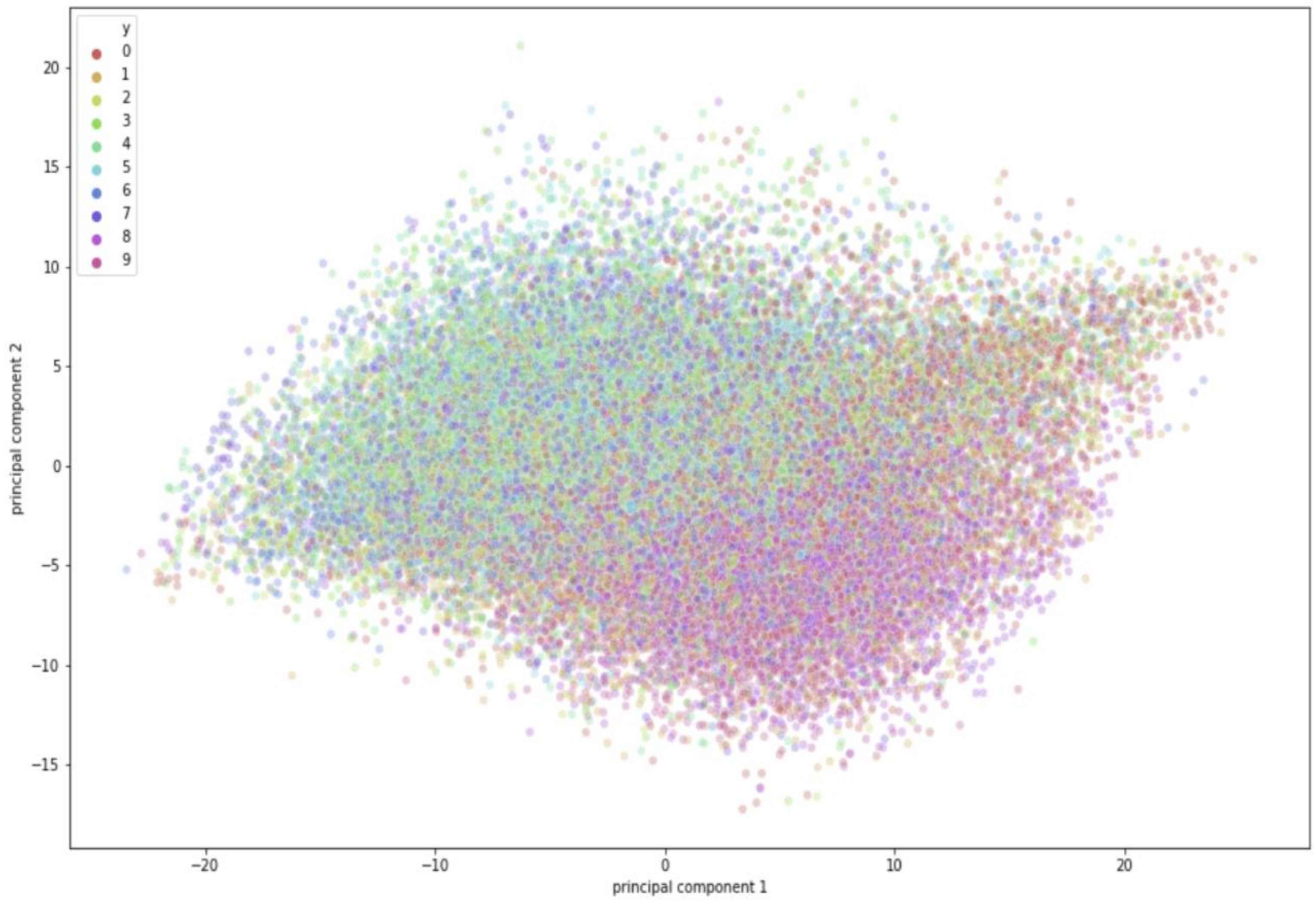
```
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
curr_img = np.reshape(x_train[0], (32,32,3))
plt.imshow(curr_img)
print(plt.title("(Label: " + str(label_dict[y_train[0][0]]) + ")"))

# Display the first image in testing data
plt.subplot(122)
curr_img = np.reshape(x_test[0],(32,32,3))
plt.imshow(curr_img)
print(plt.title("(Label: " + str(label_dict[y_test[0][0]]) + ")"))
```

```
Text(0.5, 1.0, '(Label: frog)')
Text(0.5, 1.0, '(Label: cat)')
```



Visualizing PCA by only two components

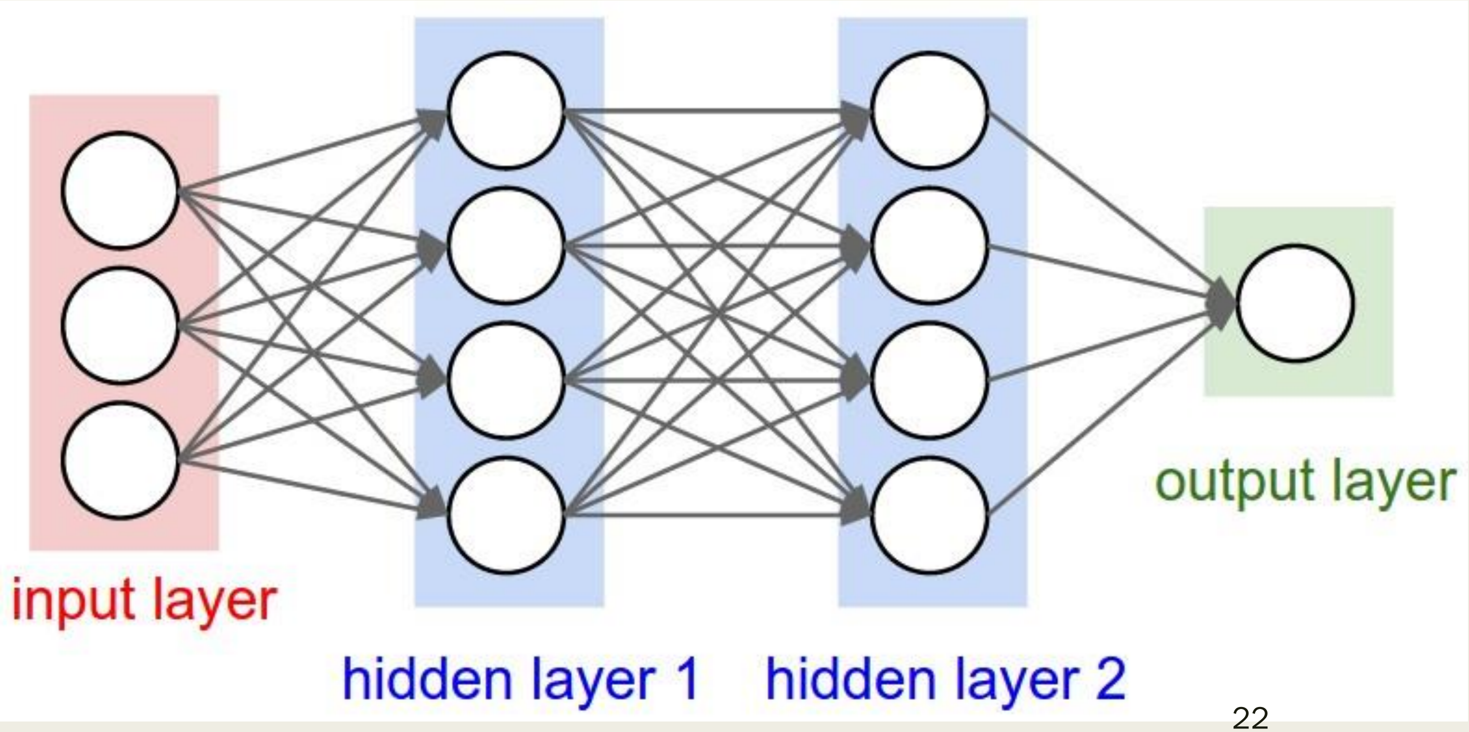
- However, lose a lot of variance, only 40.33% of variation explained.
- In real cases, we acquire a data explained rate = 90%, which requires 99 components instead of 3072.

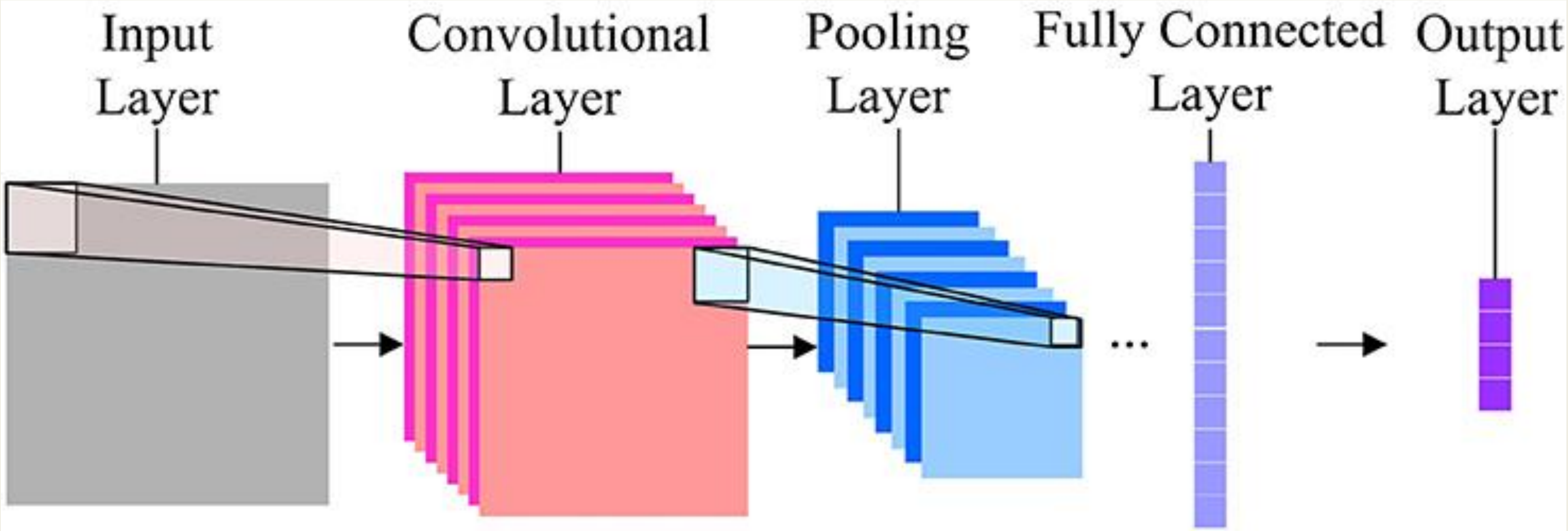


Using reduced dimension data for Convolutional Neural Network (**CNN**)

- Will not be introduced with details, since CNN is usually considered as a computer science approach, optimizing the algorithm by backward selection according to gradient.

- CNN itself is a technique of classifying images as a part of deep learning.




Input
Layer
Convolutional
Layer
Pooling
Layer
Fully Connected
Layer
Output
Layer
...

PCA to improve efficiency

- Using 99 components instead of 3072,

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/20
50000/50000 [==============================] - 19s 379us/step - loss: 1.6362 -
accuracy: 0.4200 - val_loss: 1.4266 - val_accuracy: 0.4995
Epoch 2/20
50000/50000 [==============================] - 16s 323us/step - loss: 1.3200 -
accuracy: 0.5322 - val_loss: 1.3290 - val_accuracy: 0.5343
Epoch 3/20
50000/50000 [==============================] - 15s 308us/step - loss: 1.1433 -
accuracy: 0.5928 - val_loss: 1.3343 - val_accuracy: 0.5363
```

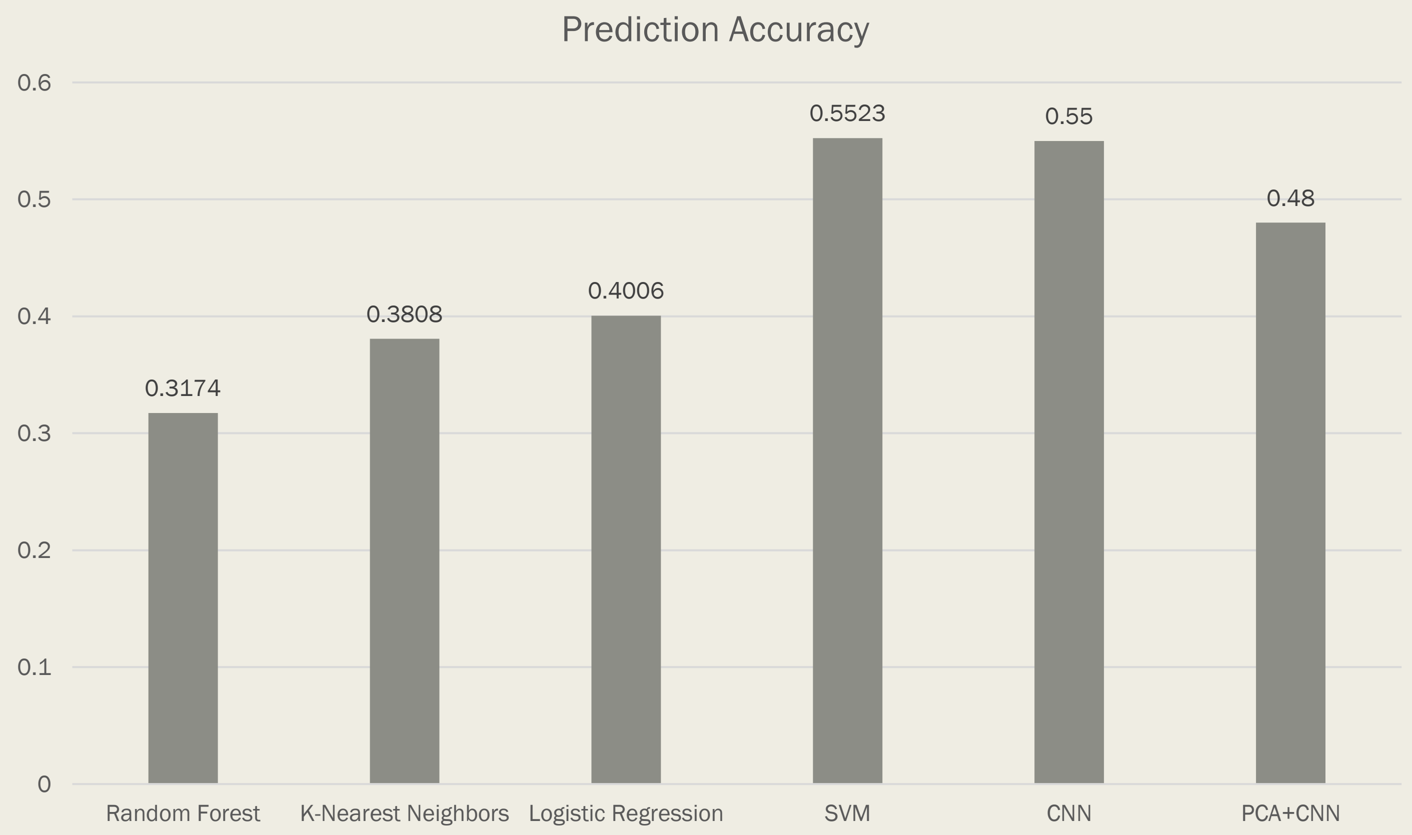
PCA+CNN

```
50000/50000 [==============================] - 37s 738us/step - loss: 2.1749 -
accuracy: 0.2377 - val_loss: 1.9239 - val_accuracy: 0.2994
Epoch 2/20
50000/50000 [==============================] - 45s 906us/step - loss: 1.8457 -
accuracy: 0.3350 - val_loss: 1.8519 - val_accuracy: 0.3459
Epoch 3/20
50000/50000 [==============================] - 39s 778us/step - loss: 1.7539 -
accuracy: 0.3697 - val_loss: 1.7713 - val_accuracy: 0.3614
```

CNN

Comparing the accuracy given by classical statistical approach

Method	Prediction Accuracy
Random Forest	0.3174
K-Nearest Neighbors	0.3808
Logistic Regression	0.4006
SVM	0.5523
CNN	0.5500
PCA+CNN	0.48



Future study:

- Combination of PCA, LDA, SVM, KNN, CNN, more advanced neural network methods would change both **model accuracy** and **model efficiency**.
- How to put these methods together, in which order should we use this method, and many other questions are waiting to be answered. Latest research is in this field.

Thank you for listening.