ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÀI THI MÔN: Machine Learning

Hình thức thi: Bài tập lớn

Đề tài:08

Giảng viên hướng dẫn : Cao Văn Chung

Sinh viên thực hiện : Nguyễn Quang Quân

Nguyễn Văn Thắng

Phạm Văn Tuấn

Lớp : K65A2 – Toán Tin

HÀ NỘI 2022

Lời nói đầu

Machine Learning là môn học rất quan trọng đối với sinh viên, đặc biệt là sinh viên ngành Toán Tin. Đây là môn học mang lại những định hướng cần thiết và quan trọng cho sinh viên khi còn trên ghế giảng đường đại học cũng như khi ra trường. Dựa vào các nguyên tắc và tư duy sẽ giúp cho mỗi sinh viên khi đứng trước một vấn đề nghiên cứu sẽ nhanh chóng tìm ra được giải pháp để giải quyết vấn đề.

Bài tập lớn này do nhóm em viết nhằm tổng kết, rút ra những bài học và để hiểu hơn những lý thuyết được thầy giảng dạy trên lớp. Qua đây nhóm em xin gửi lời cảm ơn chân thành đến thầy Cao Văn Chung đã tận tình dạy bảo chúng em.

Nội dung tiểu luận có thể không tránh khỏi một vài lỗi. Rất mong thầy thông cảm và cho những nhận xét và đánh giá để bài tiểu luận của chúng em được hoàn chỉnh hơn.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

I. Giới thiệu đề tài và dữ liệu

Cho một tập dữ liệu về hình ảnh chữ số viết tay (gồm 60000 mẫu trong tập training và 10000 trong tập test).

Dữ liệu được lấy từ: http://yann.lecun.com/exdb/mnist/ , các tệp tin được nén và được để trong 4 tệp với thông tin như dưới đây:

train-images-idx3-ubyte: training set images (dữ liệu ảnh train)

train-labels-idx1-ubyte: training set labels (dữ liệu nhãn ứng với ảnh train)

t10k-images-idx3-ubyte: test set images (dữ liệu ảnh test)

t10k-labels-idx1-ubyte: test set labels (dữ liệu nhãn ứng với ảnh test)

Dữ liệu ảnh được các chữ số viết tay ở đây được lưu liên tiếp nhau và không theo định dạng ảnh, cụ thể trong cấu trúc file như sau:

Cấu trúc file **train-images-idx3-ubyte** chứ dữ liệu ảnh training:

[thứ tự byte]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images (số ảnh = 60000)
0008	32 bit integer	28	number of rows (số dòng mỗi ảnh)
0012	32 bit integer	28	number of columns (số cột mỗi ảnh)
0016	unsigned byte	??	cường độ pixel thứ nhất
0017	unsigned byte	??	cường độ pixel thứ hai
....			
xxxx	unsigned byte	??	cường độ pixel cuối cùng

Cường độ Pixels được sắp xếp cạnh nhau thành dòng. Giá trị cường độ Pixel là từ 0 đến 255 (1byte) nhưng để ngược: 0 là background (trắng),và 255 là foreground (đen), tuy nhiên khi in ảnh ra màn hình thì điều này không quan trọng.

Cấu trúc file **train-labels-idx1-ubyte** chứa nhãn của cách ảnh training:

[thứ tự byte]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items (số nhãn ảnh = 60000)
0008	unsigned byte	??	label cho ảnh 1
0009	unsigned byte	??	label cho ảnh 2
....			
xxxx	unsigned byte	??	label cho ảnh cuối

Ở đây nhãn cho ảnh là số nguyên từ 0 đến 9 (ứng với chữ số trong ảnh).

Cấu trúc của tệp dữ liệu ảnh test (train-images-idx3-ubyte) và nhãn ảnh test (t10k-labels-idx1-ubyte) tương tự như với dữ liệu training, số lượng ảnh là 10000.

II. Rút gọn số chiều dữ kiệu

1. Lấy dữ liệu

Do tệp dữ liệu ở định dạng nén nên cần đoạn chương trình giải nén.

Đầu tiên, ta lấy dữ liệu vào trong chương trình. Đoạn códe dưới đọc tệp dữ liệu.

```
import os
import numpy as np

data_path = '/home/nguyenquan/Desktop/Baitaplon_MachinLearning'

train_images_path = os.path.join(data_path, 'train-images-idx3-ubyte.gz')
train_labels_path = os.path.join(data_path, 'train-labels-idx1-ubyte.gz')

test_images_path = os.path.join(data_path, 't10k-images-idx3-ubyte.gz')
test_labels_path = os.path.join(data_path, 't10k-labels-idx1-ubyte.gz')
```

Xây dựng phương thức đọc dữ liệu từ tệp gzip, giải nén và đưa về định dạng là một dãy ảnh (một dãy ma trận nguyên).

```
def get_mnist_data (images_path, labels_path, num_images, shuffle = False, _is=True, image_size= 28):

    #read data
    import gzip       # to decompress gz (zip) file

    # open file training to read training data
    f_images = gzip.open(images_path, 'r')

    # skip 16 first bytes because these are not data, only deader infor
    f_images.read(16)


    # general: read num_images data samples if this parameter is set;
    # if not, read all (60000 training or 10000 test)
    real_num = num_images if not shuffle else (60000 if _is else 10000)

    # read all data to buf_images (28x28xreal_num)
    buf_images = f_images.read(image_size * image_size * real_num)

    # images
    images = np.frombuffer(buf_images, dtype=np.uint8).astype(np.float32)
    images = images.reshape(real_num, image_size, image_size,)

    # Read labels
    f_labels = gzip.open(labels_path,'r')
    f_labels.read(8)

    labels = np.zeros((real_num)).astype(np.int64)

    # rearrange to correspond the images and labels
    for i in range(0, real_num):
        buf_labels = f_labels.read(1)
        labels[i] = np.frombuffer(buf_labels, dtype=np.uint8).astype(np.int64)


    # shuffle to get random images data
    if shuffle is True:
        rand_id = np.random.randint(real_num, size=num_images)

        images = images[rand_id, :]
        labels = labels[rand_id,]

    # change images data to type of vector 28x28 dimentional
    images = images.reshape(num_images, image_size * image_size)
    return images, labels
```

Hàm xây dựng đổ dữ liệu vào biến ***train_images*** (ma trận có 60000 hàng và 28x28 cột ứng với kính thước của ảnh) và biến ***train_labels*** (1 mảng có các giá trị từ 0-9)

Gọi phương thức đọc dữ liệu để kiểm tra xem đọc đúng ko.

```
train_images, train_labels = get_mnist_data(train_images_path, train_labels_path, 60000)
test_images, test_labels = get_mnist_data(test_images_path, test_labels_path, 10000)
print(train_images.shape, train_labels.shape)
print(test_images.shape, test_labels.shape)
```

Kết quả nhận được.

```
[Running] python -u "/home/nguyenquan/Desktop/Baitaplon_MachinLearning/de_tai_8.py"
(60000, 784) (60000,)
(10000, 784) (10000,)
```

2. Rút gọn tập dữ liệu và hiểu thị trực quan

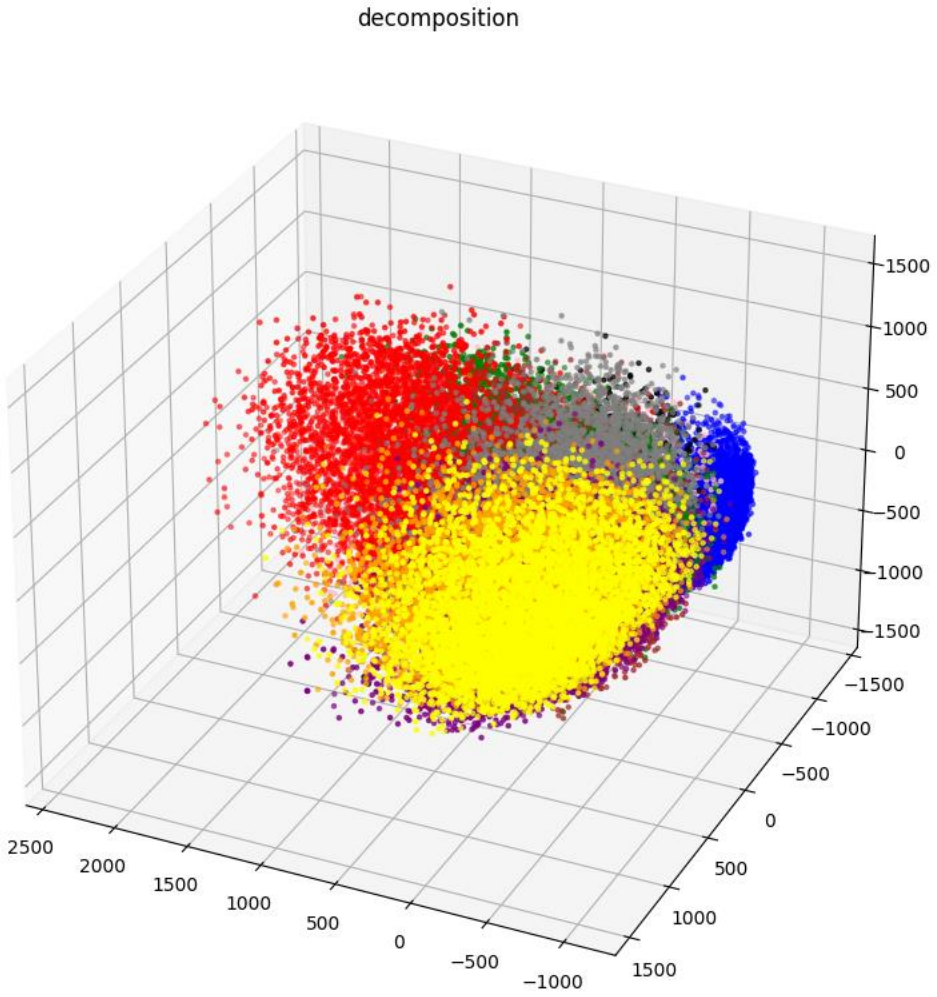
Để rút gọn số chiều dữ liệu, ta sử dụng PCA trong **sklearn.**

Để hiện thị trực quan các phân lớp dữ liệu dạng 3D và 2D ta rút gọn số chiều của dữ liệu từ 784 về 3 chiều và 2 chiều. Việc rút gọn dữ liệu và hiển trị trực quan dữ liệu được thực hiện qua đoạn code bên dưới.

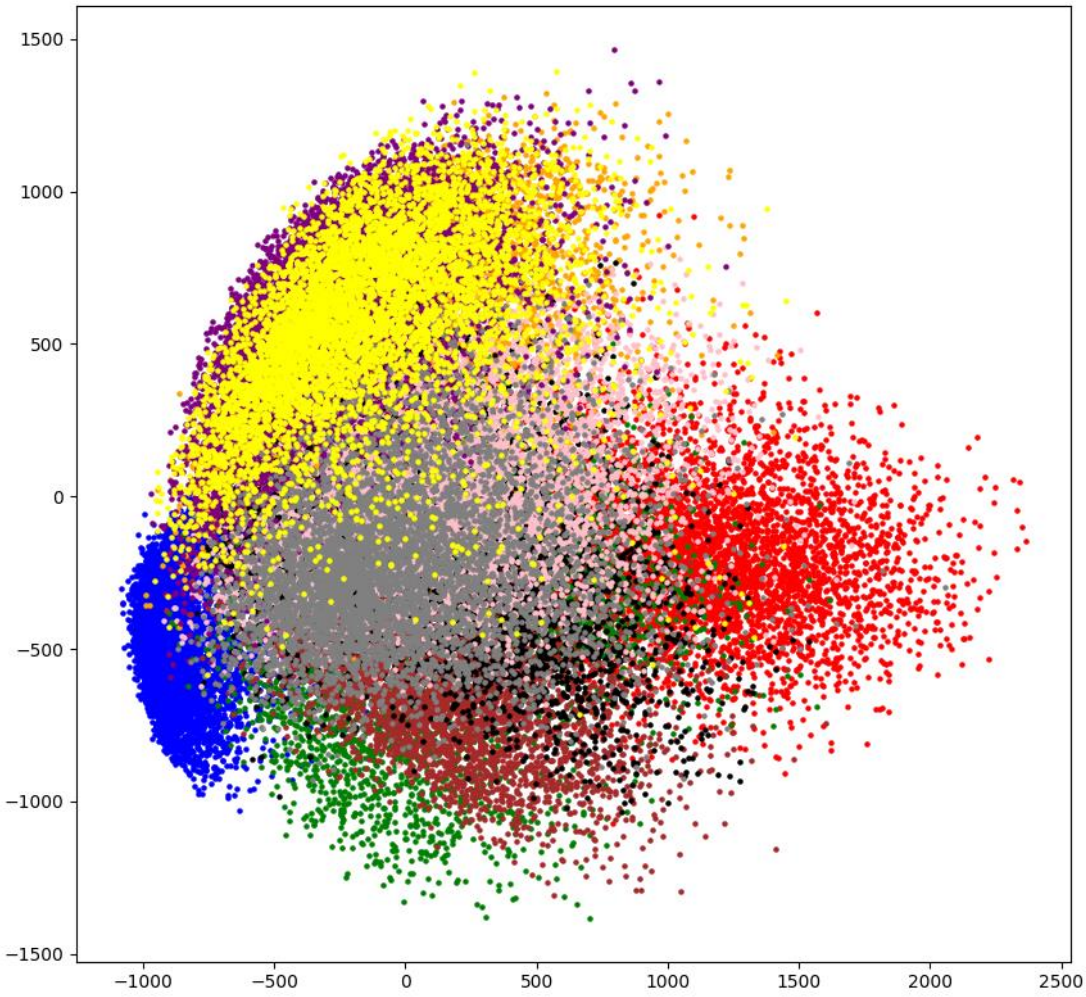
```
from sklearn.decomposition import PCA

pca3D = PCA(n_components=3)
pca3D.fit(train_images)
pac_transform3D = pca3D.transform(train_images)

fig3 = plt.figure()
fig3.set_size_inches(10,10)
ax = plt.axes(projection='3d')
colors = ['red','blue','green','brown', 'orange', 'black', 'pink','purple','gray','yellow']

for label in range(10):
    ax.scatter(pac_transform3D[train_labels==label,0],
               pac_transform3D[train_labels==label,1],
               pac_transform3D[train_labels==label, 2],
               s= 5, c = colors[label])

ax.set_title('decomposition')
plt.show()


pca2D = PCA(n_components=2)
pca2D.fit(train_images)
pac_transform2D = pca2D.transform(train_images)

fig2 = plt.figure()
fig2.set_size_inches(10,10)
for label in range(10):
    plt.scatter(pac_trans
               pac_trans
               s=5, c = colors[label])
plt.show()
```

(variable) colors: list[str]

Đồ thị nhận được sau khi giảm số chiều về 3.



Đồ thị sau khi giảm số chiều về 2.



III. Phân cụm dữ liệu

Để phân cụm dữ liệu ta sử dụng thuật toán K-Means. Mỗi cụm dữ liệu được đặc trưng bởi một tâm. Tâm là điểm đại diện nhất cho một cụm và có giá trị bằng trung bình của toàn bộ các quan sát trong cụm. Chúng ta sẽ dựa vào khoảng cách từ mỗi quan sát tới các tâm để xác định nhãn cho chúng thuộc về tâm gần nhất. Thuật toán sẽ khởi tạo ngẫu nhiên một số lượng xác định trước tâm cụm. Sau đó tiến hành xác định nhãn cho từng điểm dữ liệu và tiếp tục cập nhật lại tâm cụm. Thuật toán sẽ dừng cho tới khi toàn bộ các điểm dữ liệu được phân về đúng cụm hoặc số lượt cập nhật tâm chạm ngưỡng .

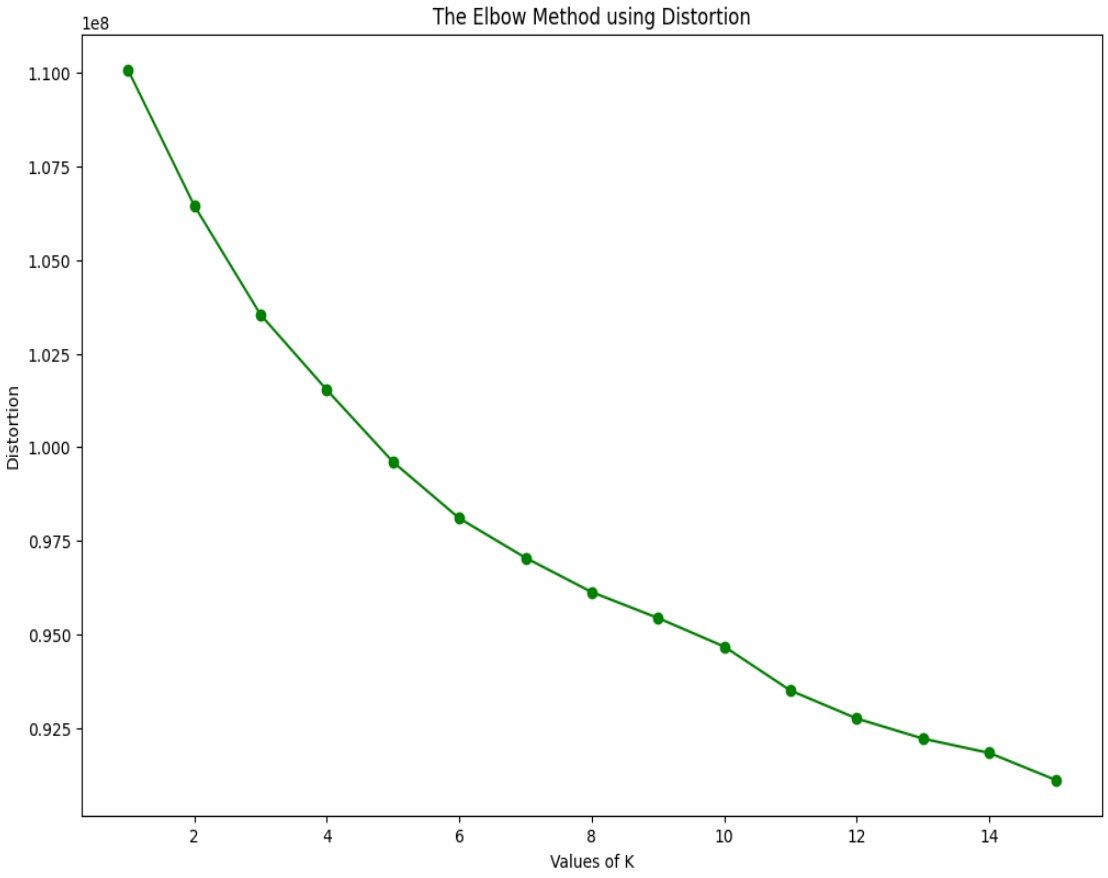
Ta đi xây dựng hàm tính sử dụng thuật toán k-means.

```python
#mean-K clustering
import math
from sklearn.cluster import KMeans
from scipy.spatial import distance as dist
def kmeans_creat_centers(X, k):
    return X[np.random.choice(X.shape[0], k, replace=False),:]

def kmeans_assign_labels(X, centers):
    D = dist.cdist(X, centers)
    return np.argmin(D, axis=1)

def kmeans_update_centers(X, labels, k):
    centers = np.zeros((k, X.shape[1]))
    for i in range(k):
        Xk = X[labels == i, :]
        centers[i,:] = np.mean(Xk, axis=0)
    return centers

def has_converged(centers, new_centers):
    return (set([tuple(a) for a in centers]) == set([tuple(a) for a in new_centers]))

def kmeans(X, k):
    centers = [kmeans_creat_centers(X,k)]
    labels = []
    it = 0
    while True:
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], k)
        if has_converged(centers[-1], new_centers):
            break
        centers.append(new_centers)
        it +=1
    return(centers, labels, it)


def kmeans_display(train_image, label_iamge, k):
    fig2 = plt.figure()
    fig2.set_size_inches(10,10)
    for label in range(k):
        plt.scatter(train_image[label_iamge == label,0],
                train_image[label_iamge == label,1],
                s=5, c = colors[label])
    plt.show()
```

Do k là số cụm ta không biết trước, do đó ta sử dụng phương pháp Elbow để tìm k phù hợp cho mô hình.

```
train_array = np.array(train_images)
distortions = np.zeros((15))
K = 15

for i in range(K):
    (centers, labels, it) = kmeans(train_array, i+1)
    centers_array = np.array(centers[-1])
    labels_array = np.array(labels[-1])
    cdist_data =0
    for index in range(60000):
        cdist_data +=math.dist(train_array[index,:], centers_array[labels_array[index],:])
    distortions[i] = cdist_data


plt.figure(figsize=(12, 8))
plt.plot(np.arange(1, K+1, 1), distortions, 'go-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```

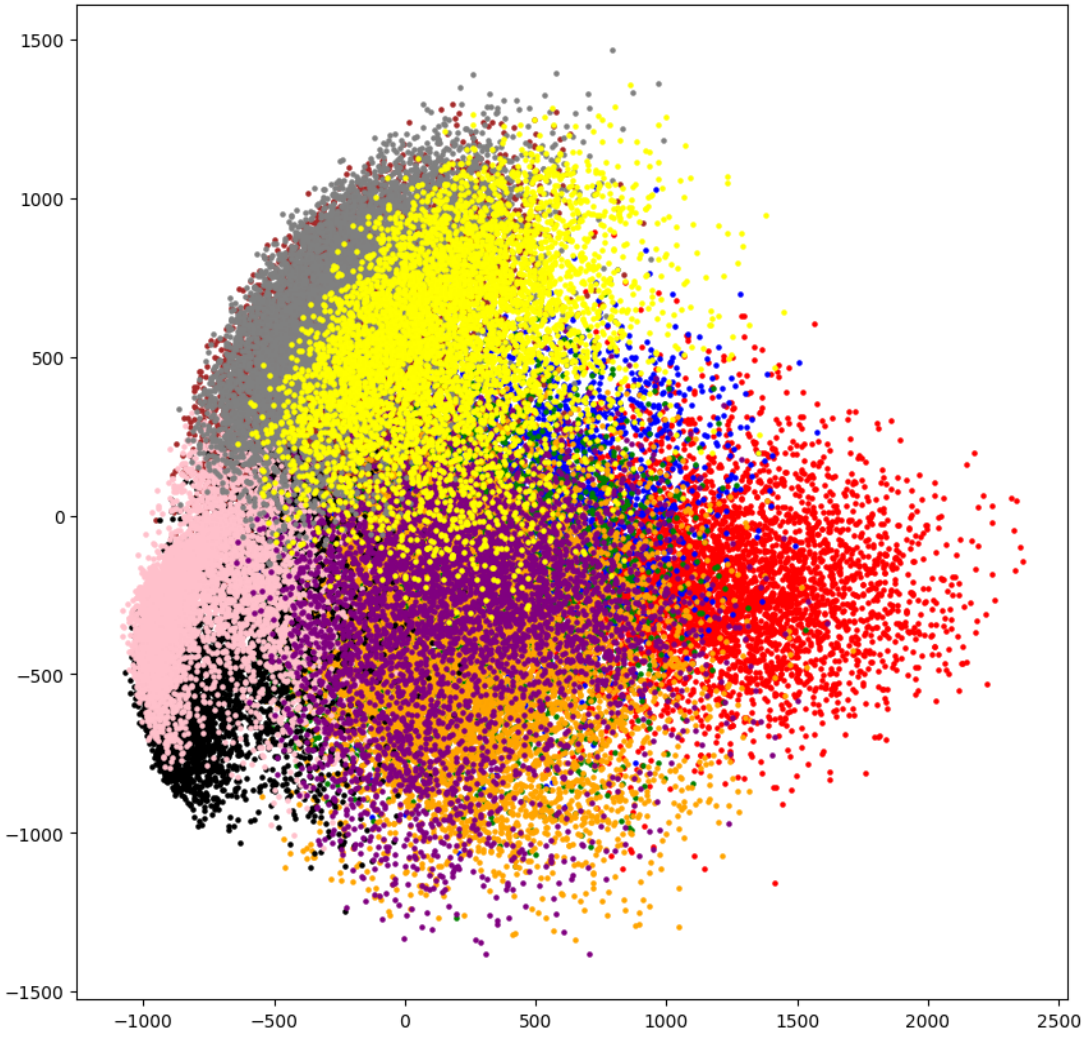
Ta thể hiện sự thay đổi của k qua đồ thị.



Ta nhận thấy khi K = 10 thì mô hình có Distortion thay đổi lớn nhất. Do vậy ta chọn K= 10 (10 phần cụm) cho mô hình này.

Ta cho chạy thuật toán phân cụm và hiển thị trực quan trên mô hình ta được.

```
(centers, label_image, it)= kmeans(train_images, 10)
print(it)
print(centers[-1])
print(label_image[-1])
kmeans_display(pac_transform2D, label_image[-1], 10)
```



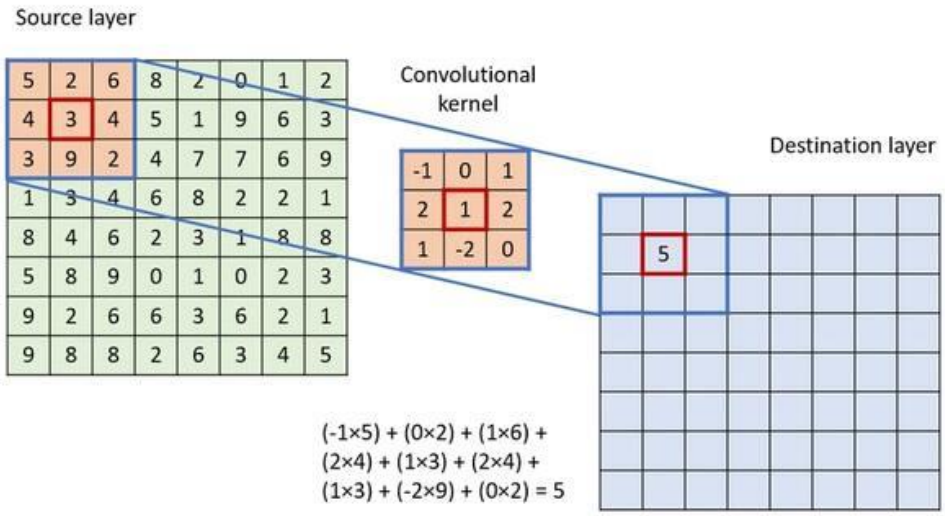
IV. Mô hình CNN

Để phân loại ảnh chính xác hơn ta sử dụng mô hình CNN.

CNN viết tắt của cụm từ Convolutional Neural Network. Đây là mô hình vô cùng tiên tiến được áp dụng nhiều trong lĩnh vực Deep learning. Mạng CNN cho phép người dùng xây dựng những hệ thống phân loại và dự đoán với độ chính xác cực cao. CNN được sử dụng nhiều trong xử lý ảnh.

CNN bao gồm những lớp cơ bản sau:

- Convolutional layer là phần quan trọng nhất trong CNN, nó có nhiệm vụ thực thi các tính toán. Các yếu tố quan trọng trong lớp Convolutioinal là:
    - Filter map, CNN sử dụng filter để áp dụng vào các vùng của ma trận hình ảnh. Các filter map là các ma trận vuông có cỡ là số lẻ. Bên trong đó là những tham số và chúng được gọi là paramaters.
    - Stride là dịch chuyển filter map theo từng pixel dựa vào các giá trị từ trái qua phải.
    - Padding là các giá trị viền xung quanh của mạ trận hình ảnh sẽ được gán các giá trị 0 để có thể tiến hành nhân tích chập mà không làm giảm kích thức ma trận ảnh ban đầu.
    - Feature map là kết quả sau mỗi lần feature map quét qua ma trận ảnh đầu vào. Sau mỗi lần quét thì lớp Convolution sẽ tiến hành tính toán.

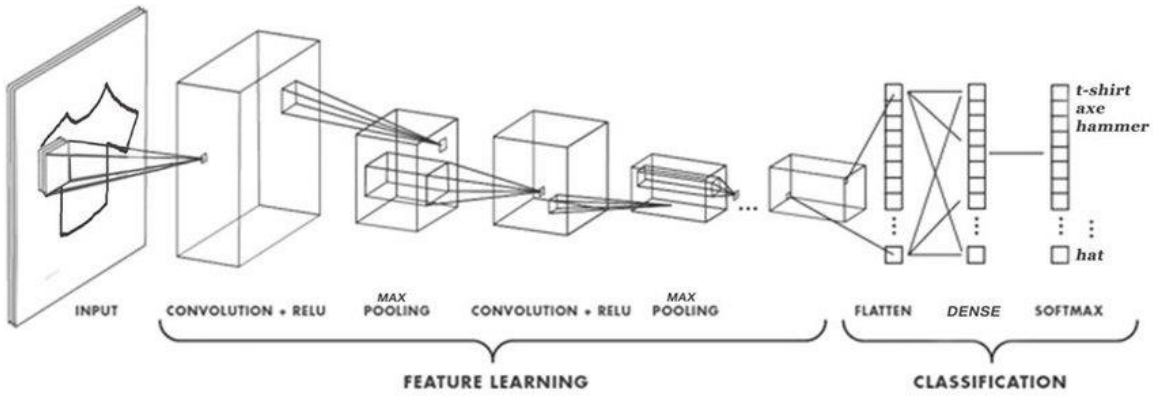


- ReLU layer là lớp kích hoạt trong CNN, được gọi là activation function. ReLU là một hàm phi tuyến. Với đầu ra là: f(x) = max(0,x).

- Pooling layer sẽ giảm bớt số lượng tham số khi hình ảnh quá lơn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mấu xuống làm giảm kích thước mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các Pooling có nhiều loại khác nhau:
    - Max Pooling
    - Average Pooling
    - Sum Pooling

- Fully connected có nhiệm vụ đưa ra kết quả sau khi lớp Convolutional layer và Pooling layer đã nhận được ảnh truyền. Lúc này, ta thu được kết quả model đã đọc được thông tin của ảnh và để liên kết chúng cũng như cho ra số output phù hợp thì ta sử dụng fully connected layer.

Hình dưới đây mô tả rõ hơn về mô hình CNN



Ta quay lại bài toán phân biệt chữ số viết tay. Ta xây dựng mô hình CNN với kiến trúc như sau:

- 03 tầng tích chập hỗn hợp Convolution, Activation ReLU, Max Pooling.
- 02 tầng fully connected với số unit phù hợp.
- Trong tầng cuối sử dụng **softmax**.

Đầu tiên ta import các thư viện cần dùng.

```
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import accuracy_score
```

Sau đó ta load dữ liệu từ dataset,mnist của keras. Bộ dự liệu được lưu dưới dạng ma trận hình ảnh (60000,28,28) đối với train_images và (60000,1) đối với train_labels, (10000, 28,28) đối với test_images và (10000,1) đối với test_labels. Tồi ta thay đổi về định dạng phù hợp.

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
print(train_images.shape)

train_images = train_images.reshape(60000,28,28,1)
test_images = test_images.reshape(10000,28,28,1)
train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)
print(train_labels[0])
```

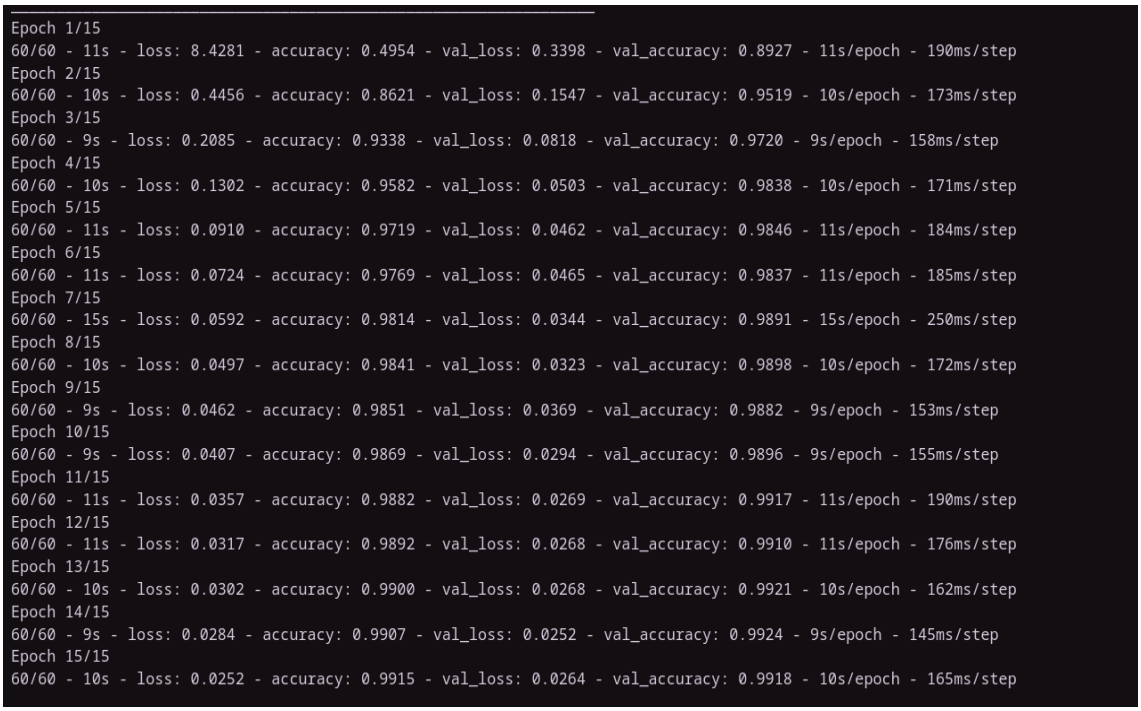
Sau đó ta xây dựng mô hình CNN theo mô tả ở trên.

```
model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(16, (3,3),padding="same", activation= 'relu', input_shape =(28,28,1)),
    tf.keras.layers.MaxPooling2D((2,2), strides=2),
    tf.keras.layers.Dropout(rate= 0.15),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu' , padding="valid"),
    tf.keras.layers.MaxPooling2D((2,2), strides=2),
    tf.keras.layers.Dropout(rate=0.1),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', padding="same"),
    tf.keras.layers.MaxPooling2D((2,2), strides=2),
    tf.keras.layers.Dropout(rate=0.10),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=256, activation='relu', use_bias=True, bias_initializer='zeros'),
    tf.keras.layers.Dense(units=10, activation='softmax', use_bias=True, bias_initializer='zeros')
])
model.compile(optimizer= RMSprop(learning_rate = 0.001),
              loss= 'categorical_crossentropy',
              metrics= ['accuracy'])
model.summary()
```

Ta huấn luyện mô hình bằng tập dữ liệu train_images và train_labels, và kiểm tra độ chính xác bằng validation.

```
history = model.fit(
    train_images,
    train_labels,
    validation_data=(test_images, test_labels),
    batch_size=1000,
    callbacks=[callbacks],
    steps_per_epoch=60,
    epochs=15,
    verbose=2
)
```

Và kết quả nhận được của mô hình CNN như sau. Ta thấy được rằng mô hình CNN có độ chính xác đến 99,18% sau 15 Epoch.



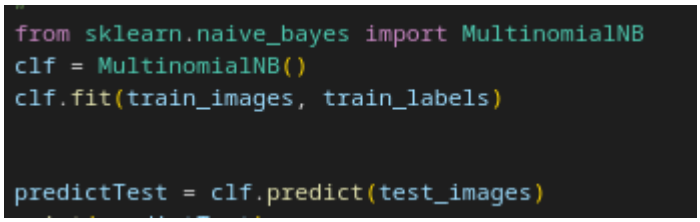
```
_________________________________________________________________
Epoch 1/15
60/60 - 11s - loss: 8.4281 - accuracy: 0.4954 - val_loss: 0.3398 - val_accuracy: 0.8927 - 11s/epoch - 190ms/step
Epoch 2/15
60/60 - 10s - loss: 0.4456 - accuracy: 0.8621 - val_loss: 0.1547 - val_accuracy: 0.9519 - 10s/epoch - 173ms/step
Epoch 3/15
60/60 - 9s - loss: 0.2085 - accuracy: 0.9338 - val_loss: 0.0818 - val_accuracy: 0.9720 - 9s/epoch - 158ms/step
Epoch 4/15
60/60 - 10s - loss: 0.1302 - accuracy: 0.9582 - val_loss: 0.0503 - val_accuracy: 0.9838 - 10s/epoch - 171ms/step
Epoch 5/15
60/60 - 11s - loss: 0.0910 - accuracy: 0.9719 - val_loss: 0.0462 - val_accuracy: 0.9846 - 11s/epoch - 184ms/step
Epoch 6/15
60/60 - 11s - loss: 0.0724 - accuracy: 0.9769 - val_loss: 0.0465 - val_accuracy: 0.9837 - 11s/epoch - 185ms/step
Epoch 7/15
60/60 - 15s - loss: 0.0592 - accuracy: 0.9814 - val_loss: 0.0344 - val_accuracy: 0.9891 - 15s/epoch - 250ms/step
Epoch 8/15
60/60 - 10s - loss: 0.0497 - accuracy: 0.9841 - val_loss: 0.0323 - val_accuracy: 0.9898 - 10s/epoch - 172ms/step
Epoch 9/15
60/60 - 9s - loss: 0.0462 - accuracy: 0.9851 - val_loss: 0.0369 - val_accuracy: 0.9882 - 9s/epoch - 153ms/step
Epoch 10/15
60/60 - 9s - loss: 0.0407 - accuracy: 0.9869 - val_loss: 0.0294 - val_accuracy: 0.9896 - 9s/epoch - 155ms/step
Epoch 11/15
60/60 - 11s - loss: 0.0357 - accuracy: 0.9882 - val_loss: 0.0269 - val_accuracy: 0.9917 - 11s/epoch - 190ms/step
Epoch 12/15
60/60 - 11s - loss: 0.0317 - accuracy: 0.9892 - val_loss: 0.0268 - val_accuracy: 0.9910 - 11s/epoch - 176ms/step
Epoch 13/15
60/60 - 10s - loss: 0.0302 - accuracy: 0.9900 - val_loss: 0.0268 - val_accuracy: 0.9921 - 10s/epoch - 162ms/step
Epoch 14/15
60/60 - 9s - loss: 0.0284 - accuracy: 0.9907 - val_loss: 0.0252 - val_accuracy: 0.9924 - 9s/epoch - 145ms/step
Epoch 15/15
60/60 - 10s - loss: 0.0252 - accuracy: 0.9915 - val_loss: 0.0264 - val_accuracy: 0.9918 - 10s/epoch - 165ms/step
```

V. Xây dựng chương trình sử dụng mô hình Naive Bayes

Ta nhận thấy tệp dữ liệu image có mẫu là các biến rời rạc (giá trị của mỗi điểm ảnh là một số nguyên nằm trong khoảng 0-255) nên ta không thể sử dụng mô hình Gaussian Naïve Bayes (vì Gaussian Naïve Bayes được sử dụng khi các mẫu là các biến liên tục).

Và ta cũng thấy rằng giá trị của các biến không phải là giá trị nhị phân (giá trị nhận là 0,1) nên ta sẽ sử dụng mô hình Multinomial Naive Byes.

Ta sử dụng phương thức MultinomialNB() trong thư viện **sklearn** để giải quyết bài toán.



VI So sánh độ chính xác của các mô hình

Ta tính độ chính xác của mô hình qua bốn cách sau:

• Accuracy : tỷ lệ giữa số lượng dự đoán đúng và tổng số mẫu được dự đoán.

• Confusion matrix : là ma trận dùng để đánh giá hiệu suất của thuật toán được sử dụng

• Recall : Recall là tỉ lệ giữa số lượng các trường hợp được dự đoán là positive (dự đoán đúng) và thực tế cũng là positive (ground truth), trên tổng số các trường hợp positive trong dữ liệu

• Precision : cho biết tỉ lệ số lượng các trường hợp được phân loại đúng trong số các trường hợp được phân loại là positive.

Ta sử dụng thư viện **sklearn** để tính các mô hình.

Code tính độ chính xác của mô hình Multinomial Naive Bayes.

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score

accuracyMultinomial = accuracy_score(test_labels, predictTest)
confusionMatrixMultinomial= confusion_matrix(test_labels, predictTest)
precisionMultinomial = precision_score(test_labels, predictTest, average='macro')
recallMultinomial = recall_score(test_labels, predictTest, average='macro')

print("Accuracy:", accuracyMultinomial)
print("Confusion matrix:\n", confusionMatrixMultinomial)
print("Precision:", precisionMultinomial)
print("Recall:", recallMultinomial)
```

Ta thu được kết quả.

```
Accuracy: 0.8365
Confusion matrix:
 [[ 912    0    2    6    1    8   14    1   36    0]
 [   0 1061    5    9    0    2    6    0   51    1]
 [  15   11  858   24   10    3   33   11   66    1]
 [   4   11   34  851    1   21    7   14   40   27]
 [   2    2    6    0  732    0   25    1   38  176]
 [  23   11    6  107   18  590   17    6   78   36]
 [  17   13   17    1    7   25  860    0   18    0]
 [   1   21   11    5   19    0    1  861   40   69]
 [   6   26   13   54   14   27    8    9  777   40]
 [   6    7    3   10   66   10    0   17   27  863]]
Precision: 0.8433162997126132
Recall: 0.8334531845906966
```

Ta nhận thấy qua các kết quả thì mô hình CNN có độ chính xác cao hơn so với mô hình Multinomial Naïve Bayes.