

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY  
UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



## BÀI TẬP LỚN - HỌC MÁY (CO3117)

## MNIST HANDWRITTEN DIGIT

---

GVHD: ThS. Võ Thanh Hùng  
Lớp: L02  
SVTH: Lê Hoàng Khánh Vinh - 2213963  
Lê Trường Thống - 2213338  
Kiều Tâm Hậu - 2210961  
Lê Quốc Huy - 2211194



## Mục lục

<b>1 Giới thiệu đề tài</b>	<b>3</b>
<b>2 Dataset</b>	<b>3</b>
2.1 Tổng quan dữ liệu . . . . .	3
2.2 Xử lý dữ liệu ban đầu . . . . .	3
2.2.1 Khai báo các gói, thư viện cần thiết . . . . .	3
2.2.2 Đọc, xử lý và chuẩn hoá dữ liệu . . . . .	4
2.3 Dánh giá dữ liệu . . . . .	4
2.3.1 Kiểm tra phân bố số lượng và tính ngẫu nhiên . . . . .	4
2.3.2 Kiểm tra một số định dạng dữ liệu ban đầu . . . . .	6
2.3.3 Kiểm tra một số dữ liệu có thể khiến mô hình nhầm lẫn . . . . .	8
2.3.4 Sử dụng K-means clustering để đánh giá sơ bộ . . . . .	10
<b>3 Giải pháp 1: Logistics Regression with Dimensionality Reduction</b>	<b>12</b>
3.1 Logistics Regression without Principal Component Analysis (PCA) . . . . .	12
3.2 Logistics Regression with Principal Component Analysis (PCA) . . . . .	14
<b>4 Giải pháp 2: Support Vector Machine</b>	<b>16</b>
4.1 Support Vector Machine Classification with Linear Kernel . . . . .	17
4.2 Support Vector Machine Classification with Polynomial Kernel . . . . .	18
4.3 Support Vector Machine Classification with Gaussian RBF Kernel . . . . .	20
<b>5 Mở rộng - Mạng nơ ron tích chập</b>	<b>22</b>
5.1 Xây dựng mô hình mạng nơ-ron tích chập . . . . .	23
5.2 Hiển thị các thông tin về model . . . . .	24
5.3 Huấn luyện mô hình với số lượng epochs nhất định . . . . .	25
5.4 Một số biểu đồ so sánh hàm mất mát và độ chính xác của training data và validation data . . . . .	25
5.5 Sử dụng mô hình để dự đoán hình ảnh . . . . .	27
5.6 Metrics đánh giá kết quả làm việc . . . . .	28
5.6.1 Accuracy and Loss . . . . .	28
5.6.2 Confusion Matrix . . . . .	29
<b>6 Kết luận chung</b>	<b>31</b>



## Danh sách thành viên và mức độ đóng góp

STT	Họ và tên	MSSV	Nhiệm vụ	Đóng góp
1	Lê Hoàng Khánh Vinh	2213963	PCA + Logistics Regression.	100%
2	Lê Trường Thống	2213338	Phân tích dữ liệu + CNN.	100%
3	Kiều Tâm Hậu	2210961	Support Vector Machine.	100%
4	Lê Quốc Huy	2211194	Phân tích dữ liệu.	100%



## 1 Giới thiệu đề tài

Trong lĩnh vực thị giác máy tính và nhận dạng hình ảnh, việc xử lý và phân loại dữ liệu hình ảnh có độ phân giải cao là một thách thức đáng kể, đòi hỏi các phương pháp hiệu quả trong việc giảm chiều dữ liệu và xây dựng mô hình phân loại chính xác. Tập dữ liệu MNIST – bao gồm các hình ảnh chữ số viết tay từ 0 đến 9 – là một trong những bộ dữ liệu tiêu chuẩn được sử dụng rộng rãi để đánh giá hiệu quả của các thuật toán học máy trong bài toán phân loại hình ảnh.

Đề tài này tập trung vào việc triển khai và so sánh hai giải pháp học máy nhằm giải quyết bài toán phân loại chữ số viết tay trên tập MNIST. Giải pháp đầu tiên sử dụng Principal Components Analysis (PCA) để giảm chiều dữ liệu đầu vào, sau đó áp dụng Logistic Regression để xây dựng mô hình phân loại. Giải pháp này hướng đến việc cải thiện hiệu suất huấn luyện và giảm thiểu độ phức tạp tính toán bằng cách loại bỏ các đặc trưng dư thừa, đồng thời vẫn giữ lại những thông tin quan trọng nhất trong dữ liệu.

Giải pháp thứ hai áp dụng trực tiếp Support Vector Machine (SVM) – một mô hình học máy mạnh mẽ với khả năng tìm kiếm siêu mặt phân tách tối ưu giữa các lớp dữ liệu. SVM được đánh giá cao trong các bài toán phân loại nhị phân và đa lớp, đặc biệt khi dữ liệu không tuyến tính có thể được ánh xạ vào không gian đặc trưng cao hơn thông qua các hàm kernel.

Mục tiêu của đề tài là đánh giá hiệu quả của từng giải pháp về độ chính xác phân loại, thời gian huấn luyện và khả năng tổng quát hóa mô hình. Từ đó, rút ra nhận xét về ưu nhược điểm của từng phương pháp trong bối cảnh xử lý dữ liệu hình ảnh độ phân giải cao như MNIST.

## 2 Dataset

### 2.1 Tổng quan dữ liệu

Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology) là một cơ sở dữ liệu lớn về các chữ số viết tay thường được sử dụng để đào tạo nhiều hệ thống xử lý hình ảnh và mô hình học máy. Nó được tạo ra bằng cách "trộn lại" các mẫu từ các bộ dữ liệu gốc của NIST và đã trở thành chuẩn mực để đánh giá hiệu suất của các thuật toán phân loại hình ảnh.

MNIST chứa 60.000 hình ảnh dùng để đào tạo mô hình và 10.000 hình ảnh để thử nghiệm mô hình về chữ số viết tay.

Định dạng của bộ dữ liệu bao gồm nhiều testcase, mỗi testcase là một hình ảnh có kích thước  $28 \times 28$  được gán nhãn label từ 0 đến 9 và các giá trị " $i \times j$ " đại diện cho pixel ở vị trí hàng thứ i và cột thứ j, với các giá trị từ 0 đến 255. Các hình ảnh được chuẩn hóa để phù hợp với hộp giới hạn  $28 \times 28$  pixel và khử răng cưa, đưa vào các mức thang độ xám.

MNIST được sử dụng rộng rãi để đào tạo và thử nghiệm trong lĩnh vực học máy, đặc biệt là đối với các tác vụ phân loại hình ảnh.

Dataset dạng .csv của bộ dữ liệu này có thể được truy cập và tải về từ:

<https://www.kaggle.com/datasets/oddrationale/mnist-in-csv>

### 2.2 Xử lý dữ liệu ban đầu

#### 2.2.1 Khai báo các gói, thư viện cần thiết

Trong quá trình hiện thực, nhóm quyết định ngoài các thư viện và gói cơ bản có trong ngôn ngữ lập trình Python thì có sử dụng thêm một thư viện mã nguồn mở là Keras. Keras là một thư viện mã nguồn mở cung cấp giao diện Python cho mạng nơ-ron nhân tạo, được tích hợp vào thư viện TensorFlow. TensorFlow cung cấp nền tảng tính toán mạnh mẽ, còn Keras giúp đơn giản



hóa quá trình xây dựng và triển khai mô hình. Keras chứa nhiều method cho phép xây dựng mạng nơ-ron thường dùng như lớp, mục tiêu, hàm kích hoạt, trình tối ưu hóa và nhiều công cụ để làm việc với dữ liệu hình ảnh và văn bản nhằm đơn giản hóa lập trình trong lĩnh vực mạng nơ-ron.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import time
import seaborn as sns
import random as rd
import keras
from sklearn.metrics import confusion_matrix
```

### 2.2.2 Đọc, xử lý và chuẩn hoá dữ liệu

Trong thư viện **pandas** của Python có hỗ trợ phương thức **read\_csv()** để đọc vào file dữ liệu có định dạng .csv:

```
train_dataset_path = pd.read_csv("mnist_train.csv")
test_dataset_path = pd.read_csv("mnist_test.csv")
```

Từ dữ liệu ban đầu, chuyển đổi thành **train\_data** và **test\_data** về dạng giống như mảng (array-like) 1 chiều. **x\_train**, **x\_test** lần lượt là thông tin của 784 pixel đại diện cho mỗi bức ảnh kích thước  $28 \times 28$  của training data và test data. **y\_train**, **y\_test** lần lượt là label của các bức ảnh.

```
train_data = np.array(train_dataset_path)
test_data = np.array(test_dataset_path)

x_train = train_data[:, 1:]
x_test = test_data[:, 1:]

y_train = train_data[:, 0]
y_test = test_data[:, 0]
```

Ta định dạng lại **x\_train**, **x\_test** từ các list có 784 giá trị pixel thành các tensor 2 chiều kích thước  $28 \times 28$  đại diện cho hình ảnh.

```
x_train = x_train.reshape(x_train.shape[0], *(28, 28, 1))
x_test = x_test.reshape(x_test.shape[0], *(28, 28, 1))
```

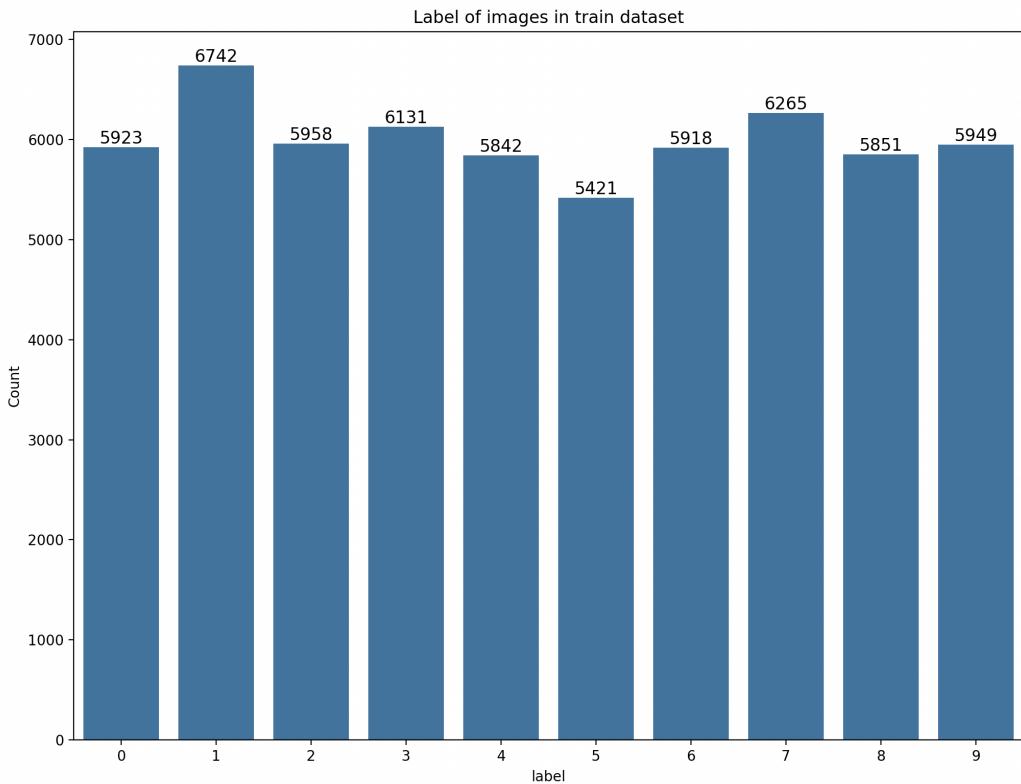
## 2.3 Dánh giá dữ liệu

### 2.3.1 Kiểm tra phân bố số lượng và tính ngẫu nhiên

Vẽ biểu đồ dạng cột kiểm tra phân bố số lượng các label trong training dataset:

```
fig = plt.figure(figsize=(12, 9))
axes = sns.countplot(x="label", data= train_dataset_path)
for p in axes.patches:
    axes.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2, p.
    get_height()), ha='center', va='bottom', fontsize=12, color='black')
axes.set_ylabel('Count')
```

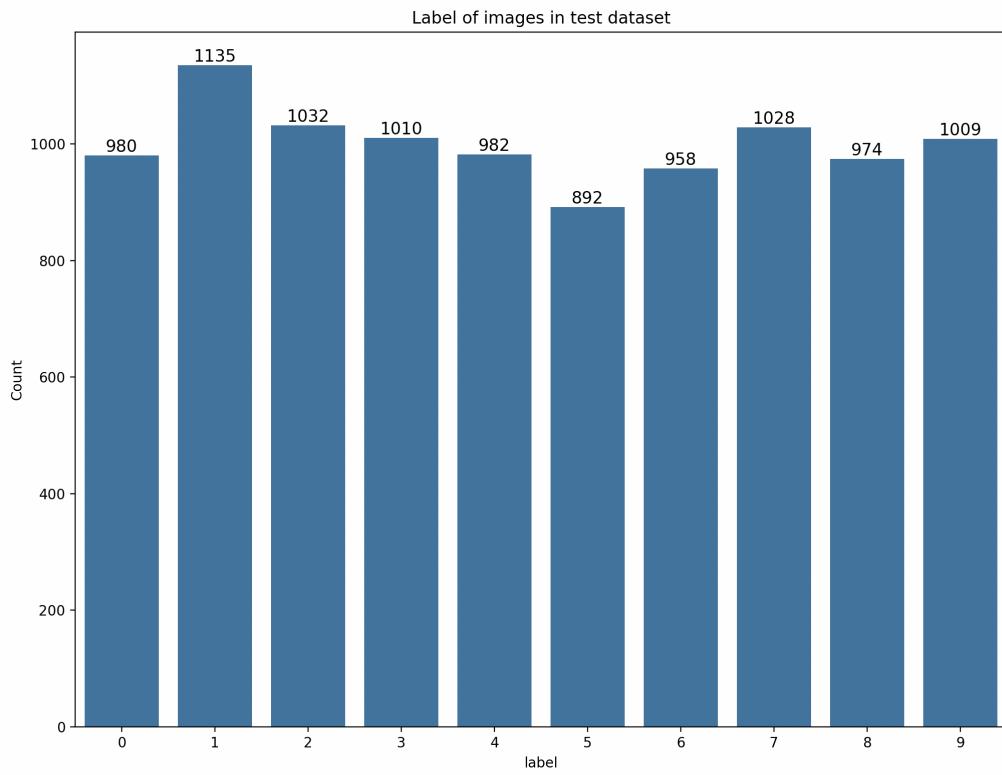
```
axes.set_title('Label of images in train dataset')
plt.show()
```



Hình 1: Phân bố data label trong tập training dataset

Vẽ biểu đồ dạng cột kiểm tra phân bố số lượng các label trong testing dataset:

```
fig = plt.figure(figsize=(12, 9))
axes = sns.countplot(x="label", data= test_dataset_path)
for p in axes.patches:
    axes.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2, p.
        get_height()), ha='center', va='bottom', fontsize=12, color='black')
axes.set_ylabel('Count')
axes.set_title('Label of images in test dataset')
plt.show()
```



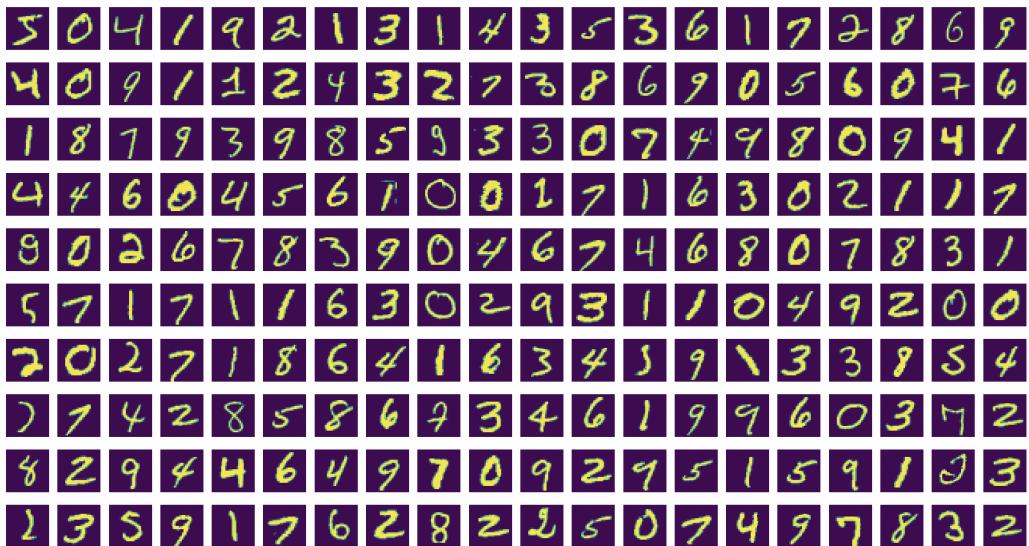
Hình 2: Phân bố data label trong tập test dataset

Từ hai biểu đồ trên, có thể nhận thấy trong cả 2 bộ dữ liệu, các dữ liệu phân bố tương đối ngẫu nhiên, tuy không quá đồng đều nhưng chênh lệch là không quá lớn, có thể hạn chế được việc mô hình đạt được giá trị chính xác cao vì chỉ học và ghi nhớ tốt đặc trưng của một số label.

### 2.3.2 Kiểm tra một số định dạng dữ liệu ban đầu

Ta thử in ra hình ảnh của khoảng 200 dữ liệu đầu tiên:

```
plt.figure(figsize=(18, 9))
for i in range(200):
    plt.subplot(10, 20, i + 1)
    plt.imshow(x_train[i])
    plt.axis('off')
plt.show()
```

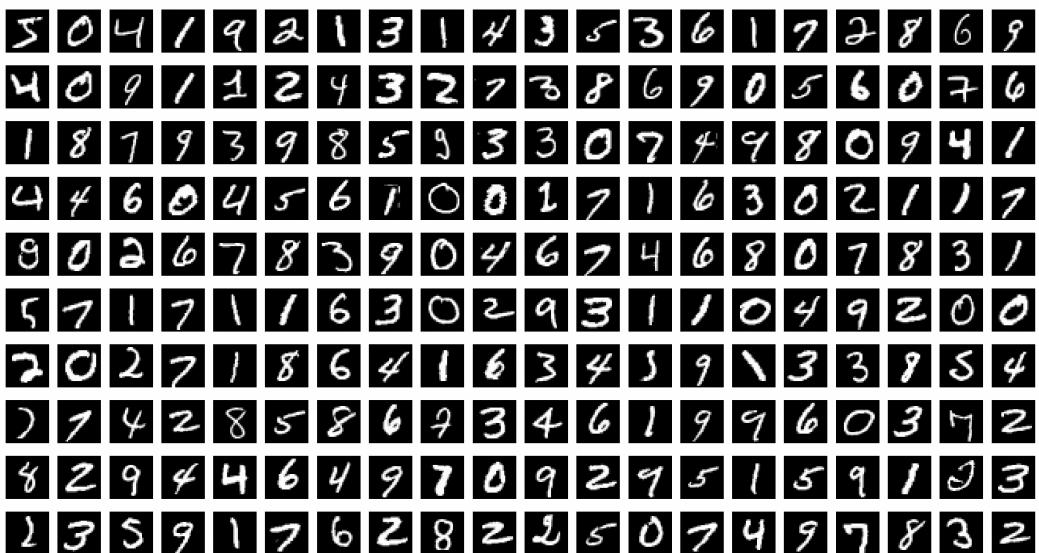


Hình 3: 200 dữ liệu đầu tiên của training dataset

Mỗi pixel đại diện cho điểm ảnh có giá trị từ 0 đến 255, chuẩn hoá dữ liệu bằng cách chia tỷ lệ để mỗi điểm ảnh có giá trị từ 0 đến 1, giúp giảm độ lớn của các phép tính số học, giúp mô hình học nhanh hơn và ổn định hơn. Trong quá trình huấn luyện mạng nơ-ron, các gradient được sử dụng để cập nhật các tham số. Khi giá trị đầu vào quá lớn, mô hình có thể gặp phải vấn đề vanishing gradient hoặc exploding gradient, làm cho việc tối ưu trở nên khó khăn.

Sau khi chuẩn hoá và định dạng lại các dữ liệu, kiểm tra dữ liệu định dạng mới bằng cách in ra một số hình ảnh đại diện.

```
x_train = x_train / 255.0
x_test = x_test / 255.0
plt.figure(figsize=(18, 9))
for i in range(200):
    plt.subplot(10, 20, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.axis('off')
plt.show()
```



Hình 4: 200 dữ liệu đầu tiên của training dataset sau khi chuẩn hoá

Nhìn vào khoảng 200 bức ảnh đầu tiên, nhóm có một số cảm nhận về hình ảnh của các số trong bộ dữ liệu ban đầu như sau:

- Mỗi hình ảnh các chữ số đều có phong cách viết rất khác nhau và đa dạng, đại diện cho chữ viết tự nhiên của nhiều người khác nhau.
- Một số chữ số được viết rất rõ ràng, hoặc có các đặc trưng riêng biệt để nhận diện được. Trong khi đó một số khác có nét viết mờ, xiêu vẹo, không đều hoặc bị méo mó biến dạng.
- Có nhiều cặp chữ số có thể gây nhầm lẫn với nhau nếu trong hình vẽ, đặc trưng phân biệt là không rõ ràng. Chẳng hạn, số 0 có thể bị nhầm lẫn với số 6 hoặc số 9 nếu phần móc đặc trưng của số 6 và số 9 là không rõ ràng; số 5 và số 6 có thể bị nhầm lẫn nếu như phần uốn cong của số 5 quá nhiều; số 1 và số 7 có thể bị nhầm lẫn do những số này có nhiều phong cách viết khác nhau, cộng với việc hình ảnh chữ viết bị xiêu vẹo khiến cho việc nhận diện dễ có nhầm lẫn.

Trong phần tiếp theo nhóm sẽ phân tích rõ hơn về tương quan của các dữ liệu với nhau.

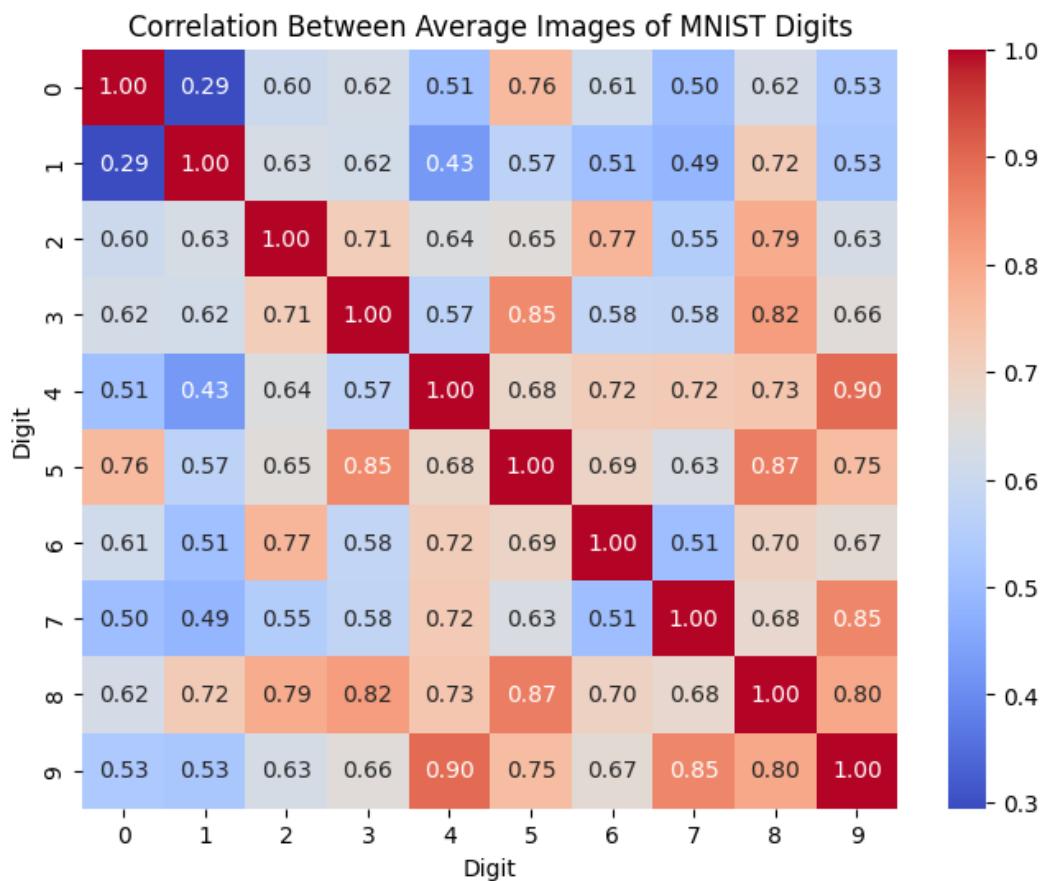
### 2.3.3 Kiểm tra một số dữ liệu có thể khiến mô hình nhầm lẫn

Dựa trên quan sát với tập dữ liệu này, nhóm nhận thấy có thể xuất hiện một số điểm dữ liệu khiến mô hình dự đoán sai lệch do chữ số có cấu trúc hình học tương tự nhau, hoặc do đặc điểm riêng của người viết. Vì vậy cần phải chú ý đến các điểm dữ liệu này, dưới đây là so sánh sự tương quan giữa các số trong tập dữ liệu:

```
mean_images = np.zeros((10, 28, 28))
X_train = x_train.reshape(x_train.shape[0], *(28, 28))
for digit in range(10):
    mean_images[digit] = X_train[y_train == digit].mean(axis=0)

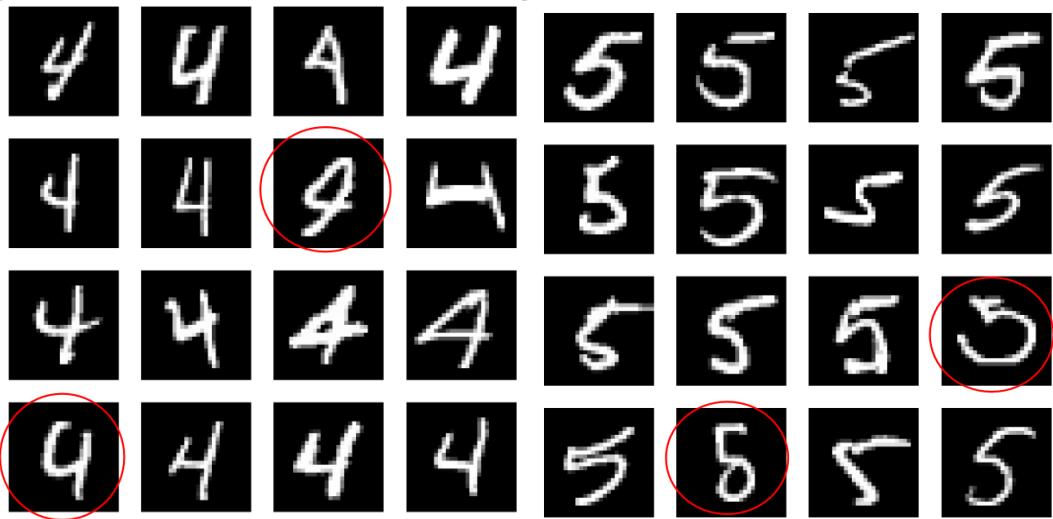
mean_images_flat = mean_images.reshape(10, -1)
corr_matrix = np.corrcoef(mean_images_flat)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', xticklabels=range(10), yticklabels=range(10))
plt.title('Correlation Between Average Images of MNIST Digits')
plt.xlabel('Digit')
plt.ylabel('Digit')
plt.show()
```



Hình 5: Ma trận tương quan giữa các số trong tập dữ liệu

Có thể thấy trong ma trận tương quan trên, số 4 và số 9 giống nhau đến 90%, số 5 và số 8 giống nhau đến 87% điều này gây ra khó khăn cho mô hình khi dự đoán các số có dạng gần giống nhau như vậy. Để chắc chắn, nhóm sẽ in ra một vài mẫu trong tập dữ liệu, các điểm dữ liệu được khoanh đỏ là các điểm dữ liệu dễ gây nhầm lẫn.



Hình 6: Các điểm dữ liệu dễ gây nhầm lẫn cho mô hình

#### 2.3.4 Sử dụng K-means clustering để đánh giá sơ bộ

Trong phần này, nhóm sẽ sử dụng thuật toán K-means clustering (unsupervised learning) để gom cụm các điểm dữ liệu có tính chất tương tự nhau thành từng nhóm từ đó đưa ra đánh giá. Vì đây là thuật toán unsupervised learning nên chỉ cần sử dụng tập `x_train` là đủ. Nhóm sẽ khởi tạo 10 centroid (bằng với số class của dữ liệu), số điểm dữ liệu là 60000.

```
K = 10
N = 60000

X = x_train[np.random.choice(x_train.shape[0], N)]
X = X.reshape(X.shape[0], 784)
kmeans = KMeans(n_clusters=K).fit(X)
pred_label = kmeans.predict(X)
```

Sau khi thực thi đoạn code trên, các centroid được lưu trong biến `kmeans.cluster_centers_`, label của mỗi điểm dữ liệu được lưu trong biến `pred_label`. Kết quả các centroid mà mô hình tìm được thật sự chưa được tốt, các điểm dữ liệu gây nhiễu cho mô hình nếu không được gán nhãn sẽ rất dễ gây nhầm lẫn khiến mô hình khó học được.

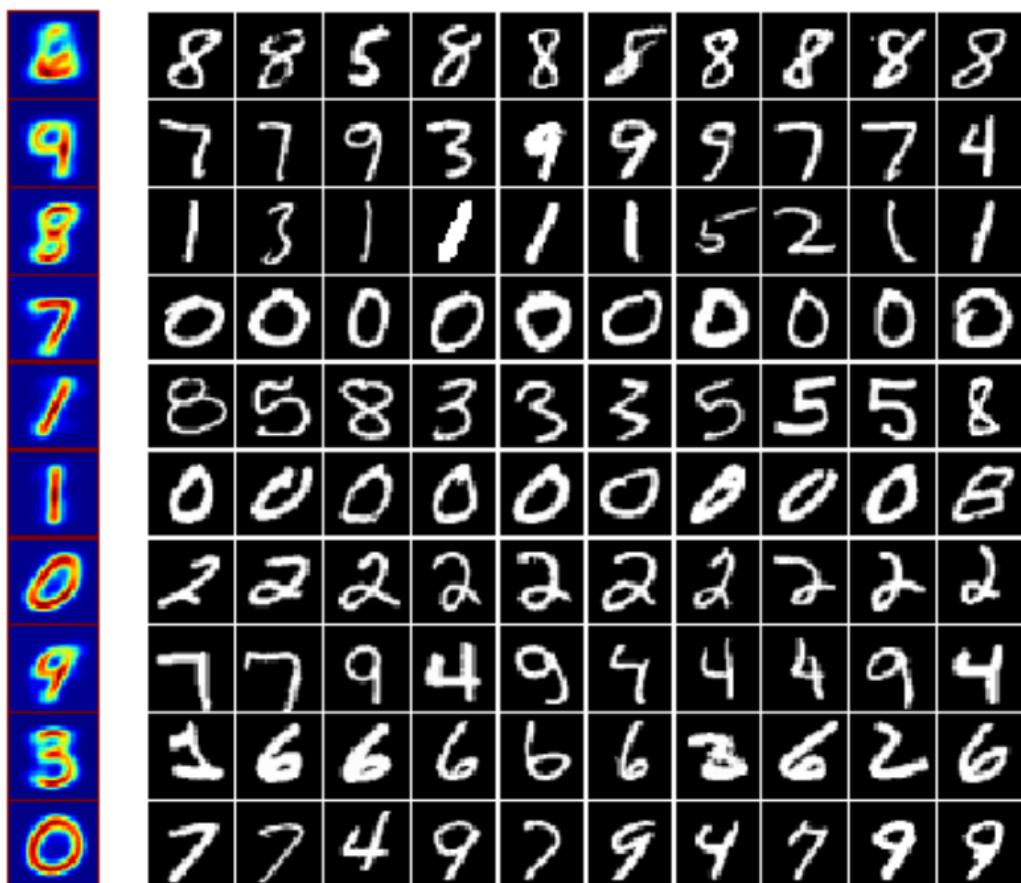
```
plt.axis('off')
A = display_network(kmeans.cluster_centers_.T, 1, 10)
f2 = plt.imshow(A, interpolation='nearest', cmap = 'gray')
plt.show()
```



Hình 7: Các centroid mà K-means tìm được

Tiếp theo, sẽ chọn 20 bức ảnh gần centroid của mỗi cluster nhất, vì càng gần centroid độ tin cậy càng cao. Ta có thể thấy dữ liệu trong mỗi hàng khá giống nhau và giống với centroid ở cột đầu tiên bên trái. Tuy nhiên có một số điểm dữ liệu bị nhầm lẫn như số 4 và 9, 5 và 8,... đã được đề cập do có cấu trúc giống nhau. Các số có cấu trúc càng giống nhau trong ma trận tương quan thì tỉ lệ bị phân nhầm vào cùng một cluster càng cao. Ở đây có hai centroid là số 1 thẳng và số 1 chéo bị nhận diện là hai số khác nhau, tuy nhiên do đây là thuật toán unsupervised không có nhãn nên K-means clustering không nhận biết được. Điều này không ảnh hưởng đến chất lượng của các mô hình supervised learning do đã có nhãn sẽ được nhóm lại thành một.

Từ các phân tích trên, để nhận diện được các chữ số viết tay có độ chính xác cao, ta chỉ cần chú ý đến các số có độ tương quan cao với nhau thì sẽ cải thiện được chất lượng của mô hình.



Hình 8: Các centroid (cột đầu) và 20 điểm gần centroid nhất



### 3 Giải pháp 1: Logistics Regression with Dimensionality Reduction

#### 3.1 Logistics Regression without Principal Component Analysis (PCA)

Ban đầu, nhóm nghiên cứu huấn luyện mô hình Logistic Regression trực tiếp trên tập dữ liệu gốc mà không thực hiện bất kỳ bước tiền xử lý nào. Kết quả cho thấy mô hình đạt được độ chính xác (accuracy) **92.14%**. Tuy nhiên, trong quá trình huấn luyện, nhóm nghiên cứu phát hiện rằng mô hình gặp phải vấn đề không hội tụ trong số lần lặp đã được giới hạn, tức là thuật toán không đạt được điều kiện dừng nhưng đã hết iteration. Điều này dẫn đến thời gian huấn luyện kéo dài đáng kể, lên tới **3 tiếng** khi chạy bằng CPU.

```
mnist = fetch_openml('mnist_784', as_frame=False)
X_train, y_train = mnist.data[:60000], mnist.target[:60000]
X_test, y_test = mnist.data[60000:], mnist.target[60000:]

log_reg = LogisticRegression(random_state=42, max_iter=100000, solver='saga')
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Snippet 1: Train MNIST dataset bằng Logistics Regression.

Nhận thấy thời gian huấn luyện quá lâu, nhóm nghiên cứu quyết định thử nghiệm các kỹ thuật chuẩn hóa dữ liệu nhằm cải thiện hiệu suất của mô hình. Hai phương pháp được sử dụng là:

- **Standard Scaler:** Chuẩn hóa dữ liệu theo phân phối chuẩn, đưa các đặc trưng về dạng có trung bình bằng 0 và độ lệch chuẩn bằng 1.
- **Min-Max Scaler:** Chuẩn hóa dữ liệu theo khoảng giá trị [0,1] bằng cách đưa mỗi đặc trưng về giá trị tối thiểu và tối đa trong tập dữ liệu.

Kết quả cho thấy rằng cả hai phương pháp trên không ảnh hưởng đến độ chính xác của mô hình, vẫn giữ nguyên mức 92.14%. Tuy nhiên, thời gian huấn luyện giảm xuống đáng kể, từ 3 tiếng xuống còn **50 phút**. Khi so sánh hai phương pháp này, nhóm nghiên cứu nhận thấy rằng Min-Max Scaler có tốc độ huấn luyện nhanh hơn, do đó, nhóm nghiên cứu quyết định chọn Min-Max Scaler làm phương pháp tiền xử lý chính trong các bước tiếp theo.

Mặc dù thời gian huấn luyện đã giảm, nhưng mô hình vẫn gặp vấn đề không hội tụ trong giới hạn số lần lặp cho trước. Do đó, nhóm nghiên cứu tiếp tục thử nghiệm phương pháp Cross Validation để tối ưu siêu tham số C của mô hình Logistic Regression. Siêu tham số C trong Logistic Regression kiểm soát regularization, tức là mức độ phạt đối với các trọng số lớn để tránh overfitting. Một giá trị C nhỏ hơn làm tăng mức độ regularization (hạn chế mô hình quá phức tạp), trong khi một giá trị C lớn hơn cho phép mô hình học kỹ hơn từ dữ liệu.

```
print("Using cuML components:")
ScalerClass = cuMinMaxScaler
LogisticRegressionClass = cuLogisticRegression
PipelineClass = cuPipeline
GridSearchCVClass = cuGridSearchCV
# -----
logistic_params = {
    'penalty': 'l2', 'solver': 'qn', 'max_iter': 10000, 'tol': 1e-3
}
grid_search_extra_params = {}
print(f"    Scaler: {ScalerClass}")
```



```
print(f" Pipeline: {PipelineClass}")
print(f" LogisticRegression: {LogisticRegressionClass}")
print(f" GridSearchCV: {GridSearchCVClass}")
pipe_lr = PipelineClass([
    ('scaler', ScalerClass()),
    ('logisticregression', LogisticRegressionClass(**logistic_params))
])

param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

scorer = 'accuracy'

# Instantiate GridSearchCV using the dynamically assigned GridSearchCVClass
grid_search = GridSearchCVClass(
    estimator=pipe_lr,
    param_grid=param_grid,
    cv=5,
    scoring=scorer,
    verbose=1,
    **grid_search_extra_params
)

start_time = time.time()

print(f"GPU Data shapes: X_train={X_train_gpu.shape}, y_train={y_train_gpu.shape}")
print(f"GPU Data types: X_train={X_train_gpu.dtype}, y_train={y_train_gpu.dtype}")

print("\nStarting cuML GridSearchCV on GPU...")
try:
    grid_search.fit(X_train_gpu, y_train_gpu)
    print("cuML GridSearchCV finished.")

except TypeError as e:
    import traceback
    print(f"\nCaught TypeError during grid_search.fit:\n{e}")
    print("\nFull Traceback:")
    traceback.print_exc()
    print("\nIf this persists, it might indicate a deeper issue or version mismatch.")
except Exception as e:
    import traceback
    print(f"\nCaught an unexpected error during grid_search.fit:\n{e}")
    print("\nFull Traceback:")
    traceback.print_exc()
end_time = time.time()
```

Snippet 2: GridSearchCV để tìm ra C tối ưu cho Logistics Regression.

Nhóm nghiên cứu sử dụng GridSearchCV để tìm kiếm giá trị C tối ưu. GridSearchCV giúp thử nghiệm nhiều giá trị của C (0.01, 0.1, 1, 10, 100 và 1000), kết hợp với Cross Validation, nhằm đảm bảo mô hình không bị overfitting hoặc underfitting. Sau khi thử nghiệm nhiều giá trị khác nhau của C, kết quả cho thấy rằng:

- Khi C = 0.1, mô hình đạt độ chính xác cao nhất là **92.58%**.
- Đặc biệt, mô hình đã hội tụ trong số lần lặp cho trước, không còn gặp vấn đề dừng sớm



hoặc chạy quá lâu như trước.

```
default_pipe = make_pipeline(MinMaxScaler(), LogisticRegression(random_state=42,
    solver='saga', C=0.1))
default_pipe.fit(X_train, y_train)
y_pred = default_pipe.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Snippet 3: Logistics Regression với MinMaxScaler và C bằng 0.1.

### 3.2 Logistics Regression with Principal Component Analysis (PCA)

Nhận thấy rằng bộ dataset MNIST không thật sự cần toàn bộ 784 features của nó để có thể phân loại hiệu quả và thời gian huấn luyện của mô hình vẫn còn khá lâu, nhóm nghiên cứu quyết định thử nghiệm phương pháp **Principal Component Analysis (PCA)** để giảm chiều dữ liệu đầu vào nhằm cải thiện hiệu suất mô hình.

Tập dữ liệu MNIST chứa hình ảnh chữ số viết tay trên nền đen với kích thước  $28 \times 28$ , tương ứng với 784 features đầu vào. Tuy nhiên, không phải tất cả các feature này đều quan trọng trong quá trình huấn luyện mô hình. Trong thực tế, nhiều pixel xung quanh chữ số chỉ là nền đen không chứa thông tin, không đóng góp đáng kể vào việc phân loại chữ số. Những feature dư thừa này không chỉ làm tăng kích thước không gian đặc trưng, mà còn làm chậm quá trình huấn luyện và có thể gây overfitting.

Để khắc phục vấn đề này, nhóm nghiên cứu áp dụng PCA, một phương pháp giảm chiều phổ biến giúp trích xuất những đặc trưng quan trọng nhất của dữ liệu, giữ lại những thành phần có phương sai cao nhất và loại bỏ những thành phần ít đóng góp vào tổng thể. Việc sử dụng PCA có thể giúp giảm số lượng feature đầu vào, từ đó:

- Giảm thời gian huấn luyện mà vẫn giữ lại phần lớn thông tin quan trọng.
- Tăng khả năng hội tụ của mô hình bằng cách loại bỏ nhiễu và giảm độ phức tạp.
- Hạn chế overfitting, đặc biệt là khi số lượng thành phần chính được giữ lại hợp lý.

Trước tiên, nhóm nghiên cứu chuyển đổi dữ liệu gốc sang một tập hợp các thành phần chính mới bằng cách áp dụng PCA không giới hạn số lượng thành phần chính:

```
pca = PCA()
X_train_pca = pca.fit_transform(X_train)
```

Snippet 4: Sử dụng PCA trên tập MNIST.

Lệnh trên giúp tìm ra các thành phần chính (principal components) theo thứ tự giảm dần về mức độ đóng góp vào phương sai tổng thể của dữ liệu.

Tiếp theo, để xác định số lượng principal components cần giữ lại, nhóm nghiên cứu vẽ Scree Plot, một biểu đồ thể hiện tỷ lệ phương sai được giải thích bởi từng thành phần chính. Điều này giúp nhóm nghiên cứu hiểu được bao nhiêu thành phần quan trọng và bao nhiêu thành phần có thể loại bỏ mà không ảnh hưởng đáng kể đến hiệu suất mô hình.

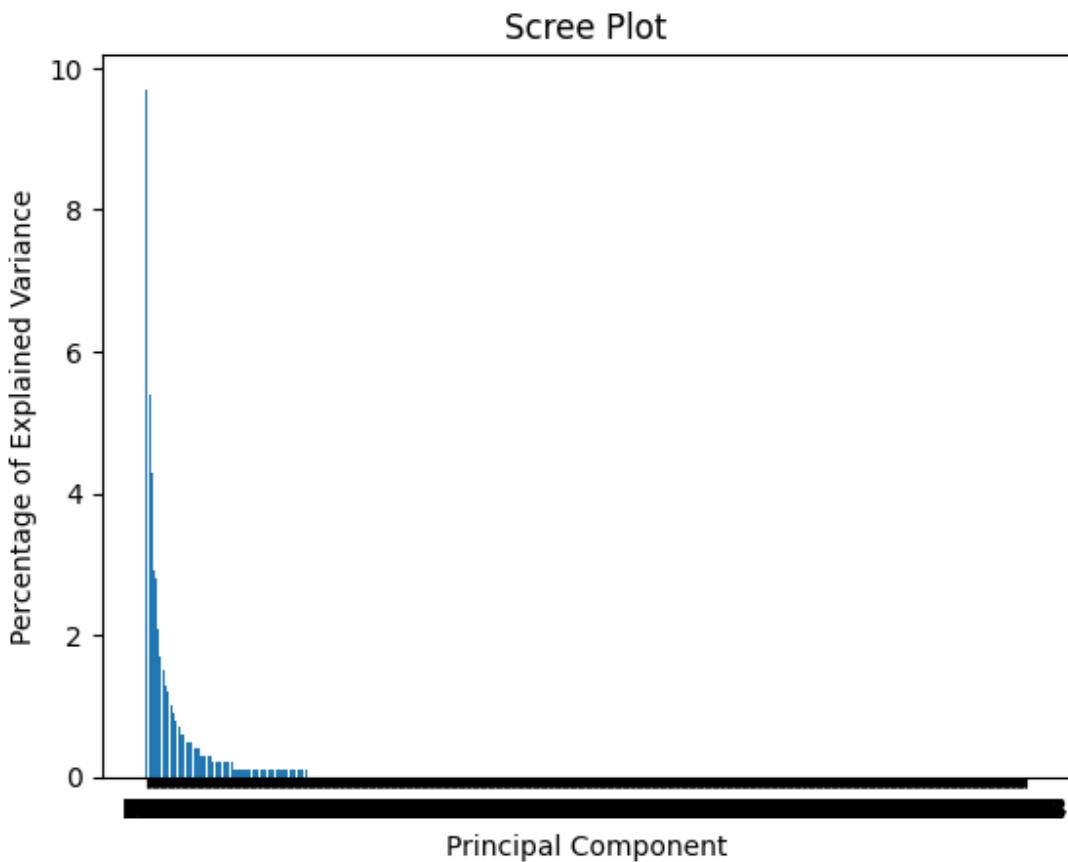
```
per_var = np.round(pca.explained_variance_ratio_ * 100, decimals=1)
labels = ['PC' + str(x) for x in range(1, len(per_var) + 1)]

plt.bar(x=range(1, len(per_var) + 1), height=per_var, tick_label=labels)
plt.ylabel('Percentage of Explained Variance')
```

```
plt.xlabel('Principal Component')
plt.title('Scree Plot')
plt.show()
```

Snippet 5: Scree Plot sau khi áp dụng PCA lên tập MNIST.

Hình ảnh Scree Plot thu được cho thấy rằng chỉ khoảng 1/5 số lượng feature (tức là khoảng 150 - 200 thành phần chính) có đóng góp đáng kể vào phương sai tổng thể. Phần còn lại có phương sai rất nhỏ, có thể bị loại bỏ mà không làm giảm đáng kể độ chính xác của mô hình.



Hình 9: Scree Plot cụ thể.

Dựa trên Scree Plot, chúng ta cần lựa chọn số lượng thành phần chính PCA Components sao cho:

- Tỷ lệ phương sai tích lũy (cumulative explained variance ratio) đạt ít nhất 95%, đảm bảo phần lớn thông tin được giữ lại.
- Số lượng thành phần giữ lại không quá lớn, tránh làm tăng chi phí tính toán không cần thiết.



Nhóm áp dụng đoạn code sau để thực hiện PCA và huấn luyện mô hình:

```
# Initialize a PCA and reduce the dimension of the original dataset
pca = PCA(n_components=0.95) # We keep 95% of variance
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train a classifier (Logistic Regression)
pca_clf = LogisticRegression(max_iter=100000, random_state=42, solver='saga')

pca_clf.fit(X_train_pca, y_train)

# Predict and compute accuracy
y_pred = pca_clf.predict(X_test_pca)
accuracy = accuracy_score(y_test, y_pred)
```

Snippet 6: Thực hiện giữ lại 95% variance của tập MNIST và bắt đầu train.

Sau khi chạy đoạn code trên, nhóm nghiên cứu thu được kết quả:

- Số lượng principal components được giữ lại là 154, đúng với dự đoán ban đầu từ Scree Plot, khi chỉ khoảng 1/5 số feature gốc (784) thực sự ảnh hưởng đến kết quả cuối cùng.
- Độ chính xác sau khi huấn luyện Logistic Regression trên tập dữ liệu đã giảm chiều là **92.04%**.

Kết quả này hoàn toàn hợp lý, vì dù số chiều dữ liệu đã giảm đi khoảng 5 lần nhưng độ chính xác chỉ giảm một lượng nhỏ so với trước đó (92.58%). Điều này chứng tỏ rằng nhiều feature trong tập dữ liệu gốc không đóng góp đáng kể vào quá trình phân loại, và PCA đã giúp loại bỏ những phần dư thừa mà không mất nhiều thông tin quan trọng.

Ngoài ra, thời gian huấn luyện giảm đáng kể so với mô hình ban đầu do số chiều dữ liệu nhỏ hơn. Điều này khẳng định rằng PCA là một phương pháp hiệu quả để giảm độ phức tạp của mô hình, đồng thời giữ nguyên hiệu suất phân loại ở mức cao.

## 4 Giải pháp 2: Support Vector Machine

Việc kết hợp PCA và Logistics Regression đem lại hiệu quả đáng kể về thời gian huấn luyện mà vẫn giữ được độ chính xác ở mức cao. Tuy nhiên khi áp dụng vào thực tế, phương pháp kết hợp này vẫn chưa thật sự có ý nghĩa, bởi mức chính xác chỉ xấp xỉ 92%, một tỷ lệ còn khiêm tốn. Bên cạnh đó, việc sử dụng PCA để giảm chiều dữ liệu đầu vào tồn tại cả hai mặt lợi và hại, điểm lợi như đã đề cập ở phần nghiên cứu 1.2, trong khi đó mặt hại nằm ở cách mô hình giảm chiều các dữ liệu mới. PCA thể hiện sự hiệu quả trên tập huấn luyện vì tập dữ liệu chỉ gồm các kí tự số màu trắng trên nền đen, trong khi thực tế, ảnh chụp các kí tự sẽ đa dạng và nhiều nhiễu, việc giảm chiều bằng PCA vô tình đánh mất các feature cần thiết để phân biệt với nhiễu. Ngoài ra, tập dữ liệu chữ số của MNIST từ chữ viết của con người với độ phức tạp cao và sự chồng chéo feature, dẫn đến mang bản chất phi tuyến, phương pháp Logistics Regression vốn tạo mối quan hệ tuyến tính giữa feature và label khó có thể biểu diễn tốt tập dữ liệu MNIST.

Cùng vì vậy, nhóm tác giả tiếp cận bài toán bằng một phương pháp khác (Support Vector Machine) nhằm khắc phục các hạn chế trên. Support Vector Machine (SVM) là một mô hình học máy mạnh mẽ và linh hoạt trong nhiều tác vụ, và đặc biệt là bài toán phân loại. Trong quá trình huấn luyện, SVM giữ được các feature mà tốc độ hội tụ vẫn nhanh. Đối với dữ liệu phi tuyến, các Kernel được áp dụng với SVM có thể giải quyết tốt. SVM hỗ trợ nhiều Kernel như



Linear Kernel, Polynomial Kernel, Gaussian RBF Kernel, ... trong đó Gaussian RBF Kernel là lựa chọn tối ưu bởi:

- RBF Kernel tập trung vào các vùng cục bộ của không gian feature, từ đó xử lý những thay đổi cục bộ một cách khéo léo mà không làm overfitting toàn bộ không gian.
- Mặc dù đều là mô hình phi tuyến tính, so với Polynomial Kernel, Gaussian RBF Kernel tránh được việc overfitting và đưa ra các decision boundary mượt mà, không quá phức tạp.

Tuy nhiên, để xác định rõ liệu Gaussian RBF Kernel tối ưu hơn khi so với Polynomial Kernel, hay kể cả Linear Kernel, chúng ta cần phải chứng minh bằng thực nghiệm. Với từng Kernel, Nhóm tác giả sử dụng Grid Search kết hợp Cross Validation để chọn ra bộ tham số tốt nhất trong giới hạn xác định. Nhóm tác giả cũng nhận thấy việc sử dụng CPU cho quá trình huấn luyện, hay Grid Search sẽ vô cùng lâu và kém hiệu quả. Từ đây, nhóm quyết định sử dụng GPU bằng thư viện ThunderSVM [2] để tăng tốc tiến trình huấn luyện và tìm kiếm tham số tốt nhất. Tương tự với cách tiếp cận 1, cách tiếp cận 2 cũng xử lý tập dữ liệu bằng MinMaxScaler trước khi tiến hành huấn luyện.

#### 4.1 Support Vector Machine Classification with Linear Kernel

Với tham số C, nhóm thử các giá trị 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, và tiến hành GridSearch, Cross Validation 3 lần với mỗi giá trị.

```
from sklearn.model_selection import GridSearchCV
from thundersvm import SVC

# Define hyperparameter grid for Linear SVM
param_grid_linear = {'C': np.logspace(-3, 3, 7)}

# Perform Grid Search for Linear SVM
grid_linear = GridSearchCV(SVC(kernel='linear'), param_grid_linear, cv=3, scoring='accuracy', verbose=3)
grid_linear.fit(X_train, y_train)
best_curr_C_linear = grid_linear.best_params_['C']

# Print best parameters
print("Best Parameters:", grid_linear.best_params_)
print("Best Accuracy:", grid_linear.best_score_)
```

Snippet 7: Thực hiện SVM với Linear Kernel.

Kết quả thu được cho thấy tham số C càng lớn tốc độ huấn luyện càng tăng, lý do bởi khi C tăng, mô hình SVM ưu tiên giảm thiểu training error đồng nghĩa tăng độ khó của việc tối ưu. Điều này dẫn đến cần nhiều support vector hơn, margin hẹp hơn và hội tụ chậm hơn, khiến thời gian đào tạo tăng nhanh. Ta có thể thấy độ chính xác tăng dần khi C tăng đến 0.1 và sau đó bắt đầu giảm, rõ ràng **C = 0.1** mang lại sự cân bằng tốt nhất giữa độ chính xác và hiệu quả. Với **C = 0.1**, mô hình đạt độ chính xác **94.69%** với tập dữ liệu test, cao hơn khi sử dụng PCA kết hợp Logistic Regression.

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
[CV 1/3] END ..... C=0.001;, score=0.921 total time= 26.9s
[CV 2/3] END ..... C=0.001;, score=0.917 total time= 24.1s
[CV 3/3] END ..... C=0.001;, score=0.921 total time= 23.9s
[CV 1/3] END ..... C=0.01;, score=0.937 total time= 19.4s
[CV 2/3] END ..... C=0.01;, score=0.935 total time= 17.9s
[CV 3/3] END ..... C=0.01;, score=0.938 total time= 18.4s
```



```
[CV 1/3] END .....C=0.0999999999999999;, score=0.940 total time= 20.0s
[CV 2/3] END .....C=0.0999999999999999;, score=0.938 total time= 23.2s
[CV 3/3] END .....C=0.0999999999999999;, score=0.941 total time= 23.4s
[CV 1/3] END .....C=1.0;, score=0.931 total time= 37.6s
[CV 2/3] END .....C=1.0;, score=0.932 total time= 36.9s
[CV 3/3] END .....C=1.0;, score=0.933 total time= 39.1s
[CV 1/3] END .....C=10.0;, score=0.919 total time= 2.1min
[CV 2/3] END .....C=10.0;, score=0.920 total time= 2.1min
[CV 3/3] END .....C=10.0;, score=0.921 total time= 1.8min
[CV 1/3] END .....C=100.0;, score=0.914 total time=11.6min
[CV 2/3] END .....C=100.0;, score=0.916 total time=13.1min
[CV 3/3] END .....C=100.0;, score=0.913 total time= 9.5min
[CV 1/3] END .....C=1000.0;, score=0.911 total time=81.7min
[CV 2/3] END .....C=1000.0;, score=0.911 total time=103.4min
[CV 3/3] END .....C=1000.0;, score=0.911 total time=67.5min
Best Parameters: {'C': 0.0999999999999999}
Best Accuracy: 0.9398666666666667
```

Snippet 8: Kết quả SVM với Linear Kernel.

## 4.2 Support Vector Machine Classification with Polynomial Kernel

Dối với Polynomial Kernel, chúng ta cần chọn bộ 3 tham số C, degree, coef0 tốt nhất, với C có các giá trị [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0], degree có các giá trị [2, 3, 4, 5], và coef0 với các giá trị [0, 0.5, 1, 2]. Tiến hành GridSearch, Cross Validation 3 lần với mỗi tổ hợp giá trị tham số C, degree, coef0 được cung cấp.

```
from sklearn.model_selection import GridSearchCV
from thundersvm import SVC

# Define hyperparameter grid for Polynomial Kernel SVM
param_grid_poly = {
    'C': np.logspace(-3, 3, 7),
    'degree': [2, 3, 4, 5],
    'coef0': [0, 0.5, 1, 2]
}

# Perform Grid Search for Polynomial Kernel SVM
grid_poly = GridSearchCV(SVC(kernel='polynomial')), param_grid_poly, cv=3, scoring='accuracy', verbose=3)
grid_poly.fit(X_train, y_train)

# Print best parameters
print("Best Parameters:", grid_poly.best_params_)
print("Best Accuracy:", grid_poly.best_score_)
```

Snippet 9: Thực hiện SVM với Polynomial Kernel.

Kết quả cho thấy với C nhỏ, mô hình bị underfitting kể cả khi tăng degree vì giá trị C đã ngăn các mô hình phức tạp. Giá trị coef0 tăng cường ảnh hưởng của các số hạng bậc thấp hơn (như hằng số và mô hình tuyến tính), giúp tăng độ chính xác nhưng vẫn còn hạn chế. Khi C mang các giá trị lớn hơn 1.0, mô hình không còn phụ thuộc nhiều vào coef0 vì mô hình đã được cho phép học phức tạp hơn, song chênh lệch độ chính xác khi thay đổi degree không nhiều, thậm chí có vài trường hợp degree tăng làm giảm đi đáng kể độ chính xác. Lí do vì degree càng lớn mô hình càng phức tạp và càng nhiều, trong khi tập dữ liệu không có lợi từ các mối quan hệ bậc cao. Bên cạnh đó, với C có giá trị lớn, độ chính xác giữa các mô hình không biến đổi nhiều khi



thay đổi các yếu tố như degree, coef0. Bộ tham số tốt nhất thu được là  $C = 1000$  và  $\text{degree} = 2$ , giá trị coef0 có thể loại bỏ, đối với tập dữ liệu test, mô hình đạt độ chính xác **98.03%**, cao vượt trội so với các phương pháp trước. Điều này xác nhận mô hình phi tuyến tính thích hợp với tập dữ liệu MNIST hơn so với các mô hình tuyến tính.

```
Fitting 3 folds for each of 112 candidates, totalling 336 fits
.....
<hidden combinations of another C, coef0, degree>
.....
[CV 1/3] END ..... C=0.1, coef0=0, degree=2;, score=0.502 total time= 29.7s
[CV 2/3] END ..... C=0.1, coef0=0, degree=2;, score=0.510 total time= 30.1s
[CV 3/3] END ..... C=0.1, coef0=0, degree=2;, score=0.543 total time= 29.8s
[CV 1/3] END ..... C=0.1, coef0=0, degree=3;, score=0.092 total time= 32.2s
[CV 2/3] END ..... C=0.1, coef0=0, degree=3;, score=0.092 total time= 31.2s
[CV 3/3] END ..... C=0.1, coef0=0, degree=3;, score=0.095 total time= 31.2s
[CV 1/3] END ..... C=0.1, coef0=0, degree=4;, score=0.089 total time= 31.6s
[CV 2/3] END ..... C=0.1, coef0=0, degree=4;, score=0.091 total time= 30.7s
[CV 3/3] END ..... C=0.1, coef0=0, degree=4;, score=0.091 total time= 31.0s
[CV 1/3] END ..... C=0.1, coef0=0, degree=5;, score=0.089 total time= 30.3s
[CV 2/3] END ..... C=0.1, coef0=0, degree=5;, score=0.091 total time= 30.2s
[CV 3/3] END ..... C=0.1, coef0=0, degree=5;, score=0.091 total time= 30.2s
.....
[CV 1/3] END ..... C=0.1, coef0=1, degree=2;, score=0.904 total time= 20.5s
[CV 2/3] END ..... C=0.1, coef0=1, degree=2;, score=0.900 total time= 20.6s
[CV 3/3] END ..... C=0.1, coef0=1, degree=2;, score=0.905 total time= 20.2s
[CV 1/3] END ..... C=0.1, coef0=1, degree=3;, score=0.914 total time= 19.2s
[CV 2/3] END ..... C=0.1, coef0=1, degree=3;, score=0.910 total time= 19.0s
[CV 3/3] END ..... C=0.1, coef0=1, degree=3;, score=0.915 total time= 19.0s
[CV 1/3] END ..... C=0.1, coef0=1, degree=4;, score=0.922 total time= 18.9s
[CV 2/3] END ..... C=0.1, coef0=1, degree=4;, score=0.917 total time= 18.2s
[CV 3/3] END ..... C=0.1, coef0=1, degree=4;, score=0.922 total time= 18.7s
.....
<hidden combinations of another C, coef0, degree>
.....
[CV 1/3] END ..... C=1000.0, coef0=0, degree=2;, score=0.978 total time= 15.5s
[CV 2/3] END ..... C=1000.0, coef0=0, degree=2;, score=0.975 total time= 15.5s
[CV 3/3] END ..... C=1000.0, coef0=0, degree=2;, score=0.976 total time= 16.2s
[CV 1/3] END ..... C=1000.0, coef0=0, degree=3;, score=0.972 total time= 15.1s
[CV 2/3] END ..... C=1000.0, coef0=0, degree=3;, score=0.970 total time= 15.2s
[CV 3/3] END ..... C=1000.0, coef0=0, degree=3;, score=0.972 total time= 15.1s
[CV 1/3] END ..... C=1000.0, coef0=0, degree=4;, score=0.915 total time= 18.3s
[CV 2/3] END ..... C=1000.0, coef0=0, degree=4;, score=0.914 total time= 18.0s
[CV 3/3] END ..... C=1000.0, coef0=0, degree=4;, score=0.918 total time= 18.4s
[CV 1/3] END ..... C=1000.0, coef0=0, degree=5;, score=0.680 total time= 24.7s
[CV 2/3] END ..... C=1000.0, coef0=0, degree=5;, score=0.676 total time= 24.4s
[CV 3/3] END ..... C=1000.0, coef0=0, degree=5;, score=0.676 total time= 24.4s
[CV 1/3] END ..... C=1000.0, coef0=0.5, degree=2;, score=0.967 total time= 20.3s
[CV 2/3] END ..... C=1000.0, coef0=0.5, degree=2;, score=0.965 total time= 19.9s
[CV 3/3] END ..... C=1000.0, coef0=0.5, degree=2;, score=0.967 total time= 20.2s
[CV 1/3] END ..... C=1000.0, coef0=0.5, degree=3;, score=0.972 total time= 18.9s
[CV 2/3] END ..... C=1000.0, coef0=0.5, degree=3;, score=0.969 total time= 17.7s
[CV 3/3] END ..... C=1000.0, coef0=0.5, degree=3;, score=0.970 total time= 18.7s
[CV 1/3] END ..... C=1000.0, coef0=0.5, degree=4;, score=0.974 total time= 16.9s
[CV 2/3] END ..... C=1000.0, coef0=0.5, degree=4;, score=0.971 total time= 17.4s
[CV 3/3] END ..... C=1000.0, coef0=0.5, degree=4;, score=0.972 total time= 16.8s
[CV 1/3] END ..... C=1000.0, coef0=0.5, degree=5;, score=0.975 total time= 16.5s
[CV 2/3] END ..... C=1000.0, coef0=0.5, degree=5;, score=0.972 total time= 17.0s
[CV 3/3] END ..... C=1000.0, coef0=0.5, degree=5;, score=0.973 total time= 16.4s
[CV 1/3] END ..... C=1000.0, coef0=1, degree=2;, score=0.961 total time= 23.6s
[CV 2/3] END ..... C=1000.0, coef0=1, degree=2;, score=0.961 total time= 23.1s
[CV 3/3] END ..... C=1000.0, coef0=1, degree=2;, score=0.962 total time= 23.4s
```



```
[CV 1/3] END .....C=1000.0, coef0=1, degree=3;, score=0.967 total time= 21.0s
[CV 2/3] END .....C=1000.0, coef0=1, degree=3;, score=0.966 total time= 19.7s
[CV 3/3] END .....C=1000.0, coef0=1, degree=3;, score=0.966 total time= 19.9s
[CV 1/3] END .....C=1000.0, coef0=1, degree=4;, score=0.970 total time= 19.5s
[CV 2/3] END .....C=1000.0, coef0=1, degree=4;, score=0.968 total time= 18.4s
[CV 3/3] END .....C=1000.0, coef0=1, degree=4;, score=0.969 total time= 19.0s
[CV 1/3] END .....C=1000.0, coef0=1, degree=5;, score=0.972 total time= 17.7s
[CV 2/3] END .....C=1000.0, coef0=1, degree=5;, score=0.969 total time= 18.3s
[CV 3/3] END .....C=1000.0, coef0=1, degree=5;, score=0.970 total time= 17.5s
[CV 1/3] END .....C=1000.0, coef0=2, degree=2;, score=0.955 total time= 30.0s
[CV 2/3] END .....C=1000.0, coef0=2, degree=2;, score=0.954 total time= 28.4s
[CV 3/3] END .....C=1000.0, coef0=2, degree=2;, score=0.957 total time= 28.4s
[CV 1/3] END .....C=1000.0, coef0=2, degree=3;, score=0.960 total time= 23.6s
[CV 2/3] END .....C=1000.0, coef0=2, degree=3;, score=0.961 total time= 23.3s
[CV 3/3] END .....C=1000.0, coef0=2, degree=3;, score=0.961 total time= 23.1s
[CV 1/3] END .....C=1000.0, coef0=2, degree=4;, score=0.964 total time= 21.2s
[CV 2/3] END .....C=1000.0, coef0=2, degree=4;, score=0.964 total time= 21.0s
[CV 3/3] END .....C=1000.0, coef0=2, degree=4;, score=0.965 total time= 21.6s
[CV 1/3] END .....C=1000.0, coef0=2, degree=5;, score=0.967 total time= 19.8s
[CV 2/3] END .....C=1000.0, coef0=2, degree=5;, score=0.966 total time= 19.8s
[CV 3/3] END .....C=1000.0, coef0=2, degree=5;, score=0.966 total time= 20.3s
Best Parameters: {'C': np.float64(1000.0), 'coef0': 0, 'degree': 2}
Best Accuracy: 0.9763999999999999
```

Snippet 10: Kết quả SVM với Polynomial Kernel.

### 4.3 Support Vector Machine Classification with Gaussian RBF Kernel

Ít hơn so với Polynomial Kernel, chúng ta chỉ cần chọn bộ 2 tham số C, gamma tốt nhất, với C, gamma có các giá trị trong danh sách sau [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]. Tiến hành GridSearch, Cross Validation 3 lần với mỗi tổ hợp giá trị tham số C, gamma được cung cấp.

```
from sklearn.model_selection import GridSearchCV
from thundersvm import SVC

# Define parameter grid
param_grid_rbf = {
    'C': np.logspace(-3, 3, 7),
    'gamma': np.logspace(-3, 3, 7)
}

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(SVC(kernel='rbf'), param_grid_rbf, cv=3, scoring='accuracy', verbose=3)
grid_search.fit(X_train, y_train)

# Print best parameters
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

Snippet 11: Thực hiện SVM với Gaussian RBF Kernel.

Tương tự với Polynomial Kernel, với giá trị C nhỏ, mô hình bị underfitting vì vẫn còn đơn giản, chưa tổng quát hóa được tập dữ liệu, gamma chưa thật sự đóng góp đáng kể. Khi giá trị của C vượt 0.01, độ chính xác của mô hình được cải thiện rõ rệt, gamma tác động rõ rệt tới mô hình vì gamma kiểm soát mức độ ảnh hưởng của từng điểm dữ liệu đến hình dạng của decision boundary. Lúc này, giá trị gamma càng cao, decision boundary trở nên bất thường hơn, dao động



xung quanh các trường hợp riêng lẻ, không tìm được một phương án tối ưu do đó độ chính xác càng giảm. Mô hình đạt độ chính xác cao nhất với tham số **C = 100** và **gamma = 0.01**, trên tập dữ liệu test, độ chính xác là **98.23%** cao hơn so với các phương pháp đã đề cập trước đó.

```
Fitting 3 folds for each of 49 candidates, totalling 147 fits
....
<hidden some combinations of C, gamma>
....
[CV 1/3] END .....C=0.001, gamma=0.001;, score=0.090 total time= 33.6s
[CV 2/3] END .....C=0.001, gamma=0.001;, score=0.090 total time= 35.6s
[CV 3/3] END .....C=0.001, gamma=0.001;, score=0.090 total time= 36.5s
[CV 1/3] END .....C=0.001, gamma=0.01;, score=0.248 total time= 33.0s
[CV 2/3] END .....C=0.001, gamma=0.01;, score=0.202 total time= 33.2s
[CV 3/3] END .....C=0.001, gamma=0.01;, score=0.191 total time= 33.0s
....
<hidden some combinations of C, gamma>
....
[CV 1/3] END .....C=1.0, gamma=0.001;, score=0.932 total time= 17.1s
[CV 2/3] END .....C=1.0, gamma=0.001;, score=0.929 total time= 17.0s
[CV 3/3] END .....C=1.0, gamma=0.001;, score=0.933 total time= 17.2s
[CV 1/3] END .....C=1.0, gamma=0.01;, score=0.973 total time= 15.8s
[CV 2/3] END .....C=1.0, gamma=0.01;, score=0.970 total time= 15.8s
[CV 3/3] END .....C=1.0, gamma=0.01;, score=0.972 total time= 15.7s
[CV 1/3] END ..C=1.0, gamma=0.0999999999999999;, score=0.936 total time= 51.5s
[CV 2/3] END ..C=1.0, gamma=0.0999999999999999;, score=0.936 total time= 51.0s
[CV 3/3] END ..C=1.0, gamma=0.0999999999999999;, score=0.942 total time= 51.8s
....
<hidden some combinations of C, gamma>
....
[CV 1/3] END .....C=10.0, gamma=0.001;, score=0.952 total time= 16.1s
[CV 2/3] END .....C=10.0, gamma=0.001;, score=0.950 total time= 16.1s
[CV 3/3] END .....C=10.0, gamma=0.001;, score=0.952 total time= 16.1s
[CV 1/3] END .....C=10.0, gamma=0.01;, score=0.981 total time= 16.6s
[CV 2/3] END .....C=10.0, gamma=0.01;, score=0.977 total time= 16.5s
[CV 3/3] END .....C=10.0, gamma=0.01;, score=0.979 total time= 16.6s
[CV 1/3] END .C=10.0, gamma=0.0999999999999999;, score=0.938 total time= 52.2s
[CV 2/3] END .C=10.0, gamma=0.0999999999999999;, score=0.939 total time= 51.9s
[CV 3/3] END .C=10.0, gamma=0.0999999999999999;, score=0.945 total time= 52.4s
....
<hidden some combinations of C, gamma>
....
[CV 1/3] END .....C=100.0, gamma=0.001;, score=0.966 total time= 18.4s
[CV 2/3] END .....C=100.0, gamma=0.001;, score=0.963 total time= 18.3s
[CV 3/3] END .....C=100.0, gamma=0.001;, score=0.965 total time= 18.3s
[CV 1/3] END .....C=100.0, gamma=0.01;, score=0.981 total time= 16.4s
[CV 2/3] END .....C=100.0, gamma=0.01;, score=0.978 total time= 16.5s
[CV 3/3] END .....C=100.0, gamma=0.01;, score=0.979 total time= 16.5s
[CV 1/3] END C=100.0, gamma=0.0999999999999999;, score=0.938 total time= 52.2s
[CV 2/3] END C=100.0, gamma=0.0999999999999999;, score=0.939 total time= 51.8s
[CV 3/3] END C=100.0, gamma=0.0999999999999999;, score=0.945 total time= 52.4s
....
<hidden some combinations of C, gamma>
....
[CV 1/3] END .....C=1000.0, gamma=0.001;, score=0.967 total time= 21.1s
[CV 2/3] END .....C=1000.0, gamma=0.001;, score=0.964 total time= 20.9s
[CV 3/3] END .....C=1000.0, gamma=0.001;, score=0.965 total time= 21.1s
[CV 1/3] END .....C=1000.0, gamma=0.01;, score=0.981 total time= 16.5s
[CV 2/3] END .....C=1000.0, gamma=0.01;, score=0.978 total time= 16.6s
[CV 3/3] END .....C=1000.0, gamma=0.01;, score=0.979 total time= 16.5s
[CV 1/3] END C=1000.0, gamma=0.0999999999999999;, score=0.938 total time= 52.1s
[CV 2/3] END C=1000.0, gamma=0.0999999999999999;, score=0.939 total time= 51.8s
[CV 3/3] END C=1000.0, gamma=0.0999999999999999;, score=0.945 total time= 52.5s
```



```
.....  
<hidden some combinations of C, gamma>  
.....  
Best Parameters: {'C': 100.0, 'gamma': 0.01}  
Best Accuracy: 0.9792333333333333
```

Snippet 12: Kết quả SVM với Gaussian RBF Kernel.

Về độ chính xác, cách tiếp cận sử dụng SVM kết hợp Kernel đạt hiệu quả cao hơn khi so với Logistic Regression kết hợp PCA, bởi khả năng giữ lại các features và tăng chiều không gian để phân loại. Trong đó, SVM với các Kernel phi tuyến tính như Polynomial và Gaussian RBF đạt độ chính xác cao nhất (**98.03%** và **98.23%**), thể hiện rằng các mô hình này thích hợp với tập dữ liệu phi tuyến như MNIST. Chúng ta có thể thấy sử dụng Gaussian RBF Kernel có độ chính xác cao hơn so với Polynomial, với bộ tham số ít phức tạp hơn (**C = 100, gamma = 0.01** so với **C = 1000, degree = 2**), cũng khẳng định sự tối ưu của Gaussian RBF Kernel trong bài toán phân loại chữ số viết tay của MNIST.

## 5 Mở rộng - Mạng nơ ron tích chập

Convolutional Neural Networks (CNN) là một trong những kiến trúc học sâu hiệu quả nhất cho các bài toán xử lý ảnh, đặc biệt là phân loại ảnh chữ số viết tay trong tập dữ liệu MNIST. Tập dữ liệu MNIST gồm 60,000 ảnh huấn luyện và 10,000 ảnh kiểm tra, mỗi ảnh là một chữ số từ 0 đến 9, với kích thước 28x28 pixel. Với độ phức tạp vừa phải và tính ứng dụng cao, MNIST trở thành chuẩn đánh giá phổ biến cho các mô hình học sâu trong thị giác máy tính.

CNN hoạt động hiệu quả nhờ khả năng học các đặc trưng không gian qua các tầng tích chập (convolutional layers), tầng gộp (pooling layers) và các tầng phi tuyến (activation layers). Không giống như các mô hình học máy truyền thống, CNN không cần đặc trưng đầu vào được trích xuất thủ công mà có thể tự học từ dữ liệu ảnh, giúp cải thiện đáng kể độ chính xác.

Trong nhiều năm qua, các mô hình CNN hiện đại đã đạt được những kết quả vượt trội trên tập MNIST:

- **Branching/Merging CNN + Homogeneous Vector Capsules** đạt 99.87% độ chính xác (sai số 0.13%) nhờ sử dụng cấu trúc phân nhánh và kết hợp với vector capsules mà không cần routing, giúp đơn giản hóa mô hình mà vẫn đạt hiệu suất cao.
- **Efficient-CapsNet** đạt 99.84% độ chính xác với số lượng tham số cực kỳ nhỏ gọn (chỉ khoảng 160,000 tham số), nhờ tích hợp attention vào mạng Capsule, giúp mô hình nhẹ hơn nhưng vẫn mạnh mẽ.
- **EnsNet** ứng dụng học tổ hợp (ensemble learning) trong CNN và các mạng con fully connected, mang lại độ chính xác 99.84%.
- Các mô hình như **SOPCNN**, **RMDL** và **R-ExplaiNet-22** cũng đạt độ chính xác từ 99.80% trở lên, mỗi mô hình theo đuổi một hướng tiếp cận riêng như tối giản kiến trúc, sử dụng nhiều mô hình con, hoặc hướng tới khả năng giải thích (explainability).

Những kết quả trên cho thấy CNN không chỉ có khả năng đạt độ chính xác rất cao trong các bài toán phân loại cơ bản như MNIST, mà còn liên tục được cải tiến để trở nên nhẹ hơn, dễ huấn luyện hơn, và dễ hiểu hơn. Chính vì vậy, CNN không chỉ là lựa chọn hàng đầu cho MNIST mà còn đóng vai trò nền tảng trong nhiều ứng dụng học sâu hiện đại như nhận diện khuôn mặt, phát hiện vật thể và xe tự hành.



Nhóm nghiên cứu quyết định mở rộng vấn đề bằng việc xây dựng mạng nơ-ron tích chập đơn giản để phân loại ảnh chữ số viết tay MNIST, nhằm kiểm tra tính hiệu quả của mô hình học sâu này. Kết quả cho thấy mặc dù chỉ là một mô hình CNN đơn giản, độ chính xác trong tác vụ phân loại ảnh chữ số viết tay đạt hiệu quả đáng kể.

## 5.1 Xây dựng mô hình mạng nơ-ron tích chập

Sau khi chuẩn hoá, định dạng và kiểm tra dữ liệu, ta bắt đầu xây dựng mô hình mạng nơ-ron tích chập **model** với các method và class như sau:

- **Input:** Khởi tạo một tensor Keras, là một đối tượng giống tensor tương trưng, mà chúng ta bổ sung một số thuộc tính nhất định cho phép chúng ta xây dựng một mô hình Keras chỉ bằng cách biết các đầu vào và đầu ra của mô hình.
- **Conv2D:** tạo ra một **kernel** với kích thước xác định, được tích chập với đầu vào của input layer để tạo ra một tensor của đầu ra, với **filter** là một số nguyên dương đại diện cho số bộ lọc trong phép tích chập hay số chiều của không gian đầu ra, **activation** là hàm kích hoạt được áp dụng cho output.
- **MaxPool2D:** áp dụng max pooling cho mỗi vùng **pool\_size** cho mỗi channel của đầu vào. Vùng áp dụng max pooling được dịch chuyển theo từng bước dọc theo mỗi chiều.
- **Dropout:** là một kỹ thuật được sử dụng trong các mạng neural để tránh overfitting trên tập huấn luyện bằng cách loại bỏ các nơ-ron (neural) với xác suất **rate** > 0. Nó giúp mô hình không bị phụ thuộc quá nhiều vào một tập thuộc tính nào đó. Các input không được đặt thành 0 được tăng tỷ lệ lên  $\frac{1}{1-\text{rate}}$  sao cho tổng trên tất cả các input là không đổi.
- **Flatten:** làm phẳng dữ liệu đầu vào và không ảnh hưởng đến **batch\_size**.
- **Dense:** là một lớp fully connected layer (lớp kết nối đầy đủ), nghĩa là tất cả các neuron trong lớp hiện tại sẽ được kết nối với tất cả các neuron trong lớp tiếp theo, thường có các tham số như **units** là một số nguyên dương, đại diện cho số lượng nơ-ron trong lớp Dense hay số chiều đầu ra, và **activation** là hàm kích hoạt trên các nơ-ron.
- **compile:** cấu hình mô hình trước khi bắt đầu training. Phương thức này xác định trình tối ưu hoá **optimizer**, hàm mất mát **loss** và các chỉ số đánh giá **metrics** trong quá trình training.
- **ReduceLROnPlateau:** được sử dụng để giảm tốc độ học (learning rate) khi hiệu suất của mô hình không được cải thiện sau một số epoch nhất định. Điều này có thể giúp mô hình hội tụ tốt hơn khi quá trình huấn luyện bị "định trệ" ở một mức hiệu suất nào đó. Method bao gồm các parameter như **monitor** là chỉ số cần theo dõi để xác định khi nào cần giảm tốc độ học, **patience** là số lượng epoch mà mô hình cho phép không cải thiện liên tiếp trong quá trình học, **min\_lr** là giá trị tốc độ học nhỏ nhất cho phép, **mode** xác định khi nào cần giảm tốc độ học.

```
model = keras.Sequential()
model.add(keras.Input(shape= (28, 28, 1)))
model.add(keras.layers.Conv2D(filters= 32, kernel_size= (3, 3), activation= 'relu',
    ))
model.add(keras.layers.MaxPool2D(pool_size= (2, 2)))
model.add(keras.layers.Dropout(rate= 0.3))
model.add(keras.layers.Conv2D(filters= 32, kernel_size= (3, 3), activation= 'relu',
    ))
model.add(keras.layers.MaxPool2D(pool_size= (2, 2)))
```

```
model.add(keras.layers.Dropout(rate= 0.3))
model.add(keras.layers.Conv2D(filters= 32, kernel_size= (3, 3), activation= 'relu',
    ))
model.add(keras.layers.MaxPool2D(pool_size= (2, 2)))
model.add(keras.layers.Dropout(rate= 0.3))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(units= 128, activation= 'relu'))
model.add(keras.layers.Dense(units= 25, activation= 'softmax'))
model.compile(loss= 'sparse_categorical_crossentropy', optimizer= 'adam', metrics= 
    ['acc'])
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor= 'val_acc', patience= 2,
    verbose= 1, mode= 'max', min_lr= 0.00001)
```

## 5.2 Hiển thị các thông tin về model

Phương thức **summary** trong Keras được sử dụng để in ra cấu trúc chi tiết của mô hình đã xây dựng, để hiểu được cách các lớp (layers) kết nối với nhau, kích thước đầu ra của từng lớp, số lượng tham số cần học, và tổng số tham số của toàn bộ mô hình.

```
model.summary()
```

Dưới đây là kết quả sau khi sử dụng phương thức **summary** của mô hình đã xây dựng phía trên.

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
dropout_1 (Dropout)	(None, 5, 5, 32)	0
conv2d_2 (Conv2D)	(None, 3, 3, 32)	9,248
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 32)	0
dropout_2 (Dropout)	(None, 1, 1, 32)	0
flatten (Flatten)	(None, 32)	0
dense (Dense)	(None, 128)	4,224
dense_1 (Dense)	(None, 25)	3,225
<b>Total params:</b> 26,265 (102.60 KB) <b>Trainable params:</b> 26,265 (102.60 KB) <b>Non-trainable params:</b> 0 (0.00 B)		

Hình 10: Thông số của mô hình

### 5.3 Huấn luyện mô hình với số lượng epochs nhất định

Phương thức **fit** được sử dụng để huấn luyện mô hình trên một tập dữ liệu cụ thể. Đây là bước mà mô hình học từ dữ liệu đầu vào và điều chỉnh các tham số của nó thông qua quá trình tối ưu hóa, để dự đoán trong tương lai. Các tham số được truyền vào bao gồm dữ liệu đầu vào, dữ liệu đầu ra tương ứng, **batch\_size** là số mẫu dữ liệu được sử dụng trong mỗi lần học, **epochs** là số lần luyện tập trên bộ dữ liệu.

```
EPOCHS = 5
BATCH_SIZE = 256
start_time = time.time()
history = model.fit(x_train, y_train, batch_size= BATCH_SIZE, epochs= EPOCHS,
                     verbose= 1, validation_data= (x_test, y_test))
end_time = time.time()
print(f'Elapsed time = {round(end_time - start_time, 2)} seconds.')
```

```
Epoch 1/5
235/235 [=====] 23s 90ms/step - acc: 0.3401 - loss: 1.9886 - val_acc: 0.9139 - val_loss: 0.3016
Epoch 2/5
235/235 [=====] 20s 87ms/step - acc: 0.8307 - loss: 0.5342 - val_acc: 0.9482 - val_loss: 0.1797
Epoch 3/5
235/235 [=====] 22s 96ms/step - acc: 0.8802 - loss: 0.3838 - val_acc: 0.9591 - val_loss: 0.1384
Epoch 4/5
235/235 [=====] 23s 97ms/step - acc: 0.9011 - loss: 0.3216 - val_acc: 0.9665 - val_loss: 0.1142
Epoch 5/5
235/235 [=====] 23s 98ms/step - acc: 0.9150 - loss: 0.2776 - val_acc: 0.9708 - val_loss: 0.0986
Elapsed time = 111.93 seconds.
```

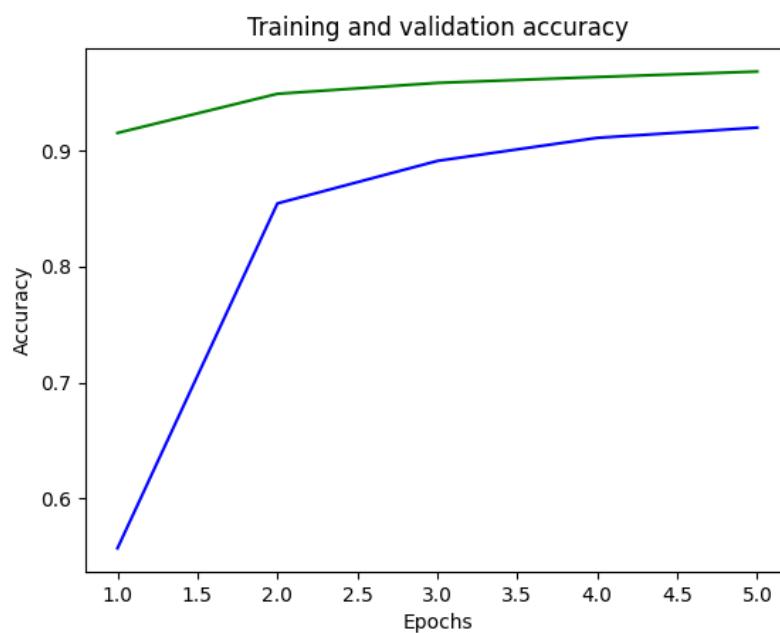
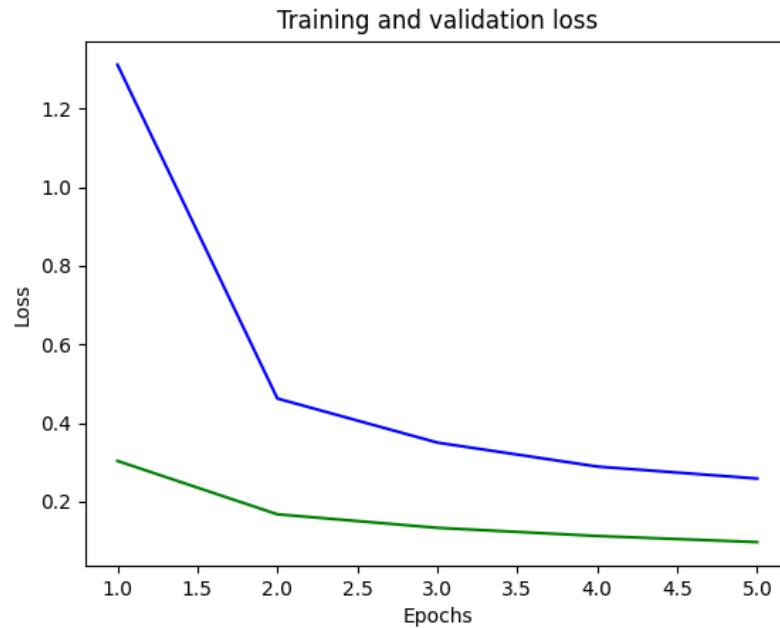
Hình 11: Huấn luyện mô hình

### 5.4 Một số biểu đồ so sánh hàm mất mát và độ chính xác của training data và validation data

Vẽ hình ảnh đồ thị so sánh training loss và validation loss theo thời gian khi số epochs tăng lên.

```
loss = history.history['loss']
validation_loss = history.history['val_loss']
epochs_range = range(1, len(loss) + 1)
plt.plot(epochs_range, loss, 'b', label= 'Training loss')
plt.plot(epochs_range, validation_loss, 'g', label= 'Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
plt.plot(epochs_range, accuracy, 'b', label= 'Training accuracy')
plt.plot(epochs_range, val_accuracy, 'g', label= 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

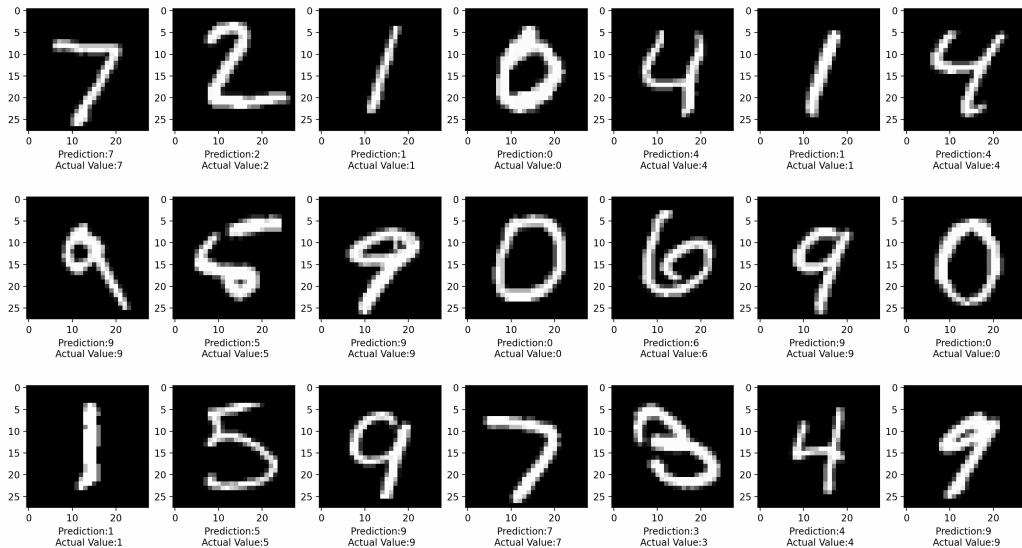


## 5.5 Sử dụng mô hình để dự đoán hình ảnh

Sau khi xây dựng model, phương thức `predict` được sử dụng để dự đoán đầu ra từ mô hình đã được huấn luyện bằng cách truyền vào 1 tham số là 1 array đại diện cho bức ảnh muốn dự đoán.

```
plt.figure(figsize=(18, 9))
for i in range(21):
    plt.subplot(3, 7, i + 1)
    testImage = x_test[i]
    prediction = model.predict(testImage.reshape(-1, 28, 28, 1))
    plt.imshow(testImage.reshape(28, 28), cmap = 'gray')
    plt.xlabel(f"Prediction:{np.argmax(prediction)} \n Actual Value:{y_test[i]}")
```

Dưới đây là kết quả của 21 hình ảnh đầu tiên trong bộ dữ liệu data test, trong đó **Prediction** là dự đoán của máy tính và **Actual Value** là label thực sự của bức ảnh được lưu trữ trong bộ dữ liệu data test.



Hình 12: Dự đoán một số hình ảnh đầu tiên

Trong thực tế, bộ dữ liệu test có khoảng 10000 hình ảnh, kết quả khi ghi nhận dự đoán cả 10000 hình ảnh như sau:

**Pass: 9708 / 10000, Success: 97.08%**

Hình 13: Kết quả khi dự đoán 10000 testcase

Như vậy, trong 10000 testcase được cung cấp, có 97.08% hình ảnh được dự đoán chính xác.



## 5.6 Metrics đánh giá kết quả làm việc

### 5.6.1 Accuracy and Loss

**Accuracy (độ chính xác):** Là chỉ số trực quan nhất, accuracy được xác định bởi tỷ lệ giữa các hình ảnh được phân loại chính xác với tổng số hình ảnh.

$$\text{accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of all predictions}}$$

Chỉ số này hoạt động tốt đối với các tập dữ liệu cân bằng, trong đó tất cả các lớp có số lượng mẫu gần như nhau. Tuy nhiên, độ chính xác có thể gây hiểu lầm đối với các tập dữ liệu mất cân bằng, trong đó mô hình có thể ưu tiên phân loại đúng lớp đa số, ngay cả khi nó hoạt động kém trên lớp thiểu số.

**Loss (hàm mất mát):** Là một chỉ số đánh giá mức độ sai lệch giữa đầu ra dự đoán của mô hình và giá trị thực tế mong đợi. Hàm mất mát đóng vai trò quan trọng trong việc đào tạo CNN cho các tác vụ xử lý hình ảnh. Hàm mất mát tính toán mức độ dự đoán của mô hình lệch khỏi kết quả mong muốn. Hàm mất mát thường được sử dụng làm hàm tối ưu trong quá trình huấn luyện để cập nhật trọng số của mô hình. Giá trị của Loss càng thấp thì mô hình càng dự đoán chính xác.

Có khá nhiều hàm tính toán mất mát khác nhau, một trong những hàm tính toán mất mát có thể sử dụng là Cross Entropy Loss:

$$\log(Loss) = \frac{-1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(\hat{y}_{ij})$$

Trong đó:

- $N$  là số lượng mẫu được dùng để dự đoán.
- $M$  là số lượng các nhãn / lớp (label / class) được dùng để phân loại.
- $y_{ij}$  là giá trị đại diện (0 hoặc 1) cho kết quả dự đoán.
- $\hat{y}_{ij}$  là giá trị xác suất dự đoán tương ứng.

Trong **Keras**, model có cung cấp sẵn phương thức **evaluate** cho phép tính toán giá trị hàm mất mát và độ chính xác của mô hình:

```
loss, acc = model.evaluate(x_test, y_test)
print(f"Loss: {loss}, Accuracy: {acc}")
```

Kết quả sau khi thực hiện câu lệnh:

```
313/313 ━━━━━━━━ 4s 14ms/step - acc: 0.9620 - loss: 0.1205
Loss: 0.102963387966156, Accuracy: 0.9681000113487244
```

Có thể thấy, mô hình có độ chính xác rơi vào khoảng 96.8% và hàm mất mát của mô hình này là vào khoảng 0.103, tương đối gần với 0. Điều này chứng tỏ mô hình hoạt động tương đối ổn định, nhưng vẫn có thể có sai sót, nhầm lẫn, có thể xuất phát từ việc các hình ảnh có nhãn khác nhau nhưng về thông tin có được từ việc trích xuất và học của mô hình là giống nhau.

### 5.6.2 Confusion Matrix

Confusion Matrix (ma trận nhầm lẫn) là một bảng ma trận 2D được sử dụng để xác định performance của các thuật toán và mô hình nhận diện hình ảnh. Confusion Matrix cung cấp cái nhìn sâu hơn về cách mô hình phân loại các lớp (classes) và có thể chỉ ra các loại lỗi cụ thể mà mô hình mắc phải. Confusion Matrix bao gồm các yếu tố sau:

- **TP - True Positive:** số lượng dự đoán một cách chính xác.
- **TN - True Negative :** số lượng dự đoán chính xác một cách gián tiếp.
- **FP - False Positive - Type 1 Error :** số lượng dự đoán sai lệch.
- **FN - False Negative - Type 2 Error :** số lượng dự đoán sai lệch một cách gián tiếp.

Trong Confusion Matrix có các chỉ số chủ yếu thường được sử dụng để đánh giá hiệu suất mô hình:

- **Precision:** trong tất cả các dự đoán Positive được đưa ra, có bao nhiêu dự đoán là chính xác.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall:** trong tất cả các trường hợp Positive, có bao nhiêu trường hợp đã được dự đoán chính xác.

$$Recall = \frac{TP}{TP + FN}$$

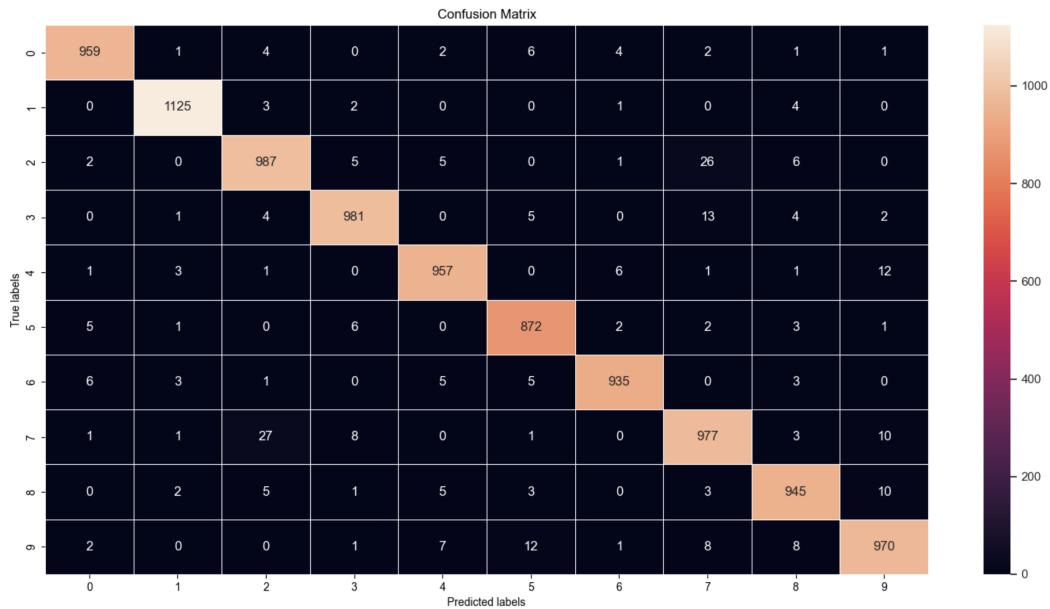
- **F1-Score:** chỉ số chung được kết hợp từ 2 chỉ số Precision và Recall để đánh giá toàn diện mô hình:

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Trong mô hình, để thực hiện thiết lập ma trận nhầm lẫn, cần thiết lập một **prediction\_list** chứa tất cả các dự đoán của tất cả hình ảnh trong test dataset, sau đó đưa danh sách dự đoán này làm tham số của phương thức **confusion\_matrix** để tiến hành vẽ ma trận nhầm lẫn:

```
predictions_list = []
for i in range(len(y_test)):
    testImage = x_test[i]
    prediction = model.predict(testImage.reshape(-1,28,28,1))
    predictions_list.append(np.argmax(prediction))
CONFUSION_MATRIX = confusion_matrix(y_test, predictions_list)
fig, axes = plt.subplots(figsize= (15, 9))
sns.set_theme(font_scale= 1)
sns.heatmap(CONFUSION_MATRIX, annot= True, linewidths= 0.5, ax= axes)
plt.show()
```

Kết quả của câu lệnh trên có thể hiện thị như hình dưới đây:



Trong Confusion Matrix, các hàng đại diện cho label thực tế (True labels), các cột đại diện cho label dự đoán (Predicted labels). Từ kết quả ma trận cho thấy các dự đoán sai của mô hình thường rơi vào các trường hợp nhầm lẫn giữa hình ảnh nào với hình ảnh nào để có các biện pháp cải thiện.



## 6 Kết luận chung

Dựa trên kết quả thực nghiệm, nhóm nhận thấy rằng việc áp dụng phương pháp giảm chiều dữ liệu bằng **PCA** (**Principal Component Analysis**) kết hợp với **Logistic Regression**, cùng với việc sử dụng mô hình **SVM** (**Support Vector Machine**), đã mang lại những kết quả khả quan trong việc giải quyết bài toán phân loại chữ số viết tay trên tập dữ liệu MNIST.

Cụ thể, phương pháp PCA giúp giảm số lượng chiều của dữ liệu bằng cách tìm ra các thành phần chính (principal components) có độ phương sai lớn nhất. Điều này giúp giảm thiểu độ phức tạp của mô hình mà không làm mất đi quá nhiều thông tin quan trọng, từ đó cải thiện tốc độ huấn luyện và hiệu suất của mô hình. Khi kết hợp PCA với **Logistic Regression**, chúng tôi nhận thấy rằng mô hình Logistic Regression hoạt động hiệu quả hơn nhờ vào việc giảm số chiều dữ liệu đầu vào, giảm đáng kể thời gian huấn luyện mô hình mà vẫn giữ lại độ chính xác gần như ban đầu.

Bên cạnh đó, phương pháp **SVM**, vốn nổi bật với khả năng tạo ra các ranh giới phân tách mạnh mẽ giữa các lớp dữ liệu, cũng cho kết quả rất tốt khi được áp dụng trực tiếp trên tập dữ liệu MNIST. SVM đặc biệt hữu ích trong các bài toán phân loại có dữ liệu không tuyến tính, và trong trường hợp này, nó có thể xử lý tốt các mối quan hệ phức tạp giữa các chữ số viết tay, giúp cải thiện độ chính xác của mô hình phân loại.

Tuy nhiên, trong bối cảnh hiện nay, **mang nơ-ron tích chập (CNN)** đã trở thành phương pháp được coi là tốt nhất và hiệu quả nhất cho bài toán nhận dạng chữ số viết tay trên tập dữ liệu MNIST. Với khả năng tự động học và trích xuất các đặc trưng quan trọng từ ảnh mà không cần phải qua bước tiền xử lý phức tạp, CNN thể hiện sự vượt trội so với các phương pháp truyền thống. Các lớp tích chập trong mô hình giúp nhận diện các đặc trưng không gian như đường nét, hình dạng và kết cấu của các chữ số, từ đó cải thiện độ chính xác phân loại. Nhờ vào khả năng xử lý các biến đổi trong hình ảnh như độ nghiêng, thay đổi kích thước hay độ sáng, CNN hiện đang là lựa chọn tối ưu trong việc giải quyết bài toán phân loại trên MNIST, đồng thời trở thành xu hướng chủ đạo trong các nghiên cứu và ứng dụng nhận dạng ảnh.

## Tài liệu tham khảo

- [1] MNIST in CSV, <https://www.kaggle.com/datasets/oddrationale/mnist-in-csv>
- [2] Wen, Zeyi and Shi, Jiashuai and Li, Qinbin and He, Bingsheng and Chen, Jian. *Thunder-SVM: A Fast SVM Library on GPUs and CPUs*. Journal of Machine Learning Research, vol. 19, pp. 797–801, 2018.
- [3] Stanford University Online Course, *CS230 - Deep Learning*, <https://cs230.stanford.edu/>
- [4] Stanford University Online Course, *CS231n - Deep Learning for Computer Vision*, <https://cs231n.stanford.edu/>
- [5] Quán Thành Thơ (2021), *Mang nơ-ron nhân tạo: Từ hồi quy đến học sâu*, Nhà xuất bản Đại học Quốc gia TP.HCM.
- [6] Michael Nielsen, *Neural Networks and Deep Learning*, <https://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf>