Image Compression and Recognition

Wanfang Zhang

April 12, 2023

Image Compression–PCA

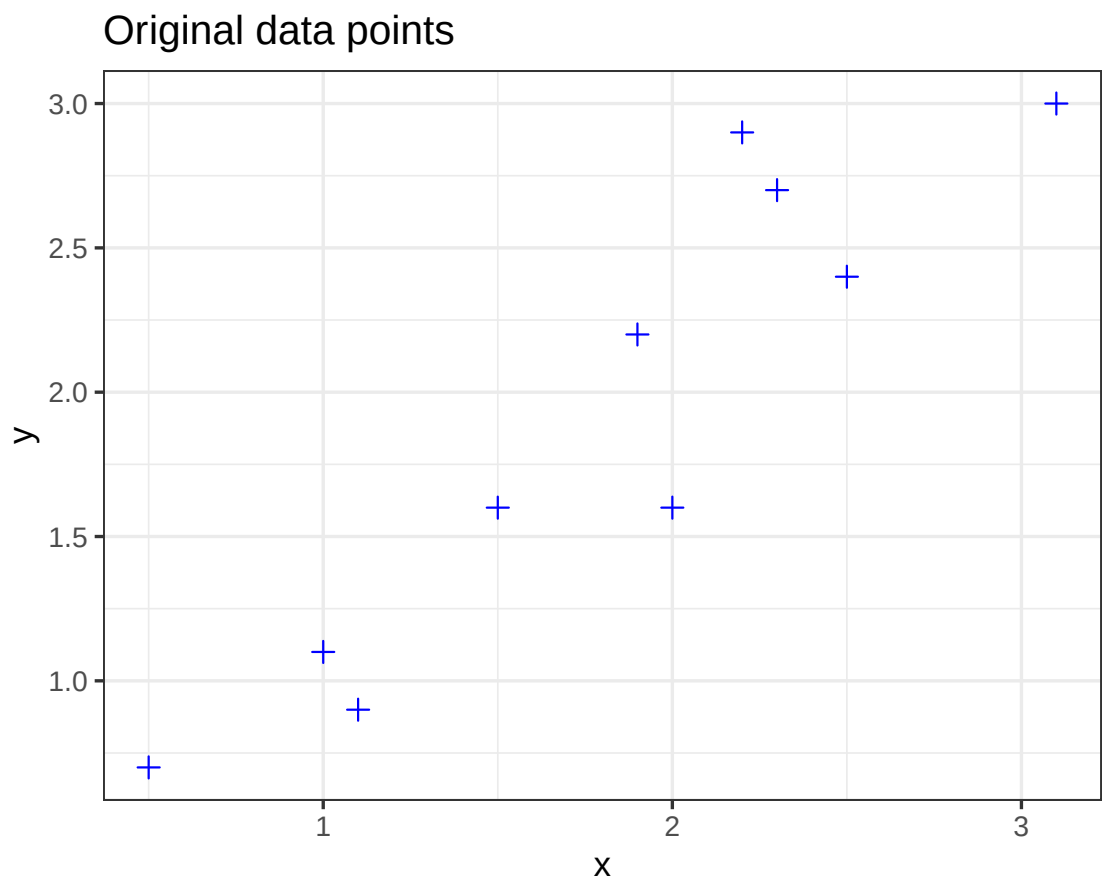
- Principal Component Analysis
  - PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component),
  - the second greatest variance on the second coordinate, and so on.

- In other words, we convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components

PCA–Example Data

Let's use a simple example to see how this algorithm works.
The following table only contains two dimensions: x and y. We are going to use a reduced data set to simplify the calculations and provide plots to show what the PCA analysis is doing at each step.

	x	y
1	2.50	2.40
2	0.50	0.70
3	2.20	2.90
4	1.90	2.20
5	3.10	3.00
6	2.30	2.70
7	2.00	1.60
8	1.00	1.10
9	1.50	1.60
10	1.10	0.90



PCA–Subtract the Mean

- First, we have to subtract the mean from each of the data dimensions. This procedure make sure our data has mean zero.
- Calculate $(x - \bar{x})$ and $(y - \bar{y})$, where $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ and $\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$

	x	y	$(x - \bar{x})$	$(y - \bar{y})$
1	2.50	2.40	0.69	0.49
2	0.50	0.70	-1.31	-1.21
3	2.20	2.90	0.39	0.99
4	1.90	2.20	0.09	0.29
5	3.10	3.00	1.29	1.09
6	2.30	2.70	0.49	0.79
7	2.00	1.60	0.19	-0.31
8	1.00	1.10	-0.81	-0.81
9	1.50	1.60	-0.31	-0.31
10	1.10	0.90	-0.71	-1.01

PCA–Calculate the Covariance Matrix

- The aim of the covariance matrix calculation is usually to see if there is any relationship between the dimensions. The covariance matrix for this 2 dimensional data set can be expressed as

$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(y,x) & \text{cov}(y,y) \end{pmatrix}$$

where

$$\text{cov}(x,y) = \text{cov}(y,x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

$$\text{cov}(x,x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})}{n-1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} = \text{var}(x)$$

$$\text{cov}(y,y) = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(y_i - \bar{y})}{n-1} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1} = \text{var}(y)$$

PCA–Calculate the Covariance Matrix

	x	y	$(x-\bar{x})$	$(y-\bar{y})$	$(x-\bar{x})$ $(y-\bar{y})$	$(x-\bar{x})^2$	$(y-\bar{y})^2$
1	2.50	2.40	0.69	0.49	0.34	0.48	0.24
2	0.50	0.70	-1.31	-1.21	1.59	1.72	1.46
3	2.20	2.90	0.39	0.99	0.39	0.15	0.98
4	1.90	2.20	0.09	0.29	0.03	0.01	0.08
5	3.10	3.00	1.29	1.09	1.41	1.66	1.19
6	2.30	2.70	0.49	0.79	0.39	0.24	0.62
7	2.00	1.60	0.19	-0.31	-0.06	0.04	0.10
8	1.00	1.10	-0.81	-0.81	0.66	0.66	0.66
9	1.50	1.60	-0.31	-0.31	0.10	0.10	0.10
10	1.10	0.90	-0.71	-1.01	0.72	0.50	1.02

With these values we can obtain easily the covariance matrix:

$$C = \begin{pmatrix} 0.6165 & 0.6154 \\ 0.6154 & 0.7165 \end{pmatrix}$$

PCA–Eigenvectors and Eigenvalues

- The eigenvalues explain the variance of the data along the new feature axes, the eigenvector with the highest eigenvalue is the first principal component.

$$\begin{pmatrix} a & c \\ c & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$$

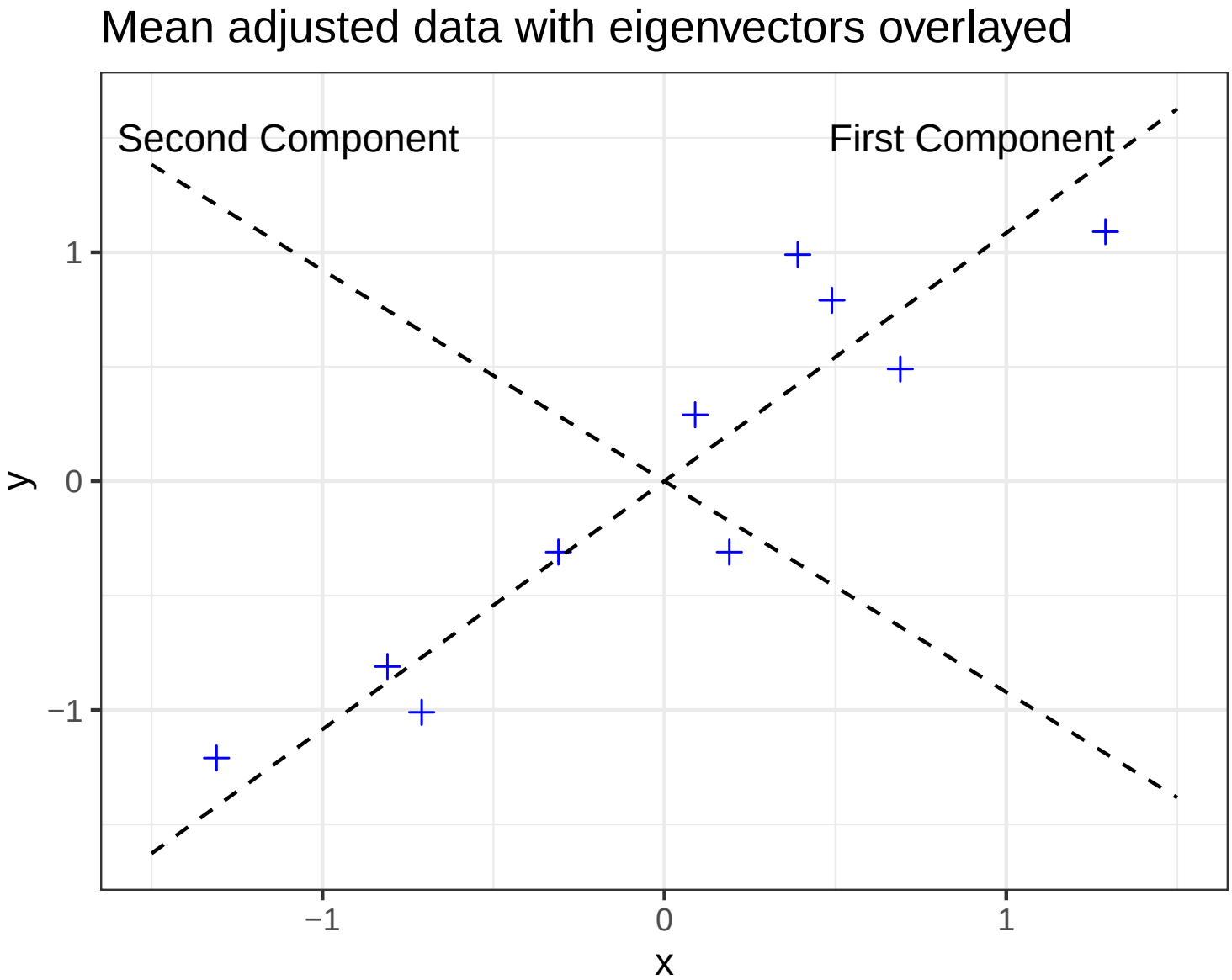
where $a = 0.6165$, $b = 0.7165$ and $c = 0.6154$.
With R function: eigen() and eigen()$vector:

$$\text{eigenvalues} = \begin{pmatrix} 1.2840277 \\ 0.0490834 \end{pmatrix}$$

$$\text{eigenvectors} = \begin{pmatrix} 0.6778734 & -0.7351787 \\ 0.7351787 & 0.6778734 \end{pmatrix}$$

PCA–Eigenvectors and Eigenvalues

The number of eigenvalues and eigenvectors that exits is equal to the number of dimensions the data set has.



PCA–Choosing Components

- Once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest.

- In order to decide, let's analyze the percentage of variances explained by each principal component.

```
> summary(pca)
Importance of components:
                            PC1      PC2
Standard deviation      1.1331 0.22155
Proportion of Variance 0.9632 0.03682
Cumulative Proportion  0.9632 1.00000
```

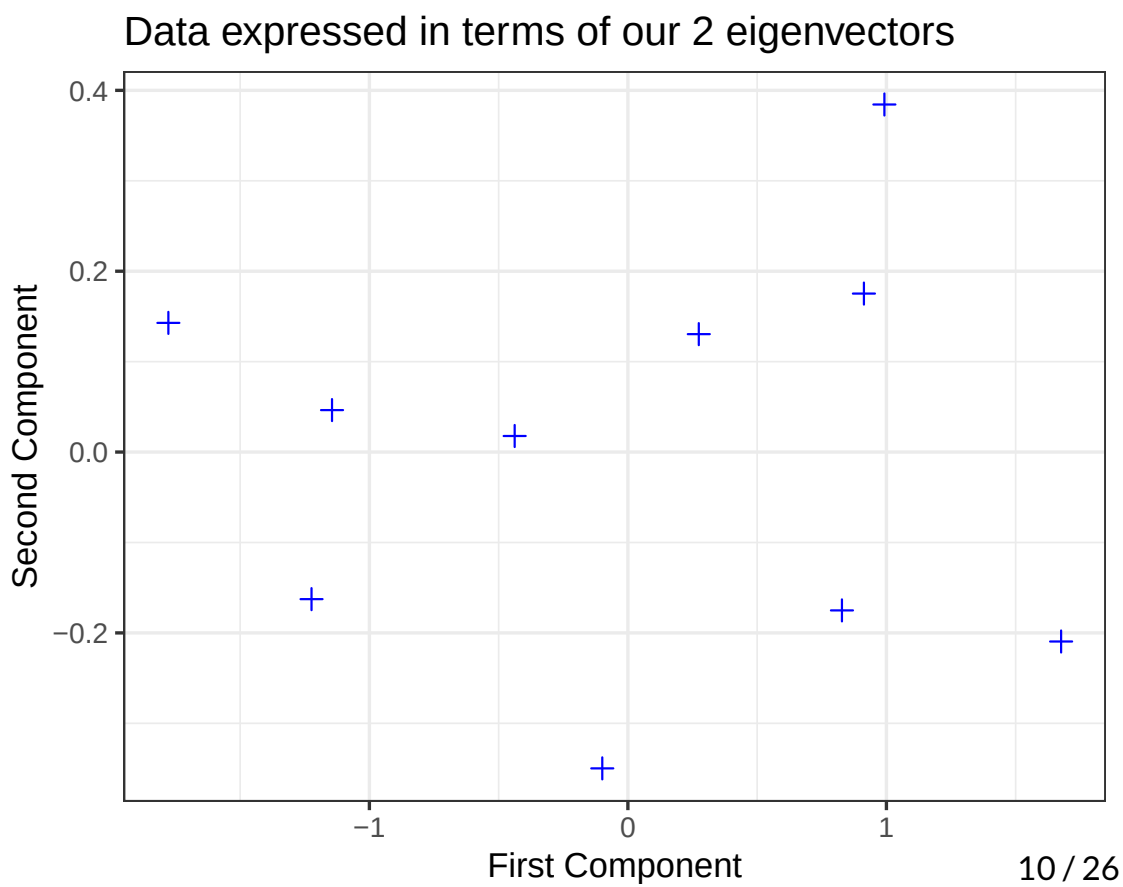
- PC1 explains 96% of the total variance, so we can discard the second component.

PCA–Deriving the New Dataset

Once we have chosen the components that we wish to keep in our data, we simply take the transpose of the vector and multiply it on the left of the original data set, transposed.

```
dataNew=as.data.frame(t(t(eigen$vectors) %*%
                        rbind(x-mean(x), y-mean(y))))
```

	New x	New y
1	0.83	-0.18
2	-1.78	0.14
3	0.99	0.38
4	0.27	0.13
5	1.68	-0.21
6	0.91	0.18
7	-0.10	-0.35
8	-1.14	0.05
9	-0.44	0.02
10	-1.22	-0.16



PCA–Getting the Original Data Back

If we have reduced the number of eigenvectors in the final transformation, then the retrieved data has lost some information.

```
dataOld=as.data.frame(t(t(pca$x[, 1] %*%
                    t(pca$rotation[, 1])) + pca$center))
```

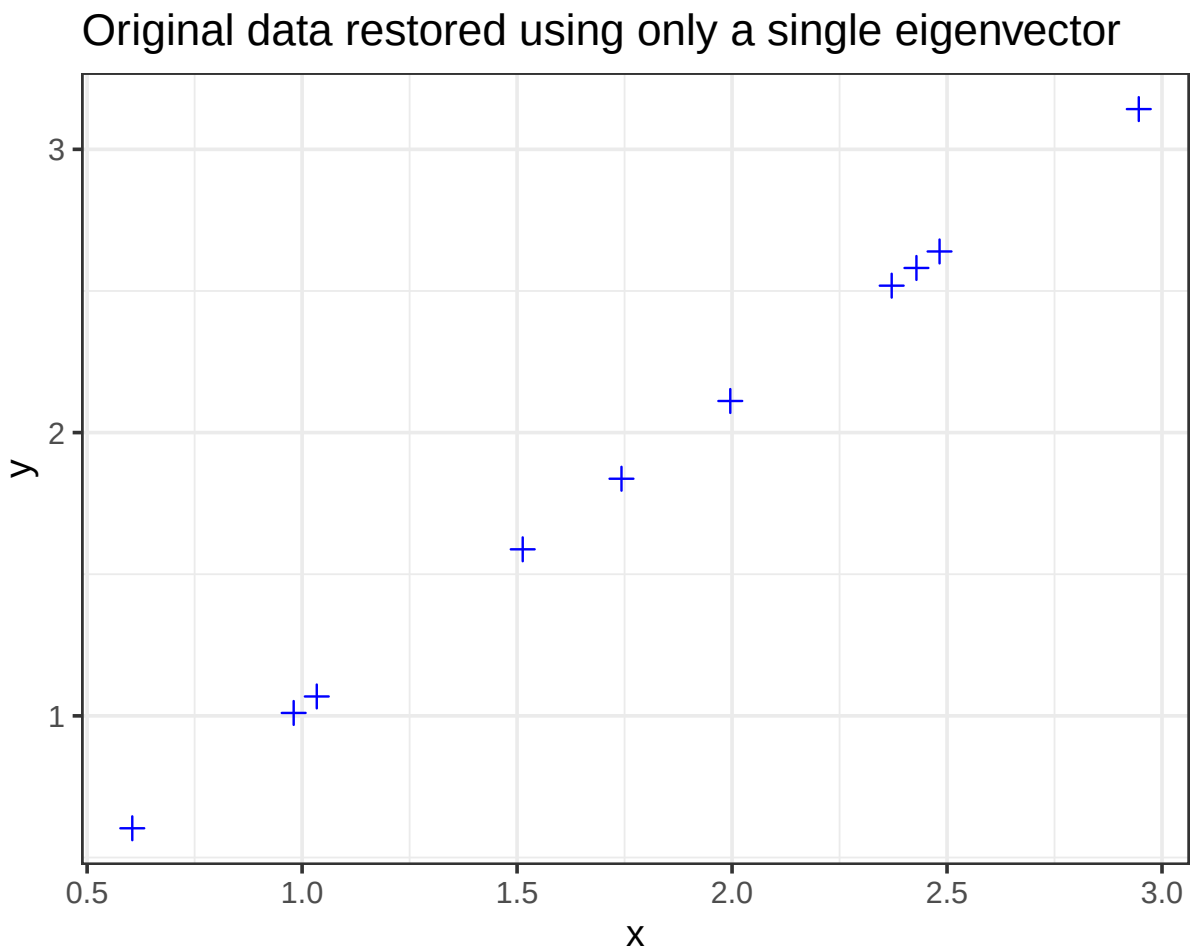


Image Compression

- In image data, each cell in the array corresponds to a specific pixel in the image, and the value in that cell represents the intensity of the corresponding color channel at that pixel.
  For example:

```
>image=readJPEG("C:/meme.jpg")
>img_gray=apply(image, c(1,2),
function(x) weighted.mean(x,c(0.299, 0.587, 0.114)))
>str(image) #color image
 num [1:1290, 1:1080, 1:3]
 0.788 0.788 0.788 0.78 0.773 ...

>str(img_gray) #grayscale image
 num [1:1290, 1:1080]
 0.799 0.799 0.8 0.793 0.785 ...
```

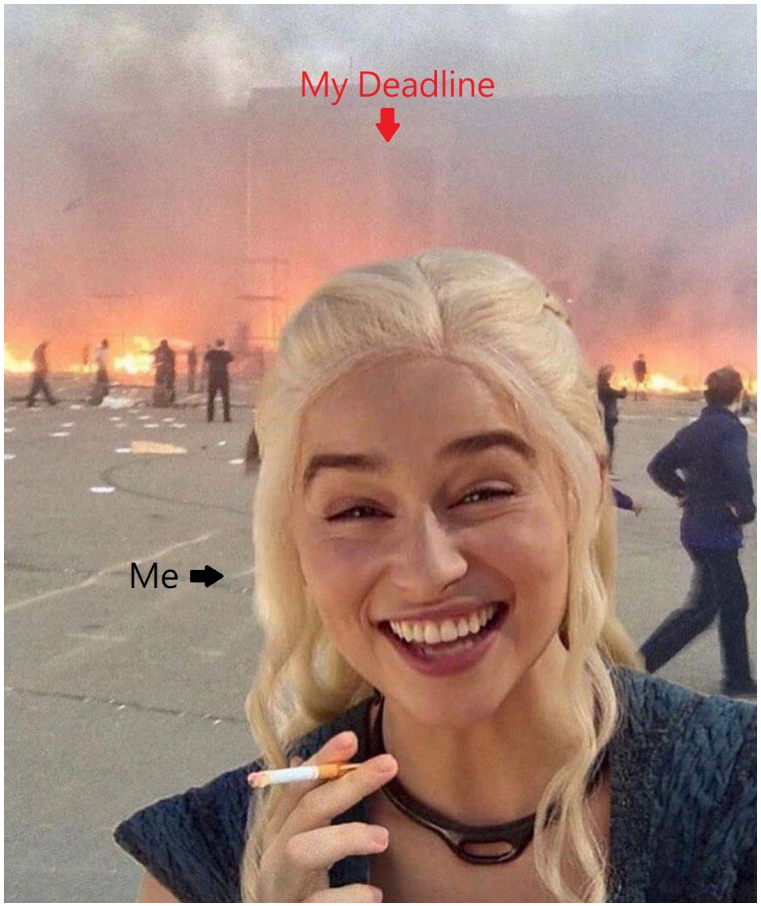
- Image data are more complicated than our scatter plot example.

Image Compression–Color Image

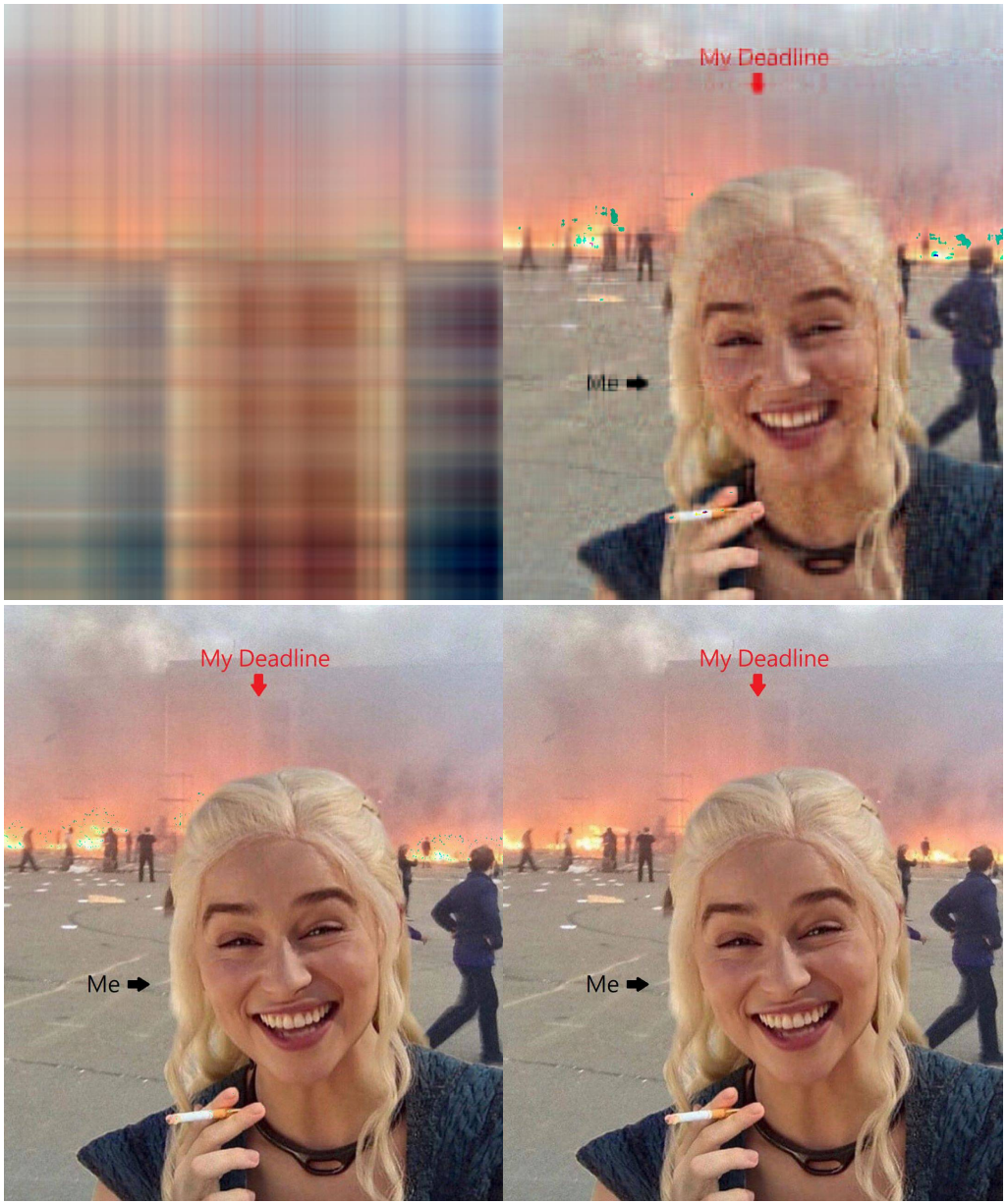
- We are going to break down each color scheme into three data frames.
- Then we can apply the PCA separately for each color scheme.
- we reconstruct the image four times: using 2, 30, 200 and 600 principal components.

```
pcar=prcomp(image[,,1], center=FALSE)
pcag=prcomp(image[,,2], center=FALSE)
pcab=prcomp(image[,,3], center=FALSE)
pcaimage=list(pcar, pcag, pcab)
pcnum=c(2, 30, 200, 600)
for(i in pcnum){
  pca.img=sapply(pcaimage, function(j){
  compressed.img=j$x[,1:i] %*% t(j$rotation[,1:i])
  }, simplify='array')
  writeJPEG(pca.img,
paste("C:reconstruction with",round(i,0),"PCs.jpg"))
}
```

Image Compression–Color Image

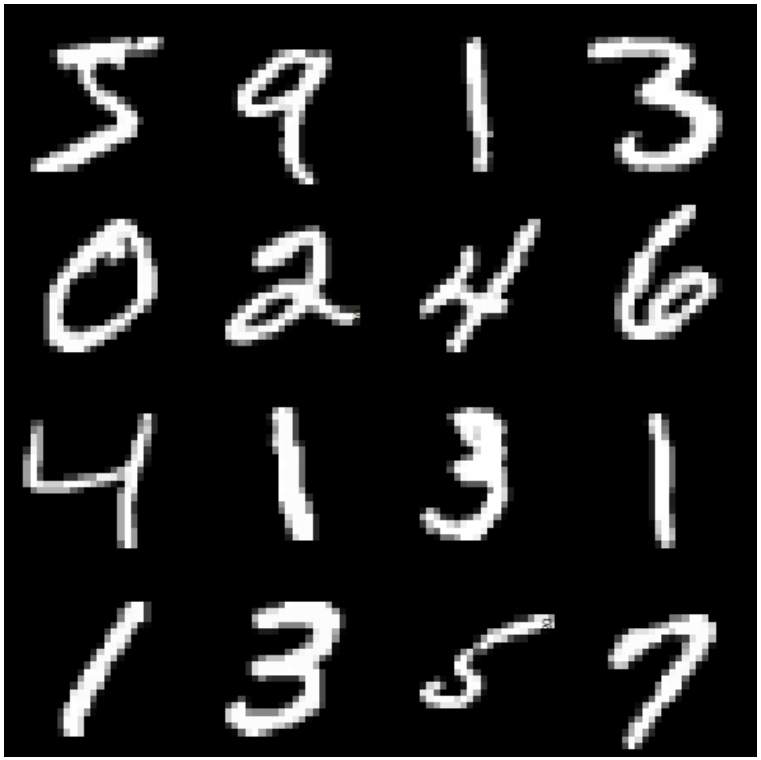


Original Image

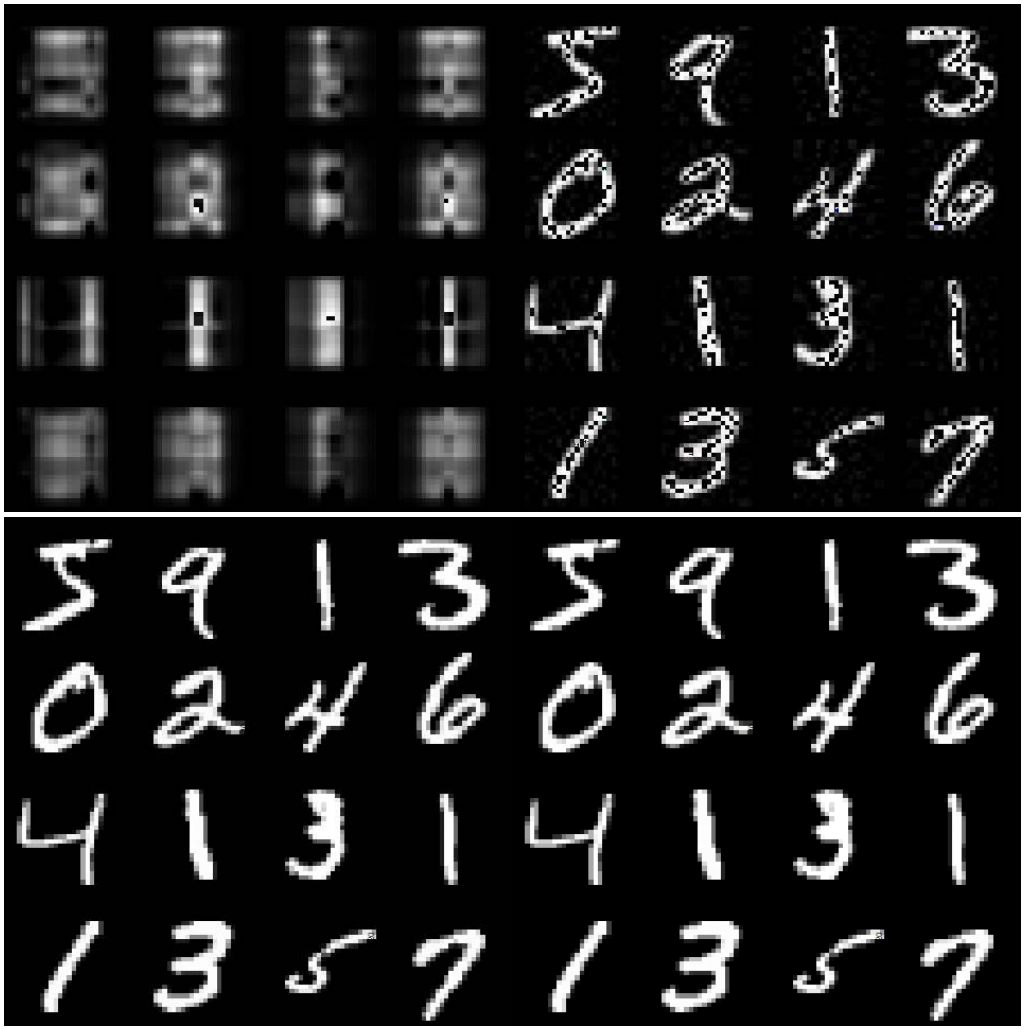


Reconstructed Image

Image Compression–Grayscale Image



Original Image



Reconstructed Image

Image Recognition–MNIST Data

- MNIST dataset is built into the Keras library with contains 60,000 training images and 10,000 test images.
- Apply neural network model for handwriting digit recognition.

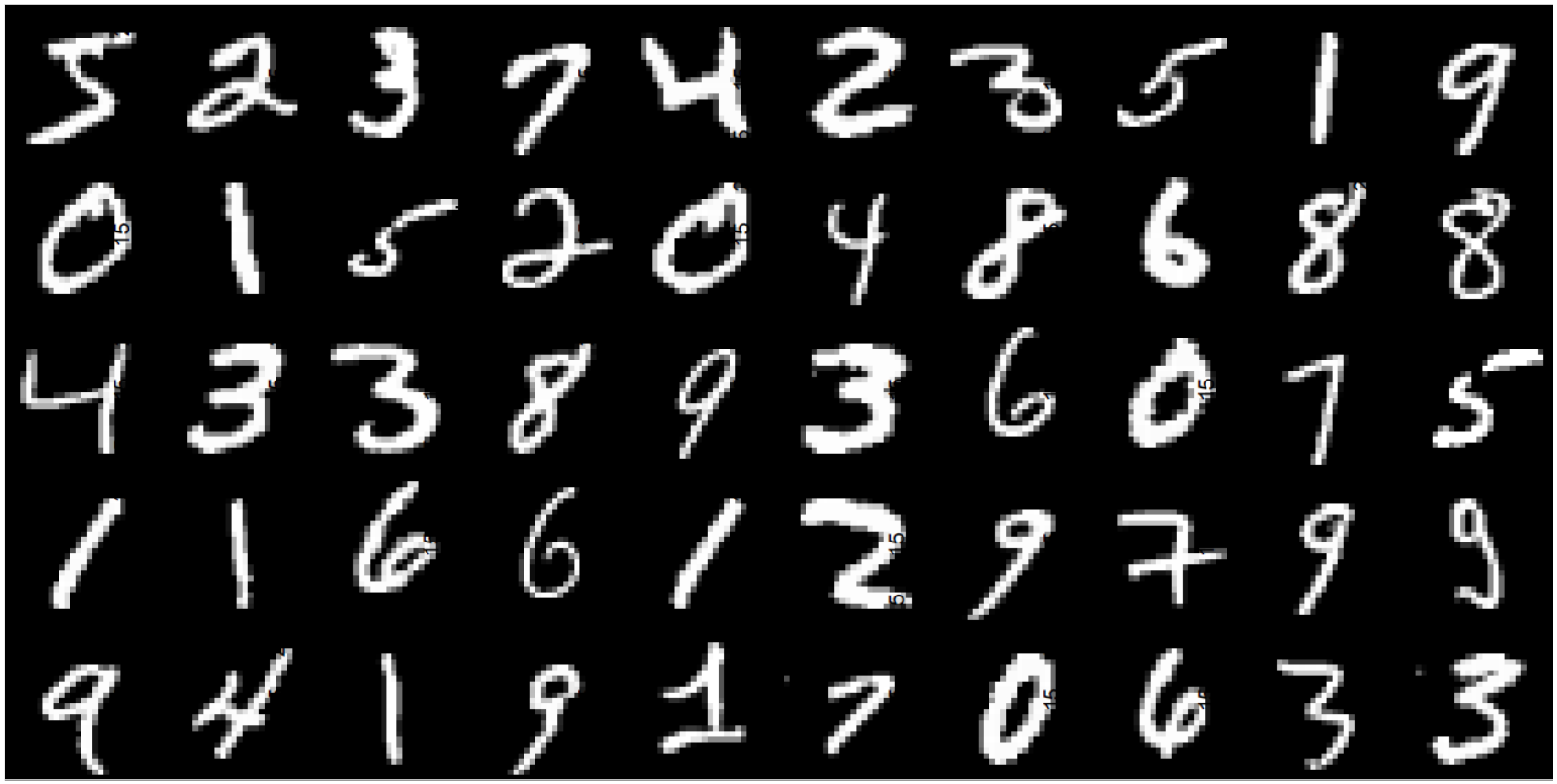


Image Recognition–Logistic regression single as neural layer network

- logistic function is the log-odds:

$$\text{logit}(p(y=1 \mid \mathbf{x})) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p)$$
$$= b + w_1 x_1 + \ldots + w_n x_n = z$$

- logistic sigmoid is the inverse of logistic function:

$$\text{logit}^{-1}(p(y=1 \mid \mathbf{x})) = \frac{1}{1+e^{-(b+w_1 x_1 + \ldots + w_n x_n)}} = \sigma(z)$$

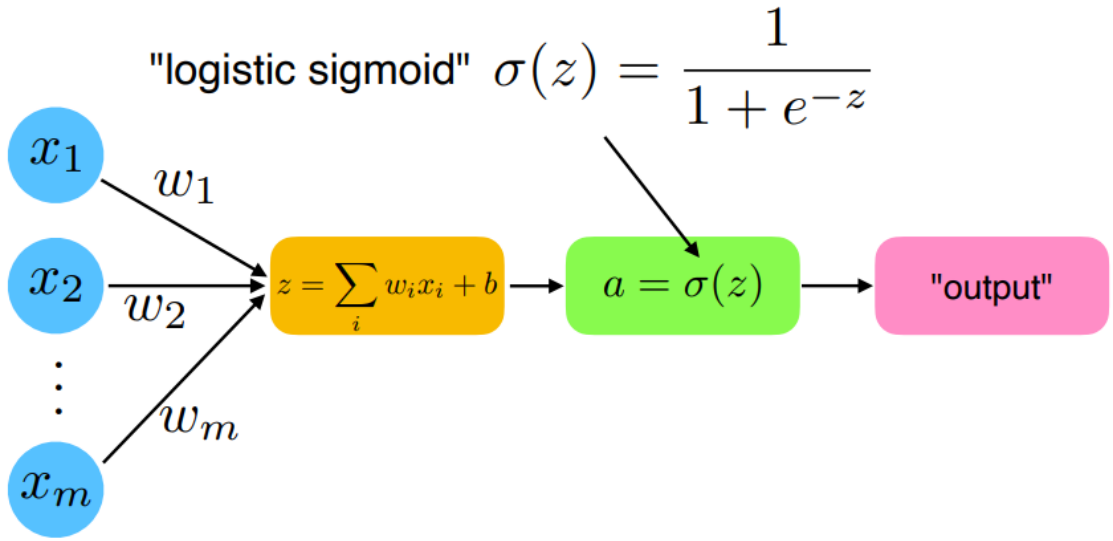


Image Recognition–Multinomial logistic regression in R

```
#1.Load the MNIST dataset:
mnist=dataset_mnist()
X_train=mnist$train$x
X_test=mnist$test$x
y_train=mnist$train$y
y_test=mnist$test$y
#2.Reshape the input data:
X_train=array_reshape(X_train, c(nrow(X_train), 784))
X_train=X_train / 255
X_test=array_reshape(X_test, c(nrow(X_test), 784))
X_test=X_test / 255
#3.Fit a multinomial logistic regression model:
model_lr=multinom(as.factor(y_train) ~ .,
          data=as.data.frame(X_train),
          MaxNWts=10000,decay=5e-3, maxit=100)
prediction_lr=predict(model_lr, X_test, type = "class")
```

Image Recognition–Logistic regression result

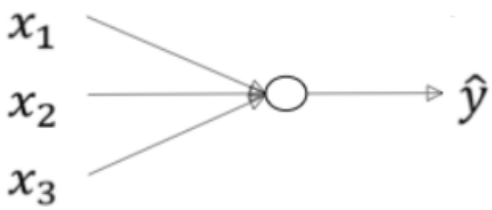
Table: Predicted class vs True labels in test data

	0	1	2	3	4	5	6	7	8	9
0	938	0	6	4	2	10	7	3	7	6
1	0	1112	13	1	4	3	5	15	20	7
2	1	3	914	22	3	1	7	18	6	2
3	2	2	14	905	2	25	0	4	20	9
4	4	0	12	3	921	13	12	9	14	38
5	12	1	7	31	0	785	15	2	33	11
6	14	4	13	4	8	16	908	1	12	2
7	4	1	11	10	3	9	1	942	11	25
8	4	12	38	20	8	22	3	1	839	7
9	1	0	4	10	31	8	0	33	12	902

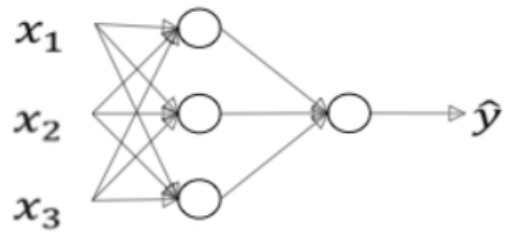
```
> accuracy=mean(prediction_lr == mnist$test$y)
> cat('Test␣accuracy:', accuracy, '\n')
Test accuracy: 0.9166
```

Image Recognition–Deep neural networks (DNN)

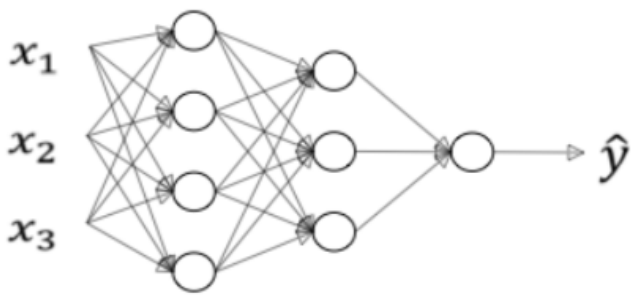
- What is deep neural networks?
- Why do we need it?



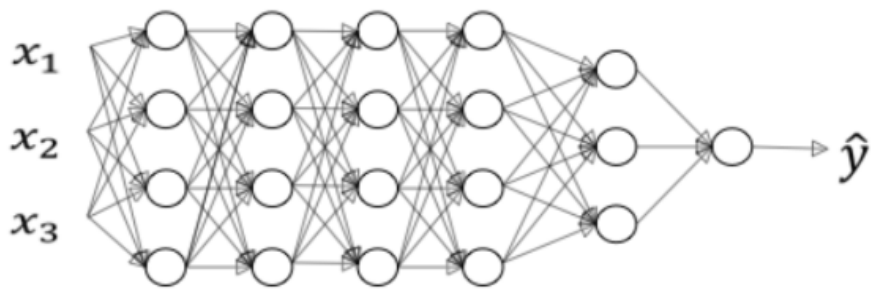
logistic regression



1 hidden layer



2 hidden layers



5 hidden layers

Image Recognition–What is neural network doing?

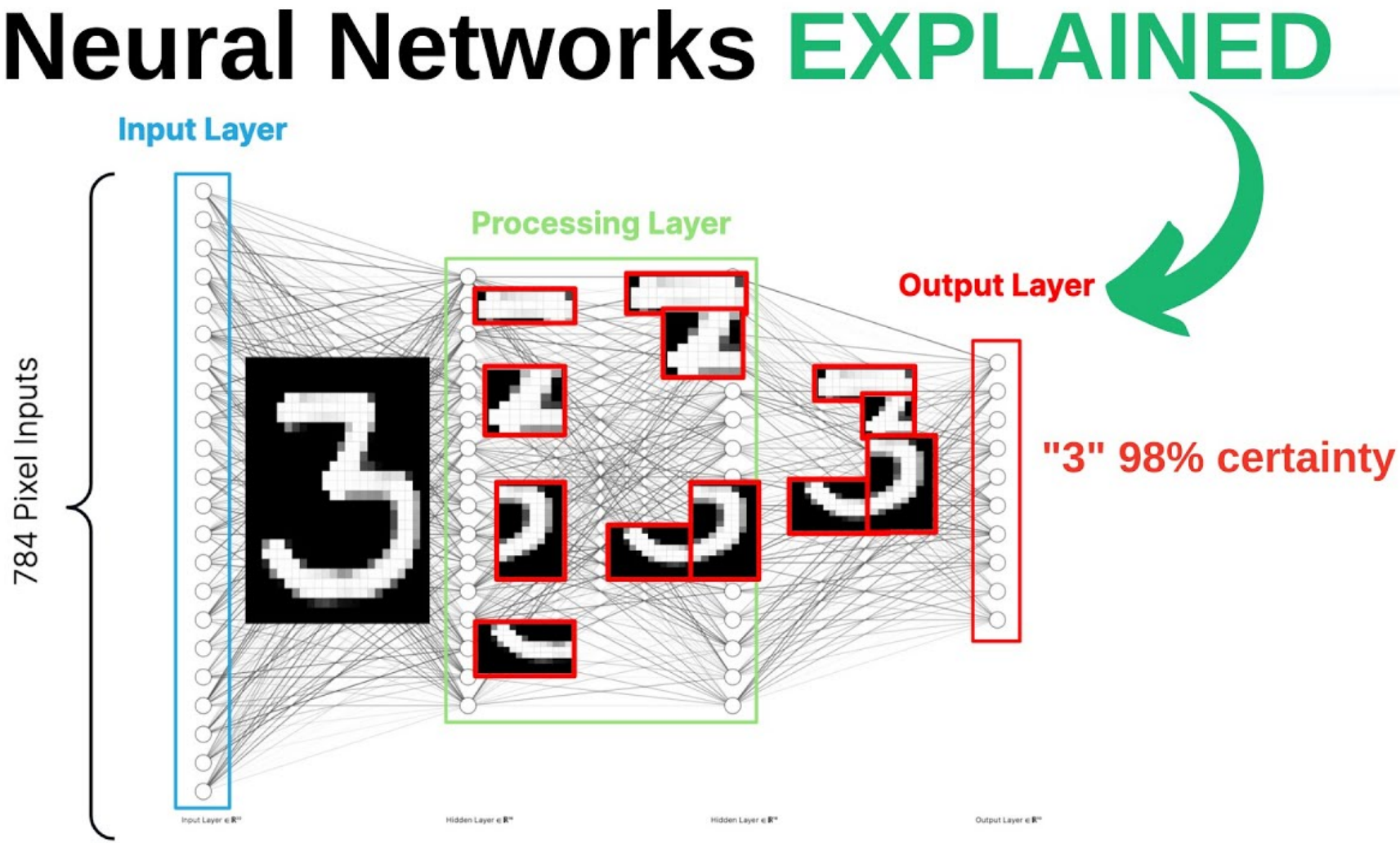


Image Recognition–Keras and TensorFlow in R

```
#1.Load the MNIST dataset:
mnist=dataset_mnist()
X_train=mnist$train$x
X_test=mnist$test$x
y_train=mnist$train$y
y_test=mnist$test$y
#2.Reshape the input data:
X_train=array_reshape(X_train, c(nrow(X_train), 784))
X_train=X_train / 255
X_test=array_reshape(X_test, c(nrow(X_test), 784))
X_test=X_test / 255
#3.Convert labels to one-hot encoding:
y_train=to_categorical(y_train, num_classes = 10)
y_test=to_categorical(y_test, num_classes = 10)
#4.Deep neural network model fitting:
```

Image Recognition–Model Training with dense layers

```
model=keras_model_sequential()%>% layer_dense(units=256,
  activation="relu",input_shape=c(784))%>%
  layer_dropout(rate= 0.25) %>%
  layer_dense(units= 128, activation= "relu") %>%
  layer_dropout(rate= 0.25) %>%
  layer_dense(units= 64, activation= "relu") %>%
  layer_dropout(rate= 0.25) %>%
  layer_dense(units= 10, activation= "softmax")
```

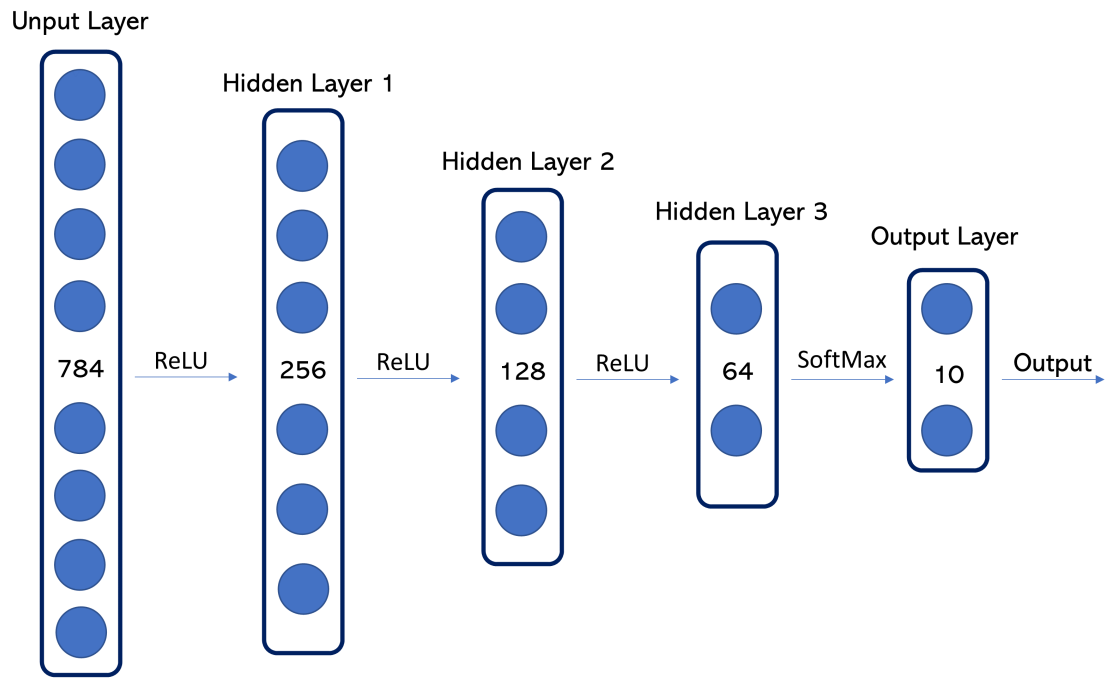


Image Recognition–Deep neural network result

Table: Predicted class vs True labels in test data

	0	1	2	3	4	5	6	7	8	9
0	972	1	0	0	0	1	3	1	0	2
1	0	1128	2	1	0	1	2	0	1	0
2	1	0	1018	2	1	0	1	7	2	0
3	1	0	6	993	0	2	0	4	2	2
4	2	0	2	0	964	0	4	5	0	5
5	2	0	0	14	0	867	4	0	3	2
6	2	2	0	1	3	4	944	1	1	0
7	1	2	5	1	0	0	0	1015	0	4
8	0	2	4	3	2	2	0	5	955	1
9	2	4	0	6	7	1	0	8	1	980

```
> accuracy=mean(predicted_classes == mnist$test$y)
> cat("Test␣accuracy:", accuracy, '\n')
Test accuracy: 0.9836
```

Image Recognition–Add Convolution layers

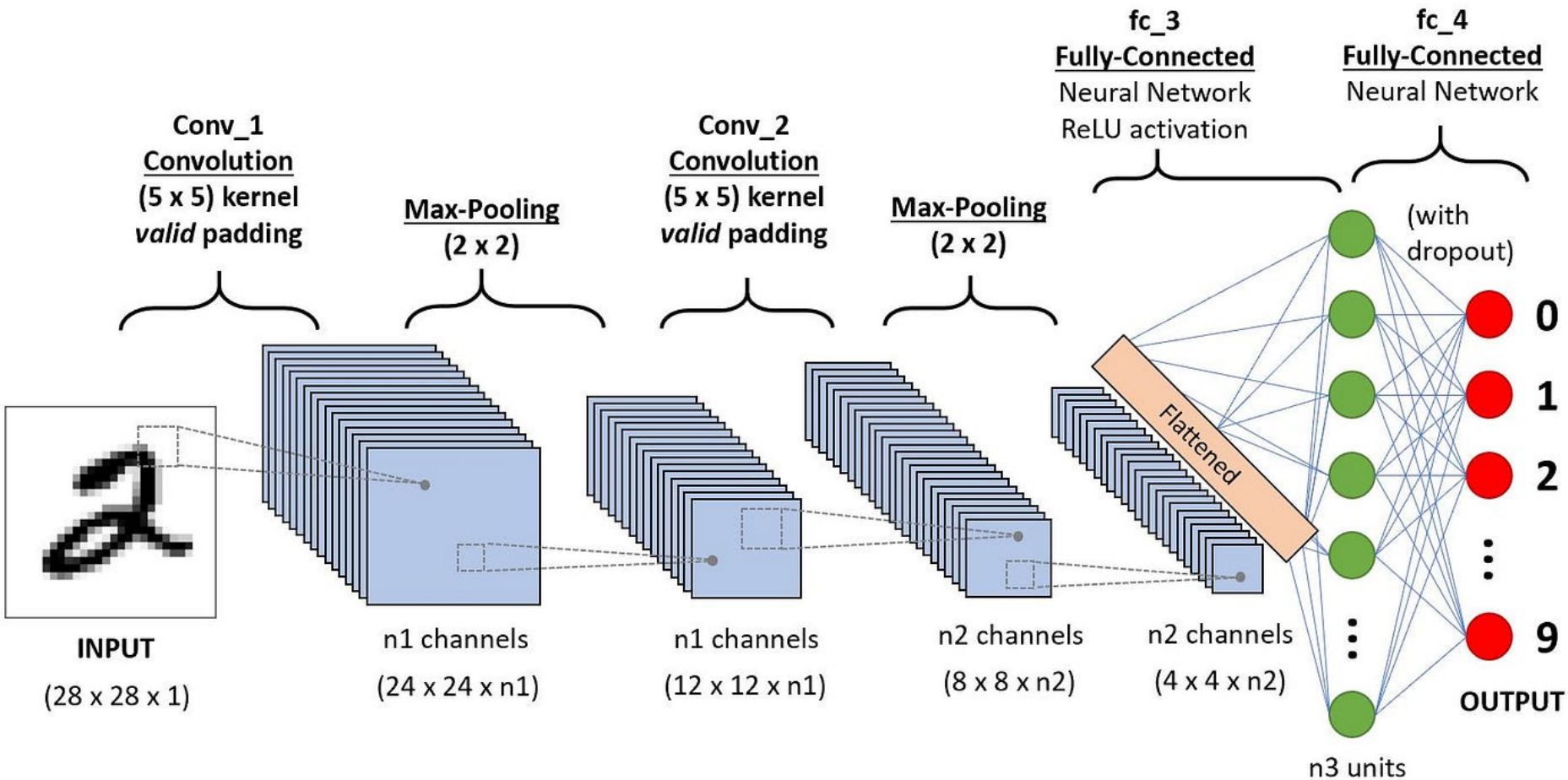


Image Recognition–Add Convolution layers