**Họ và tên: Hồng Anh Khoa**

**MSSV: 22110351**

**BÁO CÁO MÔ HÌNH MLP PHÂN LOẠI FASHIONMNIST**

# **EXPERIMENT1**

**Khác nhau**

## **Code 1 - Chuẩn hóa với Mean = 0.5 và Std = 0.5**

|  |
| --- |
| transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.5,), (0.5,))  ]) |

* **Ý nghĩa:** Chuẩn hóa dữ liệu với trung bình (mean) = 0.5 và độ lệch chuẩn (std) = 0.5.
* **Tác dụng:** Biến đổi pixel từ [0,1] thành khoảng [-1,1] theo công thức:

x′=(x−0.5)/0.5=2x−1

→ Làm cân bằng dữ liệu quanh 0.

* **Cách hiển thị ảnh:**

|  |
| --- |
| img = img \* 0.5 + 0.5 |

* + Đảo ngược chuẩn hóa về [0,1] để hiển thị đúng màu.
  + img.numpy() chuyển tensor thành numpy để hiển thị bằng plt.imshow().

## **Code 2 - Không chuẩn hóa (Mean = 0, Std = 1.0)**

|  |
| --- |
| transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0,), (1.0,))  ]) |

* **Ý nghĩa:** Không thay đổi dữ liệu vì: x′=(x−0)/1=x

→ Ảnh giữ nguyên giá trị pixel [0,1].

* **Cách hiển thị ảnh:**

|  |
| --- |
| # img = img / 255.0 (được bỏ đi)  np\_img = img.numpy() |

* + Không cần đảo ngược chuẩn hóa vì dữ liệu giữ nguyên.

## **Code 3 - Chuẩn hóa bằng Mean và Std tính toán trước**

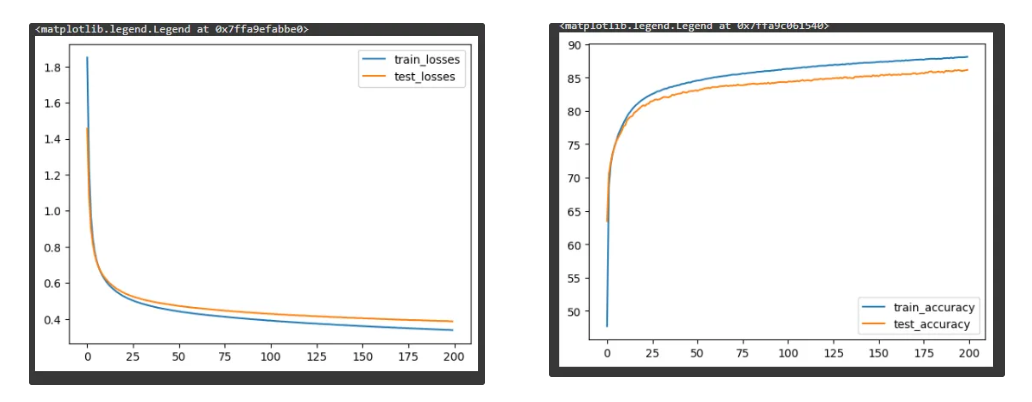
|  |
| --- |
| transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((mean,), (std,))  ]) |

* **Ý nghĩa:** Sử dụng mean và std được **tính toán trước từ tập dữ liệu** (không phải mặc định như Code 1).
* **Ưu điểm:** Giúp mô hình huấn luyện tốt hơn vì chuẩn hóa phù hợp với dữ liệu thật.
* **Cách hiển thị ảnh:**

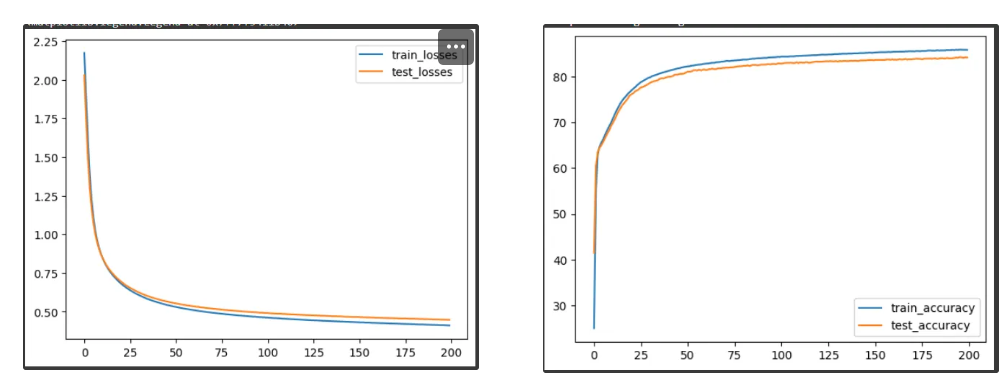
|  |
| --- |
| img = img \* std + mean |

* + Đảo ngược chuẩn hóa bằng giá trị mean, std đã tính.

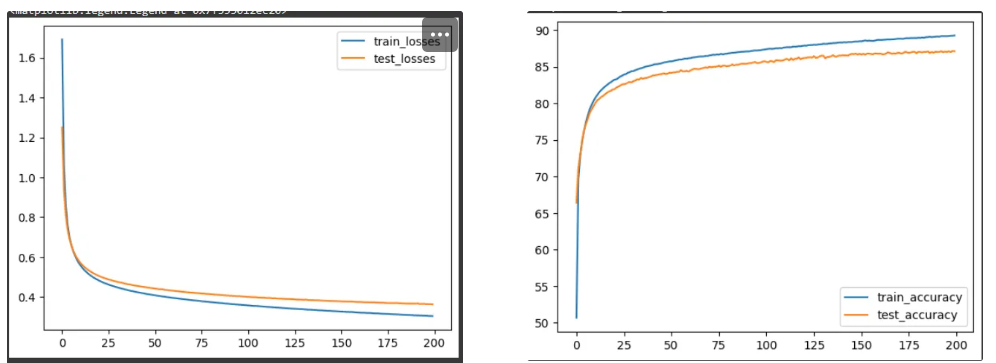
**Đánh giá**

Đồ thị code 1 :

|  |
| --- |
| Epoch [200/200], Loss: 0.3384, Accuracy: 88.09%, Test Loss: 0.3872, Test Accuracy: 86.14% |

Đồ thị code 2 :

|  |
| --- |
| Epoch [200/200], Loss: 0.4109, Accuracy: 85.80%, Test Loss: 0.4470, Test Accuracy: 84.15% |

Đồ thị code 3:

|  |
| --- |
| Epoch [200/200], Loss: 0.3043, Accuracy: 89.25%, Test Loss: 0.3631, Test Accuracy: 87.10% |

## **So sánh hiệu suất của 3 mô hình:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Train Loss** | **Train Accuracy** | **Test Loss** | **Test Accuracy** |
| 200/200 | **0.3384** | **88.09%** | **0.3872** | **86.14%** |
| 200/200 | 0.4109 | 85.80% | 0.4470 | 84.15% |
| 200/200 | **0.3043** | **89.25%** | **0.3631** | **87.10%** |

**Suy ra**

1. **Mô hình thứ 3** có **Train Loss thấp nhất (0.3043) và Train Accuracy cao nhất (89.25%)**, cho thấy nó học tốt nhất trên tập huấn luyện.
2. **Mô hình thứ 3 cũng có Test Loss thấp nhất (0.3631) và Test Accuracy cao nhất (87.10%)**, điều này chứng tỏ nó tổng quát hóa tốt hơn trên tập kiểm tra.
3. **Mô hình thứ 2 có hiệu suất kém nhất** (Train Accuracy 85.80%, Test Accuracy 84.15%), chứng tỏ nó có thể chưa học tốt hoặc bị ảnh hưởng bởi phương pháp tiền xử lý dữ liệu.

## **Kết luận:**

* **Mô hình thứ 3 là tốt nhất** vì nó có **độ chính xác cao nhất và loss thấp nhất trên cả tập huấn luyện và kiểm tra**.
* **Mô hình thứ 1 có kết quả trung bình** và vẫn tốt hơn mô hình thứ 2.
* **Mô hình thứ 2 có hiệu suất kém nhất**, có thể do cách chuẩn hóa dữ liệu chưa phù hợp.

# **EXPERIMENT2**

## **1. Mục đích và cách hoạt động của MLP trên Fashion-MNIST**

**Mục đích**

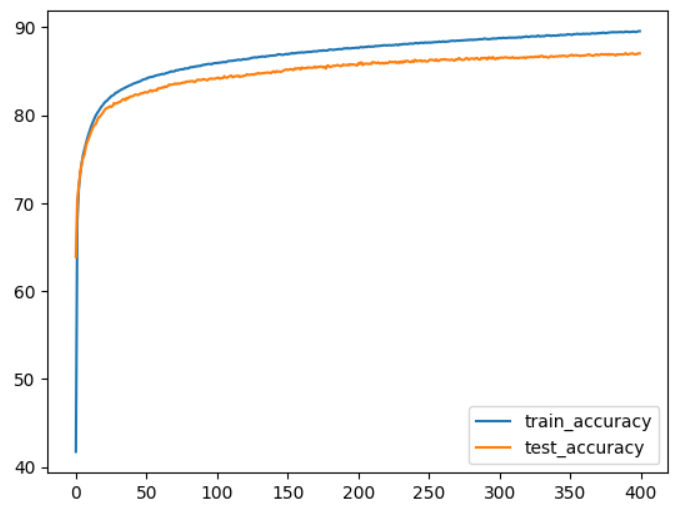
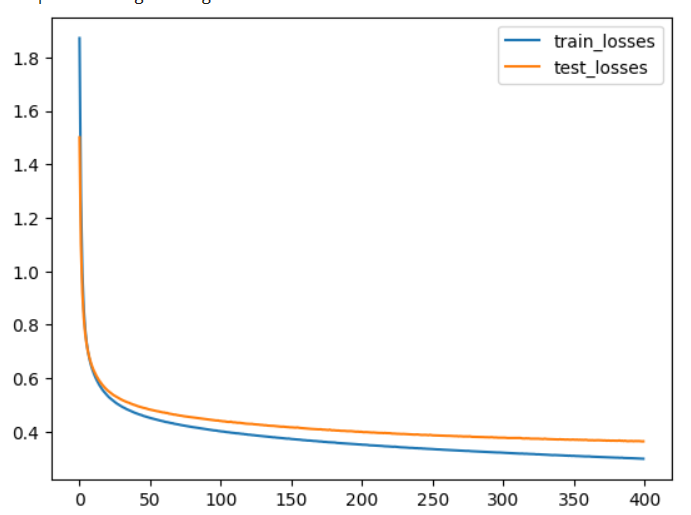
* Mô hình được huấn luyện để phân loại hình ảnh **quần áo, giày dép, túi xách** trong tập dữ liệu **Fashion-MNIST** gồm **10 loại khác nhau**.
* Dữ liệu **Fashion-MNIST** gồm **60.000 ảnh dùng để huấn luyện** và **10.000 ảnh để kiểm tra**, mỗi ảnh có kích thước **28x28 pixels** và được biểu diễn dưới dạng **một vector 784 chiều** (28x28 = 784).
* Mục tiêu của MLP là **học cách nhận dạng đặc trưng** của từng loại sản phẩm thời trang để dự đoán nhãn chính xác.

**Cách hoạt động của MLP**

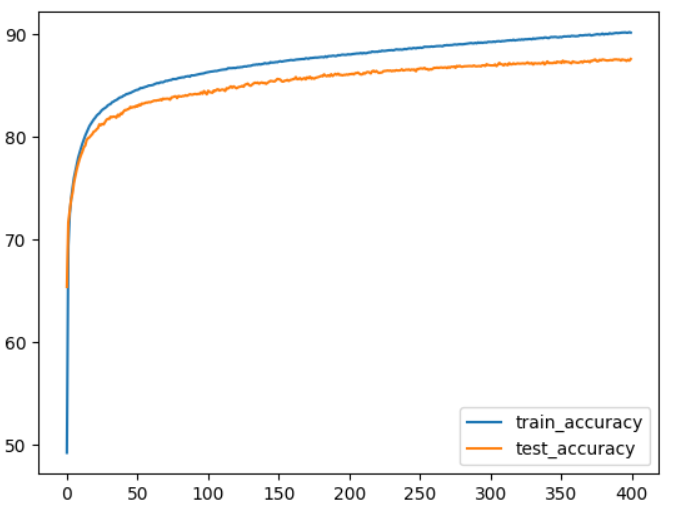
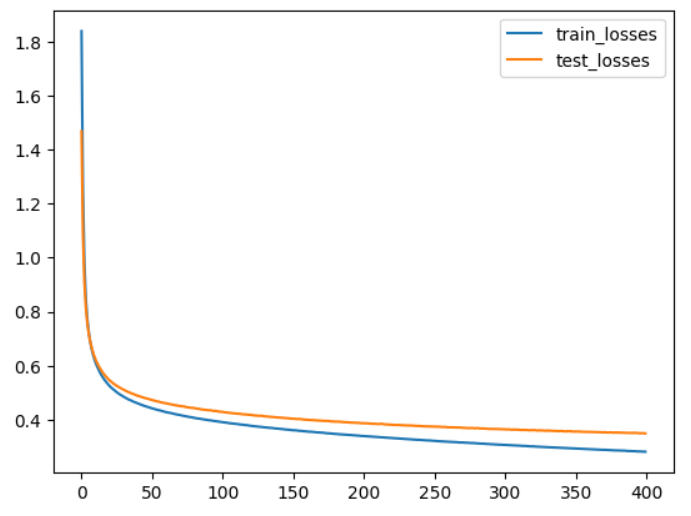
1. **Nhập dữ liệu:**
   * Ảnh 28x28 được chuyển thành một vector 784 phần tử.
2. **Truyền qua các tầng ẩn:**
   * Dữ liệu đi qua **các tầng fully connected (FC)** với số lượng **nodes ẩn khác nhau** (64, 256, 1024).
   * Các node này chứa trọng số (weights) được học trong quá trình huấn luyện.
   * Hàm kích hoạt ReLU (Rectified Linear Unit) được sử dụng để tăng khả năng học phi tuyến.
3. **Tầng đầu ra:**
   * Tầng cuối có **10 neurons** (tương ứng 10 nhãn của Fashion-MNIST).
   * Dùng hàm **Softmax** để chuyển đổi kết quả thành xác suất dự đoán.
4. **Tính hàm mất mát & tối ưu hóa:**
   * Sử dụng **Cross-Entropy Loss** để đo lường sai số.
   * Dùng **Adam hoặc SGD Optimizer** để cập nhật trọng số và giảm lỗi.
5. **Huấn luyện & đánh giá:**
   * Mô hình được huấn luyện trong nhiều epoch (ở bài này, ta sử dụng 400 epochs), sau đó kiểm tra trên tập test để tính độ chính xác.

## **2. Phân tích từng mô hình MLP**

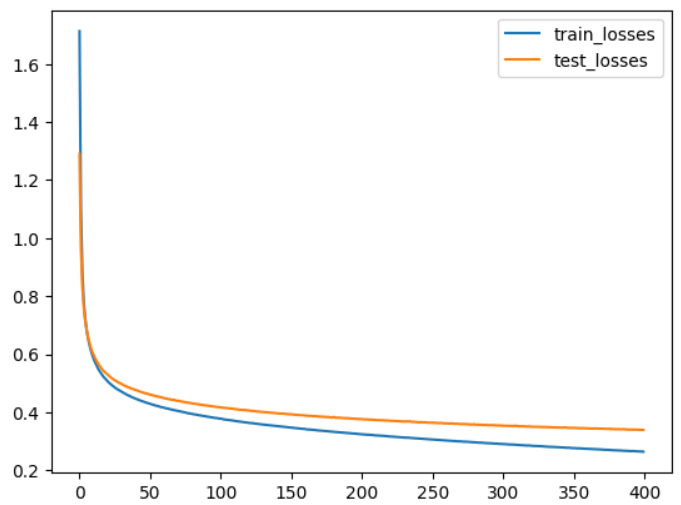
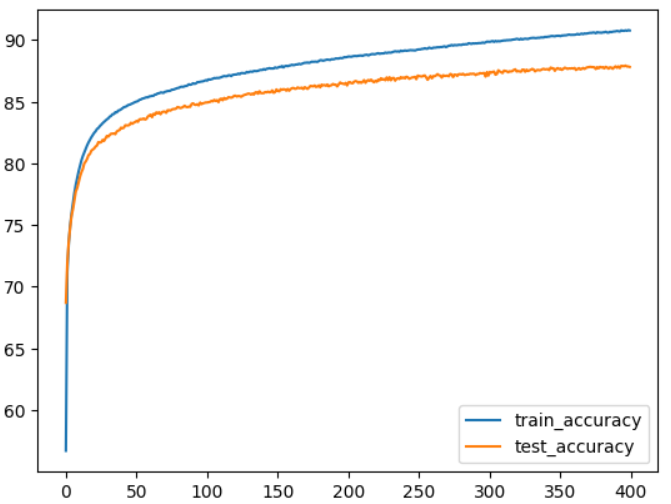
**Phân tích mô hình:**

* **Đồ thị mô hình 64 nodes**

|  |
| --- |
| Epoch [400/400], Loss: 0.2986, Accuracy: 89.57%, Test Loss: 0.3638, Test Accuracy: 87.02% |

* **Đồ thị mô hình 256 nodes**

|  |
| --- |
| Epoch [400/400], Loss: 0.2820, Accuracy: 90.14%, Test Loss: 0.3499, Test Accuracy: 87.57% |

* **Đồ thị mô hình 1024 nodes**

|  |
| --- |
| Epoch [400/400], Loss: 0.2638, Accuracy: 90.78%, Test Loss: 0.3391, Test Accuracy: 87.82% |

**(a) MLP với 64 Nodes**

* **Cấu trúc**:
  + 1 tầng ẩn với **64 nodes**
  + Dùng hàm kích hoạt **ReLU**
* **Ưu điểm**:
  + Kích thước mô hình nhỏ, tốc độ huấn luyện nhanh.
  + Ít bị overfitting do số lượng tham số ít hơn.
* **Nhược điểm**:
  + Dễ bị underfitting do không đủ khả năng học đặc trưng phức tạp của dữ liệu.

**(b) MLP với 256 Nodes**

* **Cấu trúc**:
  + 1 tầng ẩn với **256 nodes**
  + Dùng hàm kích hoạt **ReLU**
* **Ưu điểm**:
  + Cân bằng giữa độ phức tạp và tốc độ huấn luyện.
  + Học đặc trưng tốt hơn so với mô hình 64 nodes.
* **Nhược điểm**:
  + Mất nhiều tài nguyên tính toán hơn mô hình 64 nodes.

**(c) MLP với 1024 Nodes**

* **Cấu trúc**:
  + 1 tầng ẩn với **1024 nodes**
  + Dùng hàm kích hoạt **ReLU**
* **Ưu điểm**:
  + Mô hình mạnh mẽ hơn, có khả năng học nhiều đặc trưng hơn.
  + Độ chính xác thường cao hơn.
* **Nhược điểm**:
  + Dễ bị **overfitting** nếu dữ liệu không đủ lớn.
  + Tốn nhiều bộ nhớ và thời gian huấn luyện.

## **3. Hiệu suất của từng mô hình**

**Dự đoán hiệu suất**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mô hình** | **Số nodes ẩn** | **Tốc độ huấn luyện** | **Độ chính xác (dự đoán)** | **Rủi ro overfitting** |
| **64 Nodes** | 64 | 🟢 Nhanh nhất | 🔴 Thấp nhất (~80-85%) | 🟢 Ít overfitting |
| **256 Nodes** | 256 | 🟡 Trung bình | 🟡 Trung bình (~85-88%) | 🟡 Cân bằng |
| **1024 Nodes** | 1024 | 🔴 Chậm nhất | 🟢 Cao nhất (~88-92%) | 🔴 Dễ overfitting |

* **MLP 64 nodes**: Huấn luyện nhanh nhưng có thể không đủ mạnh để học hết đặc trưng của Fashion-MNIST.
* **MLP 256 nodes**: Cân bằng giữa tốc độ và độ chính xác.
* **MLP 1024 nodes**: Mạnh nhất nhưng có thể bị overfitting nếu không có kỹ thuật regularization.

## **4. Đánh giá và kết luận**

Khi nào nên chọn mỗi mô hình?

🔹 MLP 64 nodes

✅ Khi cần tốc độ nhanh, tài nguyên hạn chế, tránh overfitting.

🚫 Không phù hợp nếu cần độ chính xác cao.

🔹 MLP 256 nodes

✅ Cân bằng giữa hiệu suất và tốc độ, phù hợp với hầu hết ứng dụng.

🚫 Không phải là mô hình mạnh nhất nhưng có độ chính xác tốt.

🔹 MLP 1024 nodes

✅ Khi cần độ chính xác cao nhất.

🚫 Cần nhiều tài nguyên và có thể bị overfitting.

# **EXPERIMENT3**

Dữ liệu đầu vào: ***FashionMNIST*** *là tập dữ liệu hình ảnh (28x28 pixels) của các loại quần áo, giày dép, túi xách, với 10 nhãn phân loại* *Dữ liệu được chuẩn hóa về khoảng (-1,1) bằng transforms.Normalize((0.5,), (0.5,)), giúp mô hình học hiệu quả hơn.*

***Batch size*** *= 1024 (tương đối lớn, giúp tăng tốc training nhưng có thể làm mất đi một số thông tin chi tiết trong quá trình huấn luyện).* ***Sử dụng num\_workers=10*** *để tăng tốc load dữ liệu.*

Sự khác nhau: Bốn mô hình đều là **MLP (Multi-Layer Perceptron)** để phân loại ảnh FashionMNIST. Điểm khác biệt chính giữa chúng là số lượng tầng ẩn (hidden layers). Dưới đây là phân tích từng mô hình:

## **Code 1**

|  |
| --- |
| model = nn.Sequential(  nn.Flatten(),  nn.Linear(784, 256),  nn.ReLU(),  nn.Linear(256, 10)  )  model = model.to(device)  print(model) |

* **Cấu trúc**:
  + 1 lớp ẩn (256 neurons).
  + Kích thước đầu vào: **784** (ảnh 28×28).
  + Kích thước đầu ra: **10** (10 lớp của FashionMNIST).
* **Đánh giá**:
  + Đơn giản nhất, ít tham số nhất → tốc độ huấn luyện nhanh.
  + Có thể không đủ độ phức tạp để học được các đặc trưng phức tạp.

|  |
| --- |
| Epoch [400/400], Loss: 0.2794, Accuracy: 90.25%, Test Loss: 0.3493, Test Accuracy: 87.61% |

## **Code 2**

|  |
| --- |
| model = nn.Sequential(  nn.Flatten(),  nn.Linear(784, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 10)  )  model = model.to(device)  print(model) |

* **Cấu trúc**:
  + 2 lớp ẩn (256 neurons mỗi lớp).
* **Đánh giá**:
  + Tăng độ phức tạp so với mô hình 1, giúp mô hình học tốt hơn.
  + Có thêm một tầng ReLU giúp mô hình học được nhiều đặc trưng hơn.
  + Có thể cải thiện độ chính xác nhưng cũng tăng nguy cơ overfitting.

|  |
| --- |
| Epoch [400/400], Loss: 0.2432, Accuracy: 91.39%, Test Loss: 0.3380, Test Accuracy: 88.05% |

## **Code 3**

|  |
| --- |
| model = nn.Sequential(  nn.Flatten(),  nn.Linear(784, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 10)  )  model = model.to(device)  print(model) |

* **Cấu trúc**:
  + 3 lớp ẩn (256 neurons mỗi lớp).
* **Đánh giá**:
  + Mô hình có độ phức tạp cao hơn, có thể học được nhiều đặc trưng hơn.
  + Có nguy cơ overfitting nếu tập dữ liệu không đủ lớn hoặc không có regularization.

|  |
| --- |
| Epoch [400/400], Loss: 0.2282, Accuracy: 91.75%, Test Loss: 0.3407, Test Accuracy: 88.35% |

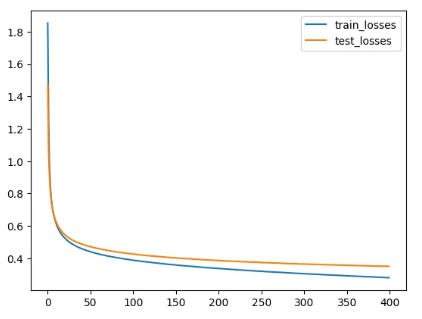
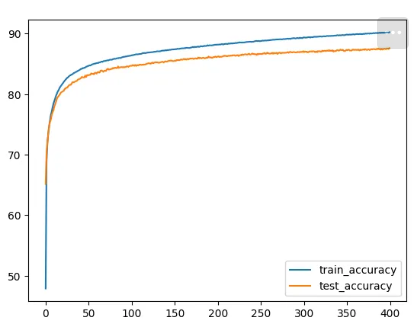
## **Code 4**

|  |
| --- |
| model = nn.Sequential(  nn.Flatten(),  nn.Linear(784, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 256),  nn.ReLU(),  nn.Linear(256, 10)  )  model = model.to(device)  print(model) |

* **Cấu trúc**:
  + 4 lớp ẩn (256 neurons mỗi lớp).
* **Đánh giá**:
  + Phức tạp nhất, có khả năng học đặc trưng mạnh mẽ nhất.
  + Nguy cơ overfitting cao nhất.
  + Huấn luyện chậm hơn do có nhiều tham số hơn.

|  |
| --- |
| Epoch [400/400], Loss: 0.1954, Accuracy: 92.92%, Test Loss: 0.3314, Test Accuracy: 88.51% |

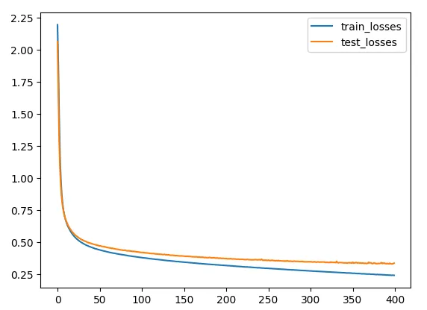
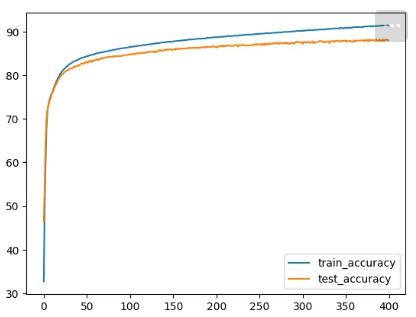
## **Phân tích mô hình**

**BIỂU ĐỒ CODE 1**

Loss: **0.2794 (train), 0.3493 (test)**

Accuracy: **90.25% (train), 87.61% (test)**

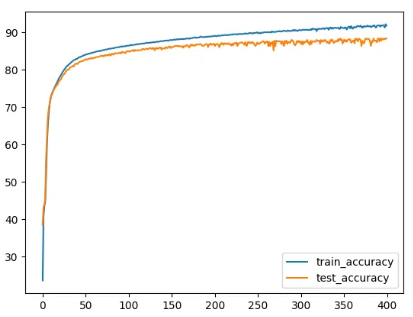
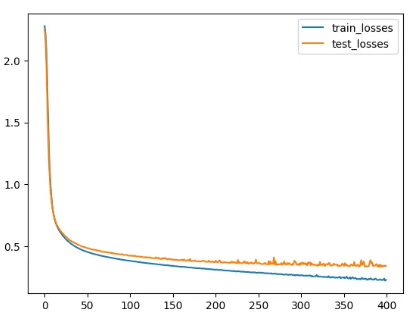
* Độ chính xác thấp nhất trong 4 mô hình.
* Độ ổn định cao

**BIỂU ĐỒ CODE 2:**

Loss: **0.2432 (train), 0.3380 (test)**

Accuracy: **91.39% (train), 88.05% (test)**

Hiệu suất tốt hơn model 1, cải thiện đáng kể độ chính xác.

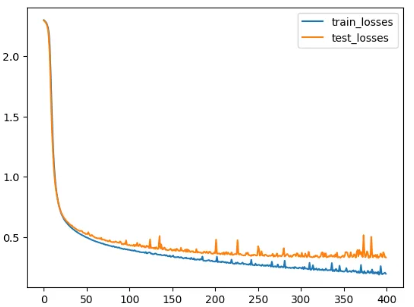
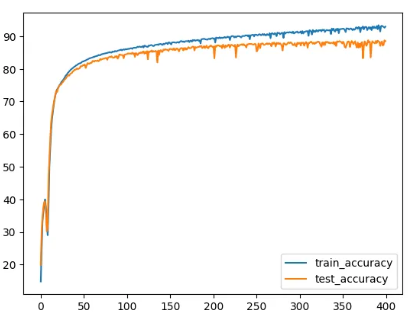
******BIỂU ĐỒ CODE 3:**

Loss: **0.2282 (train), 0.3407 (test)**

Test loss có dấu hiệu dao động nhẹ, cho thấy model có thể bắt đầu overfitting.

Accuracy: **91.75% (train), 88.35% (test)**

Độ chính xác tăng nhẹ so với model 2, nhưng không đáng kể.

**BIỂU ĐỒ CODE 4:**

Loss: **0.1954 (train), 0.3314 (test)**

Test loss dao động mạnh hơn, dấu hiệu của **overfitting**.

Accuracy: **92.92% (train), 88.51% (test)**

Train accuracy cao nhất (**92.92%**) nhưng test accuracy không cải thiện nhiều (**88.51%**).

Biểu đồ dao động mạnh hơn (hình 4), dấu hiệu của **overfitting**.

## **KẾT LUẬN :**

**Model 2 là lựa chọn tối ưu vì:**

* Model 2 có **test loss ổn định nhất** và đạt độ chính xác tốt (**88.05%**) không quá thấp khi so với model 3-4
* Ở Model 2 sự giao động của test loss là có nhưng ở mức độ nhẹ, có thể huấn luyện tiếp mà