



Phân loại hình ảnh thời trang bằng các phương pháp học máy

Thành viên:

Hoàng Thiết Lâm - 22001605
Bùi Hữu Phước - 22001630
Phạm Gia Nguyên - 22001626

Giảng viên hướng dẫn:
Cao Văn Chung

Lời cảm ơn

Chúng tôi xin bày tỏ lòng biết ơn sâu sắc và kính trọng đối với những người đã hỗ trợ chúng tôi trong quá trình xây dựng bản báo cáo này.

Trước hết, chúng tôi xin gửi lời cảm ơn chân thành nhất đến thày Cao Văn Chung, giảng viên môn Học máy. Những kiến thức mà chúng tôi áp dụng trong báo cáo này đã được đúc kết từ các bài giảng sâu sắc và lời khuyên quý báu của thày trong suốt quá trình học tập. Thày đã giúp chúng tôi hiểu rõ các thuật toán Học máy, không chỉ về khía cạnh Toán học mà còn về ứng dụng thực tiễn.

Tiếp theo, chúng tôi xin gửi lời tri ân chân thành đến anh Lê Ngọc Toàn, trợ giảng môn Học máy. Trong suốt các buổi thực hành, anh đã luôn đồng hành, giải đáp thắc mắc và hỗ trợ chúng tôi khi triển khai các mô hình Học máy vào những bài toán cụ thể. Sự hướng dẫn nhiệt tình và những lời khuyên hữu ích của anh đã đóng góp không nhỏ vào thành công của bản báo cáo này.

Về nội dung báo cáo, bên cạnh các mô hình được giảng dạy trong môn học như Principal Component Analysis (PCA), KMeans, Gaussian Mixture Model (GMM), Logistic Regression, Support Vector Machine (SVM), và Convolutional Neural Networks (CNN), một số mô hình khác nằm ngoài phạm vi chương trình đã được nhóm chúng tôi nghiên cứu và triển khai. Dù phải tự tìm kiếm tài liệu và tự phân tích vấn đề, chúng tôi đã học hỏi được nhiều kỹ năng quan trọng trong quá trình thực hiện, bao gồm: viết báo cáo, tìm kiếm tài liệu, làm việc nhóm, và tư duy giải quyết vấn đề.

Chúng tôi tin rằng, nhờ những kiến thức nền tảng từ môn Học máy cùng kinh nghiệm tích lũy qua việc thực hiện báo cáo này, chúng tôi đã tiến thêm một bước trong hành trình trở thành những kỹ sư Học máy chuyên nghiệp.

Phân chia công việc giữa kỲ

STT	Công Việc	Người Phụ Trách
1	Lý thuyết và thực nghiệm Logistic Regression	Bùi Hữu Phước
2	Lý thuyết và thực nghiệm Convolutional Neural Network	Hoàng Thiết Lâm
3	Lý thuyết và thực nghiệm Principal Component Analysis	Phạm Gia Nguyên
4	Lý thuyết và thực nghiệm T-Distributed Neighbor Embedding	Hoàng Thiết Lâm

Mục lục

1 Giới thiệu	6
2 Cơ sở lý thuyết	7
2.1 Phương pháp giảm chiều t-distributed Stochastic Neighbor Embedding	7
2.1.1 Thuật toán t-SNE	7
2.1.2 Tối ưu hóa thuật toán t-SNE	8
2.2 PCA	9
2.2.1 Khái niệm về PCA [1]	9
2.2.2 Mục tiêu và Ý tưởng của PCA	9
2.2.3 Các bước thực hiện PCA	10
2.2.4 Mối quan hệ giữa PCA và SVD	12
2.3 Hàm xác suất phân lớp Softmax	13
2.4 Phương pháp Softmax Regression	13
2.4.1 Khởi tạo và hướng đi	14
2.4.2 Kiến trúc mô hình Softmax Regression	14
2.4.3 Hàm mất mát của Softmax Regression	15
2.4.4 Tối ưu hàm mất mát	15
2.5 Convolutional Neural Network	16
2.5.1 Cấu trúc cơ bản của CNN	16
2.5.2 Tầng Convolution	17
2.5.3 Tầng Pooling	18
2.5.4 Tầng Flatten	18
2.5.5 Tầng Fully-connected	19
2.5.6 Tầng Dropout[2]	19
2.5.7 Tầng Batch Normalization[3]	21
2.6 Độ đo đánh giá kết quả	22
2.6.1 Ma trận Confusion	22
2.6.2 Accuracy	23
2.6.3 Precision	23
2.6.4 Recall	23
3 Thực nghiệm - Đánh giá	24
3.1 Dữ liệu sử dụng	24
3.2 Áp dụng các phương pháp giảm chiều và minh họa	25
3.2.1 PCA	25
3.2.2 T-SNE	31
3.3 Sử dụng Softmax Regression	33
3.3.1 Áp dụng Softmax Regression với dữ liệu chưa giảm chiều	33

3.3.2	Áp dụng Softmax Regression với dữ liệu đã giảm chiều	36
3.3.3	Lựa chọn mô hình phù hợp và đánh giá	36
3.4	Sử dụng Convolutional Neural Network	38
4	Kết luận	41

Danh sách hình ảnh

2.1	Hình biểu diễn các bước lặp của t-SNE từ tập dữ liệu MNIST. Mỗi mẫu dữ liệu ứng với ảnh viết tay của số 2/4/6/8. Tham số được sử dụng là perplexity = 30, $\alpha = 12$, $h = 200$, $K_0 = 40$. Ba hàng đầu ứng với bước làm quá ban đầu, ba hàng cuối ứng với bước biểu diễn.	9
2.2	Trục quan PCA trên đồ thị 2D. [1]	11
2.3	Mô hình Softmax Regression.	14
2.4	Minh họa cấu trúc CNN	16
2.5	Minh họa về tầng Convolution	18
2.6	Minh họa về tầng Pooling	18
2.7	Minh họa tầng Flatten	19
2.8	Minh họa tầng Fully Connected	19
2.9	Bên trái là mạng MLP gốc với 2 tầng Hidden. Bên phải là ví dụ của mạng MLP mới sau khi áp dụng Dropout. Các node bị gạch X là bị loại bỏ	20
2.10	Chia MLP thành nhiều mạng con	21
2.11	Ma trận nhầm lẫn cho từng lớp.	23
3.1	Ảnh thực tế của bộ dữ liệu, mỗi ảnh ứng với một nhãn	24
3.2	Biểu đồ phương sai tích lũy.	26
3.3	Biểu đồ phương sai giải thích.	26
3.4	Phương sai tích lũy của 100 thành phần chính đầu tiên.	27
3.5	5 ảnh ngẫu nhiên được giảm về 100 và 50 chiều.	28
3.6	5 ảnh ngẫu nhiên được giảm về 100 và 50 chiều.	28
3.7	6 cặp ảnh đầu tiên khi trực quan hóa.	29
3.8	Trực quan hóa 2D sau khi giảm chiều bằng PCA.	29
3.9	Trực quan hóa 3D sau khi giảm chiều bằng PCA.	30
3.10	Trực quan hóa dữ liệu gốc đã được giảm xuống 2 chiều qua thuật toán t-SNE	31
3.11	Trực quan hóa dữ liệu gốc đã được giảm xuống 3 chiều qua thuật toán t-SNE	32
3.12	Trực quan hóa dữ liệu giảm chiều bằng PCA trước, đã được giảm xuống 2 chiều qua thuật toán t-SNE	32
3.13	Trực quan hóa dữ liệu giảm chiều bằng PCA trước, đã được giảm xuống 3 chiều qua thuật toán t-SNE	33
3.14	So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.	34

3.15	So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.	34
3.16	So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.	35
3.17	Ma trận nhầm lẫn của phương pháp Softmax Regression.	36
3.18	Ma trận Confusion của mô hình CNN	39
3.19	Hình ảnh mô tả sự thay đổi của Accuracy khi huấn luyện mô hình . .	39
3.20	Hình ảnh mô tả sự thay đổi của Loss khi huấn luyện mô hình . . .	40

Danh sách bảng

3.1	Bảng thống kê dữ liệu FashionMNIST	25
3.2	Giá trị trước và sau chuẩn hóa	25
3.3	Kiểm tra giá trị NaN trong bộ dữ liệu	25
3.4	Thực nghiệm với các bộ dữ liệu và không dùng hiệu chỉnh L2. . . .	33
3.5	Thực nghiệm với các bộ dữ liệu sau khi dùng hiệu chỉnh L2. . . .	35
3.6	Thực nghiệm với các bộ dữ liệu và không dùng hiệu chỉnh L2. . . .	36

Danh sách thuật toán

1	Quá trình huấn luyện với Batch Normalization và Mini-Batch B . . .	22
2	Quá trình dự đoán với Batch Normalization	22

Chương 1

Giới thiệu

Phân loại ảnh là một trong những bài toán kinh điển trong lĩnh vực Xử lý ảnh và Thị giác máy tính. Bản báo cáo này sẽ thực hiện phân loại ảnh thời trang để phục vụ cho việc mua sắm quần áo của những người có khiếm khuyết về thị giác. Bản báo cáo này sẽ áp dụng các thuật toán giảm chiều khác nhau để giảm bớt độ phức tạp tính toán của bộ dữ liệu có số chiều cao, kiểm tra xem có các phân cụm trong bộ dữ liệu, và áp dụng các mô hình phân loại khác nhau để so sánh và đánh giá kết quả.

Chương 2

Cơ sở lý thuyết

2.1 Phương pháp giảm chiều t-distributed Stochastic Neighbor Embedding

Thuật toán T-distributed Stochastic Neighbor Embedding(t-SNE) là một thuật toán phi tuyến giảm chiều và trực quan hóa dữ liệu nổi tiếng. Kỹ thuật này trực quan hóa dữ liệu nhiều chiều bằng việc đặt từng điểm dữ liệu ứng với một vị trí trong không gian 2 chiều hoặc 3 chiều. So với những kỹ thuật khác, t-SNE tốt hơn bởi việc tạo một không gian đơn mà phát hiện được các cấu trúc ở những độ đo khác nhau.

2.1.1 Thuật toán t-SNE

Cho $\{X_i\}_{1 \leq i \leq n}$ là tập điểm dữ liệu với số chiều là p. t-SNE bắt đầu bởi việc tính phân phối xác suất đồng thời trên tất cả các cặp dữ liệu $\{(X_i, X_j)\}_{1 \leq i \neq j \leq n}$. Các phân phối xác suất đó được biểu diễn bởi ma trận đối xứng $\mathbf{P} = (p_{ij})_{1 \leq i,j \leq n} \in \mathbb{R}^{n \times n}$, với $p_{ii} = 0$ với mọi $1 \leq i \leq n$ và với $i \neq j$ thì,

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}, \quad p_{j|i} = \frac{\exp\left(-\frac{\|X_i - X_j\|_2^2}{2\tau_i^2}\right)}{\sum_{\ell \in \{1,2,\dots,n\} \neq i} \exp\left(-\frac{\|X_i - X_\ell\|_2^2}{2\tau_i^2}\right)} \quad (1)$$

Ở đây, τ_i là tham số hiệu chỉnh, được chọn dựa trên một độ đo gọi là perplexity (đánh giá mô hình xác suất dự đoán mẫu tốt như thế nào) và chiến lược tìm kiếm nhị phân. Tương tự, đối với không gian 2 chiều $\{y_i\}_{1 \leq i \leq n} \subset \mathbb{R}^2$, ta định nghĩa phân phối xác suất đồng thời cho tất cả các cặp $\{(y_i, y_j)\}_{1 \leq i \neq j \leq n}$, được biểu diễn qua ma trận đối xứng $\mathbf{Q} = (q_{ij})_{1 \leq i,j \leq n}$ với $q_{ij} = 0$ với mọi $1 \leq i \leq n$ và với $i \neq j$ thì,

$$q_{ij} = \frac{(1 + \|y_i - y_j\|_2^2)^{-1}}{\sum_{\ell,s \in \{1,2,\dots,n\}, \ell \neq s} (1 + \|y_\ell - y_s\|_2^2)^{-1}} \quad (2)$$

\mathbf{P} và \mathbf{Q} được coi là các ma trận tương đồng, tính toán khoảng cách theo từng cặp của dữ liệu nhiều chiều $\{X_i\}_{1 \leq i \leq n}$ và dữ liệu hai chiều $\{y_i\}_{1 \leq i \leq n}$. Sau đó, t-SNE tìm $\{y_i\}_{1 \leq i \leq n}$ mà tối thiểu hóa độ đo Kullback-Leibler divergence giữa \mathbf{P} và \mathbf{Q} với

$$(y_1, \dots, y_n) = \arg \min_{y_1, \dots, y_n} D_{KL}(\mathbf{P}, \mathbf{Q}) = \arg \min_{y_1, \dots, y_n} \sum_{i,j \in \{1,2,\dots,n\}, i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (3)$$

Nhiều thuật toán đã được đưa ra để giải bài toán tối ưu này. Thuật toán phổ biến nhất được đưa ra bởi van der Maaten and Hinton(2008)[4], một biến thể của thuật toán Gradient Descent, với phương trình cập nhật là:

$$y_i^{(k+1)} = y_i^{(k)} + h D_i^{(k)} + m^{(k+1)} \left(y_i^{(k)} - y_i^{(k-1)} \right) \quad \text{for } i = 1, \dots, n. \quad (4)$$

với k là chỉ số của lần lặp thứ mây, $h \in \mathbb{R}$ là tham số độ dài bước được chỉ định, $D_i^{(k)} = 4 \sum_{j \neq i} \left(y_j^{(k)} - y_i^{(k)} \right) S_{ij}^{(k)} \in \mathbb{R}^2$ là hệ số gradient ứng với điểm dữ liệu y_i , và $S_{ij}^{(k)} = (p_{ij} - q_{ij}^{(k)}) / (1 + \|y_i^{(k)} - y_j^{(k)}\|_2^2) \in \mathbb{R}$, và $m^{(k)} \in \mathbb{R}$ là tham số momentum(động lượng) được chỉ định.

Thuật toán khởi tạo với $y_i^0 = y_i^{(-1)}$ với $i \in \{1, 2, \dots, n\}$, được lấy ra từ một uniform distribution(phân phối đồng nhất) trên $[-0, 01, 0.01]^2$ hoặc từ phân phối Gaussian $N(0, \delta^2 I)$ với một số $\delta > 0$. Tham số động lượng $m^{(k+1)}(y_i^{(k)} - y_i^{(k-1)})$ được xét đến trong công thức đơn giản là nhằm tăng tốc độ hội tụ và tránh việc bị kẹt ở một cực tiểu địa phương. Để đơn giản hóa, ta chỉ xét phiên bản đơn giản của thuật toán t-SNE dựa trên công thức Gradient Descent đơn giản

$$y_i^{(k+1)} = y_i^{(k)} + h D_i^{(k)}, \quad \text{for } i = 1, \dots, n \quad (5)$$

Số lần lặp được khuyến cáo là 1000, trong khi độ dài bước h được đặt ban đầu là 400 hoặc 800, và được cập nhật mỗi lần lặp theo một phương pháp học của Jacobs [5]

2.1.2 Tối ưu hóa thuật toán t-SNE

Thuật toán Gradient Descent đơn giản ở (5) gặp vấn đề tốc độ hội tụ chậm, thậm chí là không hội tụ. Để xử lý, van der Maaten và Hinton (2008)[4] đã đưa ra kỹ thuật early exaggeration(làm quá ban đầu), áp dụng cho những bước đầu của quá trình tối ưu hóa, giúp tạo ra những đặc trưng để trực quan hóa và tăng tốc độ hội tụ. Kỹ thuật làm quá ban đầu gồm 2 bước như sau

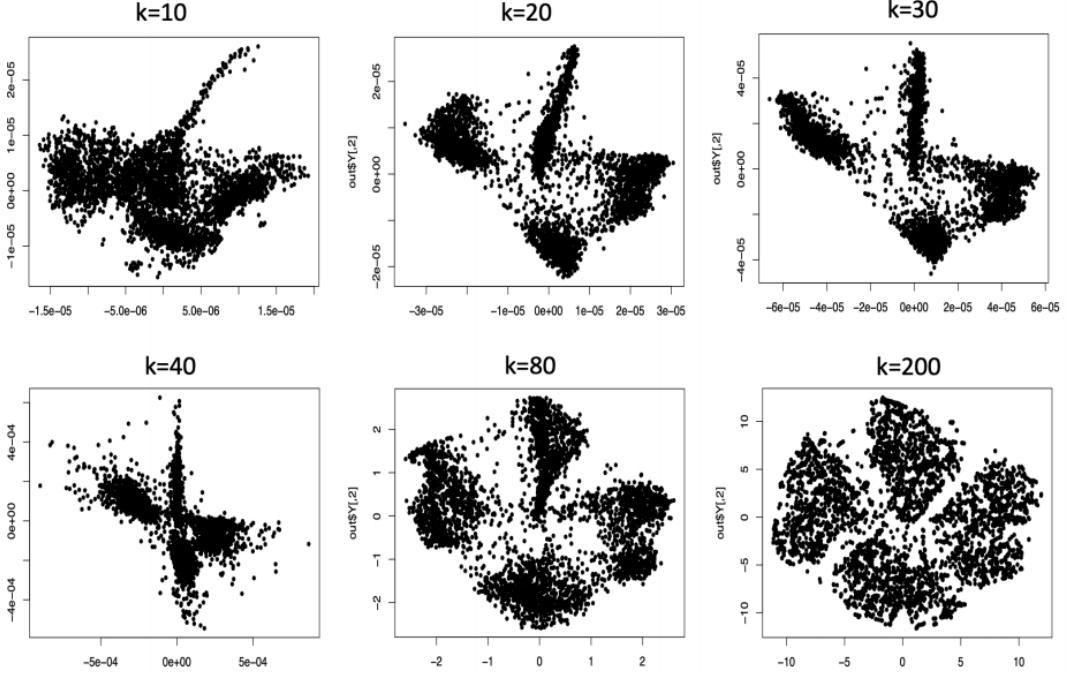
Bước làm quá ban đầu: Với $K_0 > 0$ lần lặp đầu tiên, hệ số p_{ij} trong hệ số gradient D_i^k được nhân với một hệ số làm quá $\alpha > 0$. Khi đó, phương trình cập nhật cho bước làm quá ban đầu trở thành

$$y_i^{(k+1)} = y_i^{(k)} + h \sum_{1 \leq j \leq n, j \neq i} \left(y_j^{(k)} - y_i^{(k)} \right) S_{ij}^{(k)}(\alpha), \quad i = 1, 2, \dots, n.$$

với $S_{ij}^{(k)}(\alpha) = (\alpha p_{ij} - q_{ij}^{(k)}) / (1 + \|y_i^{(k)} - y_j^{(k)}\|_2^2) \in \mathbb{R}$ và hằng số 4 trong D_i^k đã được hấp thụ vào bước tham số độ dài bước h .

Tham số được khuyến cáo nên sử dụng vào năm 2008 bởi van der Maaten và Hinton (2008)[4] là $\alpha = 4$ và $K_0 = 50$, và vào năm 2014 bởi van der Maaten[6] là $\alpha = 12$ và $K_0 = 250$. Bước làm quá ban đầu này giúp t-SNE có thể tìm những cấu trúc mang tính toàn thể ngay trong những bước đầu của tối ưu hóa bởi việc tạo những phân cụm mà có thể di chuyển trong không gian biểu diễn. Tuy nhiên, có rất nhiều câu hỏi liên quan đến việc (i) những nguyên tắc và cách hoạt động của phương pháp tính toán như trên, (ii) giới hạn của không gian đã giảm chiều, (iii), độ nhạy cảm của t-SNE đối với việc hiệu chỉnh các tham số (α, h, K_0) và (iv) cách để lựa chọn các tham số này để đạt được hiệu quả tốt nhất

Bước biểu diễn: Sau bước làm quá ban đầu, tham số làm quá α bị loại bỏ và thuật toán (5) ban đầu được tiếp tục đến khi đạt giới hạn số lần lặp. Đầu ra cuối cùng là một không gian 2 chiều $\{y_i^{K_1}\}_{1 \leq i \leq n}$ - không gian biểu diễn đã giảm chiều của dữ liệu gốc $\{X_i\}_{1 \leq i \leq n}$ mà vẫn giữ được những cấu trúc hình học quan trọng



Hình 2.1: Hình biểu diễn các bước lặp của t-SNE từ tập dữ liệu MNIST. Mỗi mẫu dữ liệu ứng với ảnh viết tay của số 2/4/6/8. Tham số được sử dụng là perplexity = 30, $\alpha = 12, h = 200, K_0 = 40$. Ba hàng đầu ứng với bước làm quá ban đầu, ba hàng cuối ứng với bước biểu diễn.

2.2 PCA

2.2.1 Khái niệm về PCA [1]

Phân tích thành phần chính (Principal Component Analysis - PCA) là một phương pháp giảm số chiều của dữ liệu, giúp tìm ra những thành phần chính (principal components) có thể giải thích tốt nhất sự biến thiên của dữ liệu. PCA thường được sử dụng trong các bài toán xử lý dữ liệu và học máy để loại bỏ các thông tin dư thừa, giảm kích thước dữ liệu nhưng vẫn giữ được thông tin quan trọng.

2.2.2 Mục tiêu và Ý tưởng của PCA

PCA là phương pháp nhằm biến đổi dữ liệu từ không gian ban đầu với số chiều cao xuống một không gian mới có số chiều thấp hơn, sao cho phần lớn phương sai của dữ liệu được bảo toàn. Điều này giúp giữ lại các thông tin quan trọng của dữ liệu mà chỉ cần sử dụng ít chiều hơn, giảm thiểu yêu cầu về lưu trữ và tăng tốc độ tính toán.

- **Mục tiêu:** Tìm một hệ cơ sở mới mà trong đó thông tin quan trọng của dữ liệu tập trung ở một số ít tọa độ nhất định, còn phần còn lại chứa ít thông tin.

Điều này cho phép giảm số chiều của dữ liệu mà vẫn duy trì các đặc điểm quan trọng.

- **Hệ cơ sở mới:** Để đơn giản hóa việc tính toán, PCA sẽ tìm một hệ cơ sở trực chuẩn, tức là các vector trong cơ sở mới này vuông góc với nhau. Giả sử hệ cơ sở trực chuẩn mới là $U = \{u_1, u_2, \dots, u_d\}$.
- **Giảm số chiều:** Để giảm chiều, ta chỉ giữ lại $k < d$ tọa độ quan trọng nhất trong hệ cơ sở mới này. Không mất tính tổng quát, có thể giả sử rằng các tọa độ quan trọng nhất nằm ở k thành phần đầu tiên, tương ứng với k vector đầu tiên trong hệ cơ sở U .

Dữ liệu đầu vào

- **Ma trận dữ liệu X :** Ma trận $X \in \mathbb{R}^{n \times d}$, trong đó:
 - n là số lượng điểm dữ liệu.
 - d là số lượng đặc trưng (số chiều) của mỗi điểm dữ liệu.
 - Mỗi hàng của X là một vector dữ liệu $x_i \in \mathbb{R}^d$, biểu diễn các đặc trưng của điểm dữ liệu thứ i .
- **Số chiều mục tiêu k :** Số chiều k mà ta muốn giảm xuống, với $k < d$. Giá trị k được chọn sao cho phần lớn phương sai của dữ liệu ban đầu vẫn được giữ lại.

Mục tiêu đầu ra

- **Ma trận dữ liệu đã giảm chiều Z :** Ma trận $Z \in \mathbb{R}^{n \times k}$ là kết quả sau khi chiếu dữ liệu X lên không gian mới có số chiều k . Mỗi hàng của Z là một vector $z_i \in \mathbb{R}^k$ biểu diễn dữ liệu ban đầu trong không gian chiều thấp hơn.
- **Ma trận các thành phần chính W :** Ma trận $W \in \mathbb{R}^{d \times k}$, trong đó mỗi cột của W là một vector riêng của ma trận hiệp phương sai, tương ứng với một trong k thành phần chính. Ma trận W được dùng để chiếu dữ liệu ban đầu X lên không gian mới:

$$Z = X'W$$

với X' là dữ liệu đã chuẩn hóa từ ma trận X .

2.2.3 Các bước thực hiện PCA

Các bước chính để thực hiện PCA bao gồm:

1. **Chuẩn hóa dữ liệu:** Đảm bảo rằng mỗi biến có trung bình bằng 0 và phương sai bằng 1. Đối với mỗi đặc trưng x , chuẩn hóa được thực hiện theo công thức:

$$x' = \frac{x - \mu}{\sigma}$$

trong đó μ là giá trị trung bình của đặc trưng và σ là độ lệch chuẩn.

2. **Tính ma trận hiệp phương sai** của dữ liệu: Ma trận hiệp phương sai giúp xác định mối tương quan giữa các đặc trưng. Giả sử ta có tập dữ liệu đã chuẩn hóa với n mẫu và d đặc trưng, ma trận hiệp phương sai C có kích thước $d \times d$ được tính như sau:

$$C = \frac{1}{n-1} X X^T$$

trong đó X là ma trận dữ liệu đã chuẩn hóa kích thước $n \times d$, và X^T là ma trận chuyển vị của X .

- 3. Tính các giá trị riêng và vector riêng của ma trận hiệp phương sai:** Các giá trị riêng và vector riêng sẽ giúp xác định các thành phần chính. Đối với ma trận hiệp phương sai C , tìm các giá trị riêng λ_i và vector riêng v_i thỏa mãn:

$$Cv_i = \lambda_i v_i$$

Vector riêng v_i biểu diễn hướng của thành phần chính, còn λ_i là phương sai theo hướng đó.

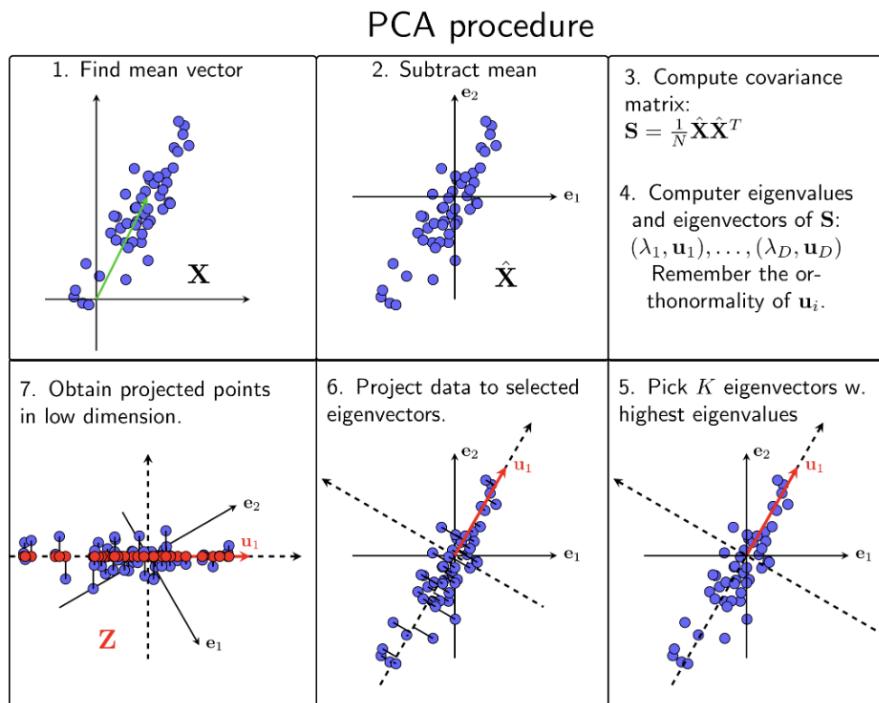
- 4. Sắp xếp các vector riêng theo thứ tự giảm dần của các giá trị riêng:** Các vector riêng được sắp xếp theo thứ tự giảm dần của các giá trị riêng tương ứng. Vector riêng với giá trị riêng lớn nhất là thành phần chính đầu tiên, tiếp theo là thành phần chính thứ hai, và cứ thế cho đến thành phần có phương sai nhỏ nhất.

- 5. Chọn số lượng thành phần chính mong muốn và chiếu dữ liệu gốc lên không gian thành phần chính:** Chọn số lượng thành phần chính k sao cho tổng phương sai dữ liệu được giữ lại đủ lớn (thường là $\geq 90\%$). Sau đó, chiếu dữ liệu gốc X' (dữ liệu đã chuẩn hóa) lên không gian thành phần chính bằng cách sử dụng ma trận W , trong đó W chứa các vector riêng đã chọn:

$$Z = X'W$$

Dữ liệu mới Z có số chiều giảm từ d xuống k .

Dưới đây là ví dụ minh họa về cách áp dụng PCA trên một tập dữ liệu nhỏ. Giả sử chúng ta có một tập dữ liệu gồm hai biến X_1 và X_2 . Ta sẽ thực hiện PCA để tìm thành phần chính và giảm chiều của dữ liệu từ hai chiều xuống một chiều. [1]



Hình 2.2: Trục quan PCA trên đồ thị 2D. [1]

2.2.4 Mối quan hệ giữa PCA và SVD

Phân rã giá trị suy biến (**Singular Value Decomposition - SVD**) là một phương pháp phân rã ma trận quan trọng trong đại số tuyến tính. Trong PCA, SVD được sử dụng như một công cụ tính toán hiệu quả để tìm các thành phần chính.

Phân rã giá trị suy biến (SVD) Cho một ma trận dữ liệu chuẩn hóa $X \in \mathbb{R}^{n \times d}$, SVD phân rã X thành ba ma trận như sau:

$$X = U\Sigma V^T$$

- $U \in \mathbb{R}^{n \times n}$: Ma trận trực giao chứa các vector riêng của XX^T .
- $\Sigma \in \mathbb{R}^{n \times d}$: Ma trận đường chéo chứa các giá trị suy biến (singular values) của X , được sắp xếp theo thứ tự giảm dần.
- $V \in \mathbb{R}^{d \times d}$: Ma trận trực giao chứa các vector riêng của X^TX , tương ứng với các thành phần chính trong PCA.

Vai trò của SVD trong PCA

- Trong PCA, ta cần tìm các thành phần chính bằng cách tính toán ma trận hiệp phương sai $C = \frac{1}{n-1}X^TX$. Tuy nhiên, thay vì tính giá trị riêng của C , ta có thể áp dụng SVD trực tiếp trên ma trận X để tìm các thành phần chính nhanh hơn.
- Các vector trong ma trận V từ SVD chính là các thành phần chính (principal components) của dữ liệu.
- Các giá trị trên đường chéo của ma trận Σ liên quan đến phương sai của dữ liệu dọc theo các thành phần chính.

Kết nối giữa SVD và ma trận hiệp phương sai Khi thực hiện SVD trên X :

$$X = U\Sigma V^T$$

Ta có thể viết lại ma trận hiệp phương sai C như sau:

$$C = \frac{1}{n-1}X^TX = V \left(\frac{\Sigma^T \Sigma}{n-1} \right) V^T$$

- Các giá trị trên đường chéo của $\frac{\Sigma^T \Sigma}{n-1}$ chính là các giá trị riêng của C .
- Các cột của V chính là các vector riêng (thành phần chính).

Lợi ích của SVD trong PCA

- SVD là phương pháp ổn định và hiệu quả hơn so với việc tính giá trị riêng của ma trận hiệp phương sai C , đặc biệt khi số chiều d của dữ liệu lớn.
- SVD có thể áp dụng trực tiếp trên ma trận dữ liệu X mà không cần tính ma trận hiệp phương sai.
- Trong nhiều thư viện như **NumPy**, **scikit-learn**, SVD được sử dụng để thực hiện PCA một cách hiệu quả.

Kết nối giữa: PCA và SVD

- PCA sử dụng giá trị riêng và vector riêng của ma trận hiệp phương sai C .
- SVD là cách tính toán hiệu quả hơn, giúp tìm ra các thành phần chính trực tiếp từ ma trận dữ liệu X .
- Trong thực tế, các thư viện hiện đại thường sử dụng SVD để triển khai PCA nhằm tăng tốc độ và độ ổn định khi xử lý dữ liệu lớn.

2.3 Hàm xác suất phân lớp Softmax

Trong bài toán phân loại nhiều lớp, ta cần tìm hàm số $g_k(\mathbf{x}, \boldsymbol{\theta})$ thể hiện xác suất đầu vào \mathbf{x} thuộc vào lớp thứ k . Cụ thể, hàm số cần thỏa mãn các điều kiện:

- $g_k(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R} \rightarrow (0, 1]$
 - Với \mathbf{x} cụ thể: $\sum_{k=1}^C g_k(\mathbf{x}, \boldsymbol{\theta}) = 1$, ở đây C là số lớp.
 - Hàm số đủ trơn và đạo hàm tính được dễ dàng.
 - Với \mathbf{x} cụ thể: $\boldsymbol{\theta}_k^T \bar{\mathbf{x}} > \boldsymbol{\theta}_h^T \bar{\mathbf{x}} \implies g_k(\mathbf{x}, \boldsymbol{\theta}) > g_h(\mathbf{x}, \boldsymbol{\theta})$.
- Ở đây, $\boldsymbol{\theta}_k = (\theta_{0,k} \ \theta_{1,k} \dots \theta_{d,k})^T$ với $k \in \{1, 2, \dots, C\}$ và $\bar{\mathbf{x}} = (1 \ \mathbf{x})^T$.

Để thỏa mãn các điều kiện trên, chúng ta sử dụng hàm số có dạng:

$$g_k(\mathbf{x}, \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}_k^T \bar{\mathbf{x}})}{\sum_{j=1}^C \exp(\boldsymbol{\theta}_j^T \bar{\mathbf{x}})}$$

Đặt $z_k = \boldsymbol{\theta}_k^T \bar{\mathbf{x}}$. Ta có hàm Softmax được xây dựng trên tập $\{z_1, z_2, \dots, z_C\}$ xác định như sau:

$$g(z_k) = \frac{\exp(z_k)}{\sum_{j=1}^C \exp(z_j)}$$

Hàm Softmax ổn định: Khi z_i quá lớn, việc tính toán $\exp(z_i)$ có thể gây ra hiện tượng tràn số. Ta khắc phục bằng cách sử dụng phiên bản ổn định hơn của Softmax:

$$g(z_k) = \frac{\exp(z_k - M)}{\sum_{j=1}^C \exp(z_j - M)}; \text{ với } M \text{ là hằng số bất kỳ.}$$

Trong thực nghiệm, M thường được chọn là: $M = \max(z_i)$.

2.4 Phương pháp Softmax Regression

Mô hình Softmax Regression (hay Logistic Regression cho nhiều lớp) là một mô hình phân loại trong học máy. Mô hình này có kiến trúc đơn giản nhưng vẫn hiệu quả và được sử dụng nhiều trong các bài toán phân loại. Softmax Regression thuộc loại mô hình phân biệt và được xếp vào nhóm mô hình tuyến tính (linear model).

Ngày nay, với sự phát triển mạnh mẽ của các mô hình học sâu nhưng Softmax Regression vẫn là một mô hình quan trọng, đặc biệt trong các ngành kinh tế và ngân hàng.

2.4.1 Khởi tạo và hướng đi

Các biến quan sát \mathbf{x} thuộc không gian d chiều : $\mathbf{x} \in \mathbf{X} \subset \mathbb{R}^d$ và $\bar{\mathbf{x}} = (1 \ \mathbf{x})^T$. Nhãn tương ứng là y , với $y \in \{1, 2, \dots, C\}$.

Giả sử ta nâng cấp Logistic Regression để áp dụng cho phân loại nhiều lớp theo cách tiếp cận one-vs-rest. Khi đó, với lớp thứ k ta cần tìm một vector $\boldsymbol{\theta}_k$ để hình thành một siêu phẳng $\boldsymbol{\theta}_k^T \bar{\mathbf{x}} = 0$ sao cho:

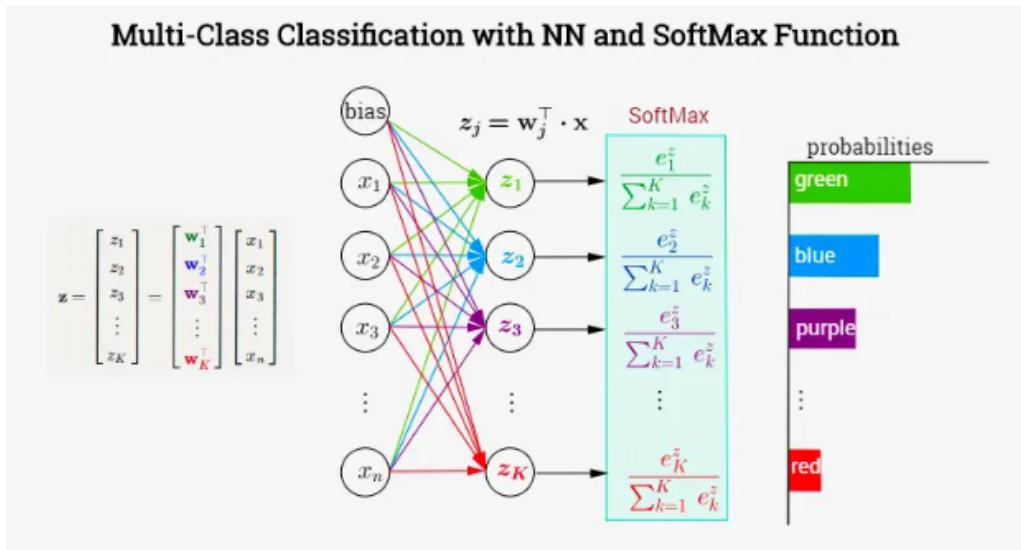
- $\forall u \in \text{class } k^{th} \implies \boldsymbol{\theta}_k^T \bar{\mathbf{u}} > 0$.
- $\forall v \notin \text{class } k^{th} \implies \boldsymbol{\theta}_k^T \bar{\mathbf{v}} < 0$.

Do đó, với C lớp chúng ta cần C vector tham số $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C$. Như vậy, các tham số tạo thành một ma trận $\boldsymbol{\theta} \in \mathbb{R}^{(d+1) \times C}$:

$$\begin{pmatrix} \theta_{0,1} & \theta_{0,2} & \dots & \theta_{0,C} \\ \theta_{1,1} & \theta_{1,2} & \dots & \theta_{1,C} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d,1} & \theta_{d,2} & \dots & \theta_{d,C} \end{pmatrix}$$

Phương pháp Softmax cũng sử dụng ma trận tham số $\boldsymbol{\theta}$ tương tự. Cột thứ k của ma trận ứng với lớp k và được sử dụng để tính giá trị cho z_k , qua đó tính toán được xác suất một điểm dữ liệu thuộc về lớp thứ k .

2.4.2 Kiến trúc mô hình Softmax Regression



Hình 2.3: Mô hình Softmax Regression.

Ở hình 2.3, mô hình gồm 2 tầng chính:

- **Tầng đầu vào:** Mỗi nút của tầng này là một đặc trưng của dữ liệu đầu vào \mathbf{x} và hệ số bias.
- **Tầng đầu ra:**
 - Các nút của tầng này nhận đầu vào từ tầng đầu vào và tính toán các giá trị z_j thông qua phép nhân ma trận trọng số $\boldsymbol{\theta}$ với đầu vào.

- Sau đó áp dụng hàm Softmax lên các giá trị z_j để mô hình đưa ra xác suất thuộc về từng lớp của \mathbf{x} .

2.4.3 Hàm mất mát của Softmax Regression

Ta có tập huấn luyện: (X, Y) . Trong đó:

- $X = \{\mathbf{x}_n\}_{n=1}^N$; $\mathbf{x}_n = (x_{1,n} \ x_{2,n} \dots x_{d,n})^T \in \mathbb{R}^d$.
- $Y = \{y_n\}_{n=1}^N$; $y_n \in \{1, 2, \dots, C\}$.

Bài báo cáo này sử dụng hàm mất mát được xây dựng thông qua sử dụng ước lượng hợp lý cực đại (MLE) - hàm log-likelihood. Hàm mất mát như sau:

$$J(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^N \left\{ \boldsymbol{\theta}_{y_n}^T \mathbf{x}_n - \log \left[\sum_{k=1}^C \exp (\boldsymbol{\theta}_k^T \mathbf{x}_n) \right] \right\}$$

Trường hợp sử dụng hiệu chỉnh L_2 , chúng ta có hàm mất mát:

$$J(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^N \left\{ \boldsymbol{\theta}_{y_n}^T \mathbf{x}_n - \log \left[\sum_{k=1}^C \exp (\boldsymbol{\theta}_k^T \mathbf{x}_n) \right] \right\} + \frac{\lambda}{2} \sum_{k=1}^C \sum_{j=1}^d \theta_{j,k}^2 \quad (2.1)$$

2.4.4 Tối ưu hàm mất mát

Với mỗi điểm dữ liệu (x_n, y_n) ta quy ước:

- $\bar{\mathbf{y}}_n$ là vector one-hot-coding của y_n .
- $p_{k,n} := P(y_n = k \mid \mathbf{x}_n; \boldsymbol{\theta}) = g(\boldsymbol{\theta}_k^T \bar{\mathbf{x}}_n)$ là xác suất điểm dữ liệu thuộc lớp thứ k theo mô hình.

Dặt $e_{k,n} := \bar{y}_{k,n} - p_{k,n}$, là sai số giữa phân lớp theo dữ liệu quan sát và xác suất phân lớp theo mô hình. Như vậy, $\mathbf{e}_n := \bar{\mathbf{y}}_n - \mathbf{p}_n$ là vector sai số giữa one-hot-coding của nhãn và vector xác suất phân lớp của mô hình tương ứng với \mathbf{x}_n .

Để tối ưu hóa $J(\boldsymbol{\theta})$ (áp dụng hiệu chỉnh L_2) bằng phương pháp Stochastic Gradient Descent (SGD), ta thực hiện các bước sau:

1. Khởi tạo: Chọn $\boldsymbol{\theta} \in \mathbb{R}^{(d+1) \times C}$ với các giá trị ngẫu nhiên nhỏ, tham số học $\eta > 0$, điều kiện dừng ($\Delta \boldsymbol{\theta}$ đủ nhỏ, giá trị hàm mất mát đủ nhỏ hoặc số vòng lặp đủ lớn).

2. Lặp lại cho từng epoch:

2.1 Tráo đổi ngẫu nhiên thứ tự các cặp dữ liệu (x_n, y_n) trong tập huấn luyện.

2.2 Lặp lại lần lượt cho từng cặp dữ liệu (x_n, y_n) :

- Tính vector xác suất phân lớp đầu ra:

$$\mathbf{p}_n = (p_{1,n} \ p_{2,n} \ \dots \ p_{C,n})^T$$

- Tính gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial (\boldsymbol{\theta})} = -\mathbf{x}_n \mathbf{e}_n^T + \lambda \boldsymbol{\theta}'$$

$$\text{Trong đó } \boldsymbol{\theta}' = \begin{pmatrix} 0 & 0 & \dots & 0 \\ \theta_{1,1} & \theta_{1,2} & \dots & \theta_{1,C} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d,1} & \theta_{d,2} & \dots & \theta_{d,C} \end{pmatrix}$$

iii. Cập nhật tham số:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \cdot \frac{\partial J(\boldsymbol{\theta})}{\partial (\boldsymbol{\theta})}$$

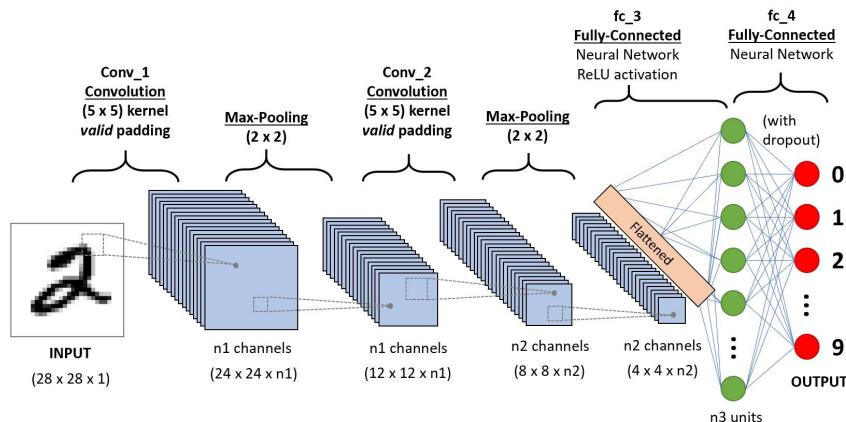
(a) Nếu đạt đủ điều kiện dừng, kết thúc. Ngược lại quay lại bước 2.

2.5 Convolutional Neural Network

Convolutional Neural Network(CNN) hay Mạng neuron tích chập là một trong những mô hình được sử dụng phổ biến trong lĩnh vực học sâu. CNN là một mô hình học sâu mạnh mẽ, thường được sử dụng trong xử lý ảnh, có khả năng tự động trích xuất đặc trưng mà không yêu cầu trích xuất thủ công.

2.5.1 Cấu trúc cơ bản của CNN

Cấu trúc cơ bản của CNN gồm các lớp được sắp xếp tuần tự, minh họa trong Hình 2.4.



Hình 2.4: Minh họa cấu trúc CNN

- Tầng Convolution
- Tầng Pooling
- Tầng Flatten
- Tầng Fully-connected

Tầng Convolution là tầng đầu tiên của CNN, theo sau đó là các tầng Convolution tiếp theo hoặc lớp Pooling. Tầng Flatten sẽ là tầng trước tầng cuối cùng là tầng Fully-connected. Với các lớp đầu, CNN sẽ trích xuất ra các đặc trưng đơn giản và dần dần về sau, CNN sẽ tăng tính phức tạp, nhận diện các yếu tố lớn hơn đến khi nhận diện được đối tượng theo dự định. [7]

Ở đây chúng tôi sẽ giới thiệu thêm 2 loại tầng giúp tránh hiện tượng Overfit bao gồm tầng **Dropout** và tầng **Batch Normalization**.

2.5.2 Tầng Convolution

Tầng Convolution là linh hồn của mô hình CNN. Tầng Convolution thực hiện phép tích chập giữa dữ liệu đầu vào (ví dụ: ảnh) và các bộ lọc học được nhằm trích xuất đặc trưng quan trọng.

Các **thành phần** của tầng Convolution:

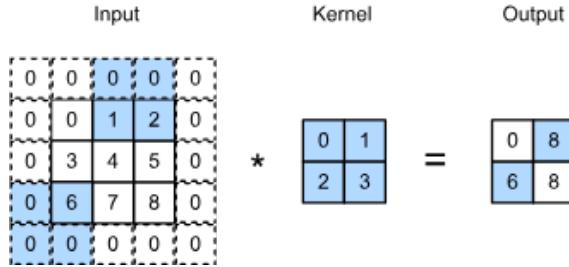
- **Dữ liệu đầu vào:** Thường là ảnh biểu diễn dưới dạng ma trận với kích thước là (Height X Width X Channel) với Height là chiều dài, Width là chiều rộng và Channel là số kênh màu. Ví dụ, ảnh có một kênh màu đại diện cho ảnh xám, trong khi ảnh có ba kênh màu tương ứng với ảnh RGB (Đỏ-Xanh lá-Xanh dương).
- **Bộ lọc(Filter):** Là một ma trận với kích thước nhỏ, các giá trị trong ma trận được khởi tạo ngẫu nhiên hoặc được học qua quá trình huấn luyện. Có nhiều dạng bộ lọc, mỗi dạng tập trung vào một đặc điểm của ảnh ví dụ như màu sắc, góc cạnh, v.v.
- **Phép toán tích chập:** Phép toán tích chập được thực hiện thông qua việc trượt bộ lọc từ trái qua phải, từ trên xuống dưới, sau đó tính toán phép nhân từng phần tử và lấy tổng giữa ma trận tại vị trí đang trượt và bộ lọc. Kết quả của phép toán tích chập là một hình ảnh (thường có kích cỡ bé hơn), gồm các đặc trưng trích xuất ra, còn được gọi là ảnh đặc trưng (feature map).

Các **siêu tham số** của tầng Convolution:

- **Số lượng bộ lọc:** Số bộ lọc được sử dụng để trích xuất đặc trưng của ảnh. Càng nhiều bộ lọc thì càng nhiều đặc trưng được trích xuất.
- **Kích thước bộ lọc:** Chiều dài và chiều rộng của bộ lọc. Kích cỡ thường được sử dụng là (3 x 3).
- **Bước nhảy(Stride):** Khoảng cách giữa các lần trượt của bộ lọc trên ảnh đầu vào. Mặc định bước nhảy = 1, tuy nhiên có thể sử dụng bước nhảy lớn hơn để có ít lần trượt hơn và giảm bớt độ phức tạp tính toán.
- **Padding:** Khi áp dụng tầng Convolution thì sẽ xảy ra hiện tượng không sử dụng các pixels ở rìa của ảnh. Vì vậy, Padding là những giá trị được thêm vào xung quanh rìa của ảnh (thường bằng 0) để kiểm soát kích thước của ảnh đặc trưng đầu ra.

Với $n_h \times n_w$ là kích thước ảnh dữ liệu, $k_h \times k_w$ là kích thước của bộ lọc, (s_h, s_w) là bước nhảy cho chiều dài và chiều rộng, (p_h, p_w) là số hàng padding cho chiều dài và chiều rộng. **Kích thước của đầu ra** sẽ là:

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor.$$



Hình 2.5: Minh họa về tầng Convolution

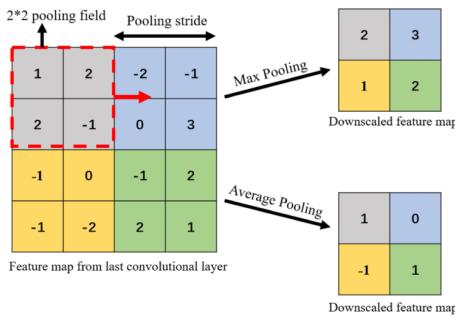
2.5.3 Tầng Pooling

Tầng Pooling dùng để giảm chiều dữ liệu, giảm số lượng tham số của đầu vào. Tương tự phép nhân tích chập, thao tác Pooling thực hiện việc trượt một bộ lọc qua các vị trí của đầu vào, nhưng khác biệt ở đây là bộ lọc này không có tham số. Thay vào đó, bộ lọc áp dụng một hàm có chức năng tổng hợp các giá trị trong vùng đang trượt qua, điền ra mảng đầu ra. Có hai loại chính của pooling:

- **Max Pooling:** Trong vùng mà bộ lọc đang trượt qua, bộ lọc sẽ chọn ra pixel có giá trị lớn nhất để ghi ra mảng đầu ra.
- **Average Pooling:** Trong vùng mà bộ lọc đang trượt qua, bộ lọc sẽ lấy ra giá trị trung bình của các pixels để ghi ra mảng đầu ra.

Thường thì Max Pooling sẽ được ưu tiên hơn Average Pooling. Các **siêu tham số** của Tầng Pooling bao gồm:

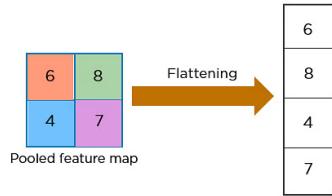
- **Kích thước bộ lọc:** Chiều cao và chiều rộng của bộ lọc. Kích cỡ thường được dùng là (2×2)
- **Bước nhảy:** Khoảng cách giữa các lần trượt của bộ lọc
- **Padding:** Tương tự Padding ở tầng Convolution.



Hình 2.6: Minh họa về tầng Pooling

2.5.4 Tầng Flatten

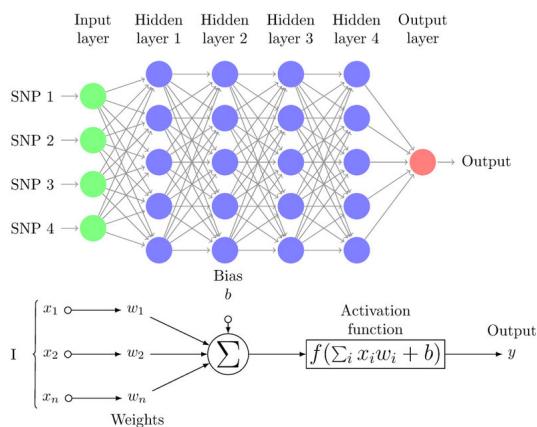
Tầng Flatten đơn giản là làm phẳng(flatten) một ma trận về dạng vector 1 chiều. Kết quả của tầng Flatten sẽ được sử dụng làm đầu vào cho tầng Fully-connected.



Hình 2.7: Minh họa tầng Flatten

2.5.5 Tầng Fully-connected

Tầng này sử dụng cấu trúc của Multi-layer Perceptron hoặc Dense Neural Networks, với tất cả các node trong mạng đều được liên kết với nhau. Đầu vào là kết quả của việc trích xuất đặc trưng thông qua các tầng Convolution, Pooling và Flatten. Trong các tầng Hidden của Multi-layer Perceptron, hàm kích hoạt được áp dụng để tạo ra tính phi tuyến. Kết quả đầu ra cuối cùng của tầng Fully Connected sẽ được đưa qua một hàm kích hoạt phù hợp để xác định xác suất dữ liệu đầu vào thuộc về một lớp.

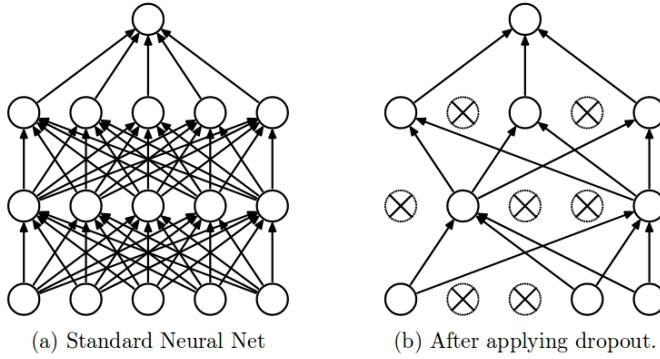


Hình 2.8: Minh họa tầng Fully Connected

Hàm kích hoạt Sigmoid thường được sử dụng trong bài toán phân loại nhị phân, trong khi Softmax được sử dụng cho bài toán phân loại nhiều lớp. Với L là số layer của Multi-layers Perceptron, số layer của Fully-connected bằng $L+1$ với 1 layer cuối dành cho xác suất cho dữ liệu đầu vào (Số layer của Multi-layers Perceptron không tính layer đầu vào).

2.5.6 Tầng Dropout[2]

Đối với các mô hình Multi-layers Perceptron gồm nhiều hidden layers, các node sẽ có tính chất liên kết đầy đủ - tất cả các node đều liên kết với nhau như hình 2.8, điều này giúp cho mô hình học được những mối quan hệ giữa đầu ra và đầu vào. Tuy nhiên, tính chất này kết hợp với việc thiếu dữ liệu để huấn luyện mô hình dẫn đến việc mô hình học được những **mối quan hệ được đút kết từ những dữ liệu nhiễu**. Những mối quan hệ nhiễu đó chỉ xuất hiện ở tập dữ liệu Training, nhưng sẽ không xuất hiện ở dữ liệu Test. Điều này dẫn đến tình trạng **Overfitting** và có nhiều phương pháp hiệu chỉnh được áp dụng: dừng việc huấn luyện khi kết quả của tập Validation bị giảm sút, thêm tham số hiệu chỉnh L1 hoặc L2.



Hình 2.9: Bên trái là mạng MLP gốc với 2 tầng Hidden. Bên phải là ví dụ của mạng MLP mới sau khi áp dụng Dropout. Các node bị gạch X là bị loại bỏ

Cách tốt nhất để hiệu chỉnh một mô hình là **lấy trung bình các dự đoán của tất cả các bộ tham số**, ứng với mỗi một bộ tham số là một xác suất hậu nghiệm dựa trên dữ liệu Training. Cách này tuy phù hợp với mô hình nhỏ và đơn giản nhưng sẽ tốn nhiều thời gian và bộ nhớ để chạy tất cả các bộ đó.

Dropout là một kĩ thuật giải quyết cả 2 vấn đề là Overfitting và tối ưu hóa việc lấy trung bình của nhiều bộ tham số. Thuật ngữ "Dropout" chỉ đến việc **bỏ bớt các nodes** trong các layer(ví dụ của một MLP). Khi đó, các **kết nối** từ trước và từ sau trở về node đấy cũng sẽ bị **loại bỏ tạm thời**, tạo nên một cấu trúc MLP mới từ cấu trúc MLP gốc. Các node bị loại bỏ **dựa trên xác suất dropout p**.

Áp dụng dropout cho một MLP (Multi-Layer Perceptron) đồng nghĩa với việc "lấy mẫu" một phiên bản "thưa" của MLP gốc, gồm những nodes không bị loại bỏ trong quá trình dropout. Với một MLP gồm n nodes, có thể tạo ra tối đa 2^n phiên bản MLP thưa. Tuy nhiên, tất cả các phiên bản MLP này đều chia sẻ chung trọng số, nên tổng số lượng tham số vẫn là $O(n^2)$. Quá trình huấn luyện MLP khi áp dụng dropout thực chất là: Lấy mẫu một MLP thưa (bằng cách loại bỏ ngẫu nhiên một số nodes trong mạng), huấn luyện dựa trên phiên bản thưa đó.

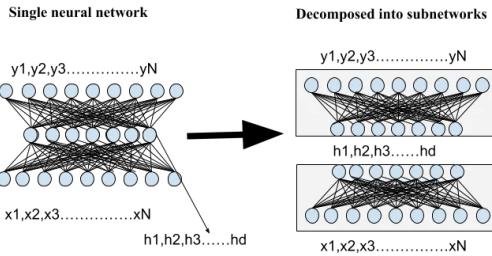
Khi một mô hình gặp vấn đề Overfitting, mô hình đó đã lỡ học được những thông tin gây nhiễu. Vì bản chất quá trình huấn luyện là giảm hàm mất mát, ứng với tất cả các nodes. Khi overfitting xảy ra, một node bất kỳ có thể thay đổi để bù đắp sai sót của các node khác. Điều này khiến mô hình **mất đi khả năng dự đoán trên dữ liệu mới**.

Bằng cách vô hiệu hóa ngẫu nhiên một số neuron trong mỗi lần cập nhật, dropout làm cho sự hiện diện của từng neuron trở nên bớt tin cậy. Điều này buộc các tầng mạng phải hoạt động độc lập hơn, phân bổ trách nhiệm xử lý đầu vào một cách đồng đều hơn. Kết quả là mô hình ít phụ thuộc vào một số mối liên kết cụ thể và **có khả năng tổng quát hóa tốt hơn**.

Một lưu ý là khi bắt đầu quá trình dự đoán(sau khi đã huấn luyện mô hình với dropout), mô hình để dự đoán sẽ không sử dụng tầng dropout. Nhưng vì chúng ta lấy tất cả nodes từ một tầng, trọng số cuối cùng sẽ lớn hơn dự kiến. Để giải quyết vấn đề này, trọng số sẽ được chuẩn hóa bởi xác suất dropout p . Tức là, với một node có trọng số w được giữ lại sau Dropout, trọng số cuối cùng để dự đoán là $w \times p$.

2.5.7 Tầng Batch Normalization[3]

Batch Normalization đại diện cho việc **chuẩn hóa các giá trị kích hoạt** của các node ở tầng Hidden, sao cho **phân phối của các node đấy là như nhau** trong quá trình training. Ví dụ với việc huấn luyện một mô hình MLP, nếu phân phối của các giá trị kích hoạt ở tầng hiện tại bị thay đổi(do trọng số và bias), phân phối của tầng tiếp theo cũng sẽ thay đổi(do output của tầng trước là input của tầng sau). Sự thay đổi của phân bố những giá trị kích hoạt Hidden còn được gọi là "**internal covariate shift**" .



Hình 2.10: Chia MLP thành nhiều mạng con

Như hình 2.10 ở bên trái, trước hết ta sẽ áp dụng chuẩn hóa đầu vào $[x_1, x_2, \dots, x_n]$ trước khi huấn luyện mô hình. Nếu chúng ta chia một MLP gốc thành nhiều mạng con thì việc áp dụng Batch Normalization tức là chuẩn hóa giá trị kích hoạt $[h_1, h_2, \dots, h_n]$ từ tầng Hidden 1 để làm đầu vào cho tầng Hidden 2

Việc chuẩn hóa một dữ liệu x bất kì tức là đưa dữ liệu về trung bình = 0 và phương sai = 1. Với số lượng node trong một tầng Hidden là d và giá trị kích hoạt của tầng đó là $x = [x_1, x_2, \dots, x_d]$, ta có công thức chuẩn hóa

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

với $\hat{x}^{(k)}$ là giá trị chuẩn hóa của node thứ k trong tầng Hidden đang xét, $\mathbb{E}[x^{(k)}]$ là trung bình của node thứ k, $\text{Var}[x^{(k)}]$ là phương sai của node thứ k.

Tuy nhiên, chúng ta không cần những node này có trung bình = 0 và phương sai = 1 mà ta **muốn mô hình tự học những giá trị này**.

Vì vậy chúng ta sẽ khai báo 2 tham số mới γ, β để học các giá trị trung bình và phương sai. Những tham số này được học và cập nhật cùng lúc với trọng số và bias trong quá trình huấn luyện. Công thức cuối cùng cho giá trị đã được chuẩn hóa của việc kích hoạt node thứ k là:

$$y^k = \gamma^k \times \hat{x}^k + \beta^k$$

Vì quá trình huấn luyện sẽ **không sử dụng toàn bộ dữ liệu** mà sẽ chỉ **tối ưu với một phần nhỏ của dữ liệu**(mini-batch) để cập nhật tham số với giá trị loss từ batch đó. Batch Normalization cũng sẽ chỉ áp dụng với trung bình và phương sai của batch đó.

Algorithm 1 Quá trình huấn luyện với Batch Normalization và Mini-Batch B

Dầu vào: Giá trị Mini-Batch $B = \{x_1, \dots, x_m\}$
Dầu vào: Tham số γ, β
Dầu ra: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

- 1: Tính trung bình của Mini-Batch: $\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$
- 2: Tính phương sai của Mini-Batch: $\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$
- 3: **for** $i = 1, \dots, m$ **do**
- 4: Chuẩn hóa: $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
- 5: Điều chỉnh tỷ lệ và dịch chuyển: $y_i \leftarrow \gamma \hat{x}_i + \beta$
- 6: **end for**

Tuy nhiên, khi dự đoán trên tập Test, ta **không áp dụng Batch Normalization như lúc huấn luyện** được vì chúng ta sẽ chỉ xét một mẫu tại một lần dự đoán(việc tính trung bình và phương sai một mẫu là không cần thiết). Vì vậy, chúng ta sẽ lưu lại trung bình và phương sai của node thứ k lúc huấn luyện và sử dụng các giá trị đó kết hợp với tham số γ, β của Batch Normalization khi dự đoán. Các bước cho quá trình dự đoán như sau:

Algorithm 2 Quá trình dự đoán với Batch Normalization

- 1: **for** $k = 1 \dots K$ **do**
- 2: // Ký hiệu: $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_B \equiv \mu_B^{(k)}, \dots$
- 3: // Trung bình và phương sai trên các mini-batch:
- 4: $\mathbb{E}[x] \leftarrow \mathbb{E}_B[\mu_B]$ // Kỳ vọng trung bình
- 5: $\text{Var}[x] \leftarrow \frac{m}{m-1} \mathbb{E}_B[\sigma_B^2]$ // Phương sai
- 6: // Phép biến đổi y trong N_{BN}^{inf} :
- 7: $y \leftarrow \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathbb{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}} \right)$
- 8: **end for**

2.6 Độ đo đánh giá kết quả

2.6.1 Ma trận Confusion

Ma trận mẫu thuẫn (Confusion Matrix) Là một phương pháp đánh giá kết quả của những bài toán phân loại với việc xem xét cả những chỉ số về độ chính xác và độ bao quát của các dự đoán cho từng lớp. Một confusion matrix gồm 4 chỉ số sau đối với mỗi lớp phân loại:

- TP (True Positive): Số lượng dự đoán dương tính đúng.
- TN (True Negative): Số lượng dự đoán âm tính đúng.
- FP (False Positive - Type 1 Error): Số lượng các dự đoán dương tính sai.
- FN (False Negative - Type 2 Error): Số lượng các dự đoán âm tính sai.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Hình 2.11: Ma trận nhầm lầm cho từng lớp.

2.6.2 Accuracy

Dộ chính xác Accuracy dùng để tính toán tỉ lệ dự đoán đúng trên tổng số mẫu dự đoán là bao nhiêu. Chỉ số này càng cao tức là mô hình dự đoán càng đúng và càng tốt (đối với dữ liệu không quá mất cân bằng).

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

2.6.3 Precision

Precision cũng là một thang đo độ chính xác. Cụ thể trong tất cả các dự đoán dương tính được đưa ra, tỷ lệ chính xác là bao nhiêu? Chỉ số này rất quan trọng trong các bài toán yêu cầu độ chính xác cao khi đưa ra dự đoán dương tính. Chỉ số này được tính theo công thức:

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

2.6.4 Recall

Dộ đo Recall (còn được gọi là sensitivity hoặc true positive rate) là một độ đo đánh giá khả năng của một mô hình hoặc hệ thống trong việc bắt tất cả các trường hợp thực sự thuộc một lớp cụ thể. Độ đo này quan tâm đến khả năng của mô hình trong việc không bỏ sót các trường hợp dương tính trong thực tế. Công thức tính của Recall:

$$\text{Precision} = \frac{TP}{TP + FN}$$

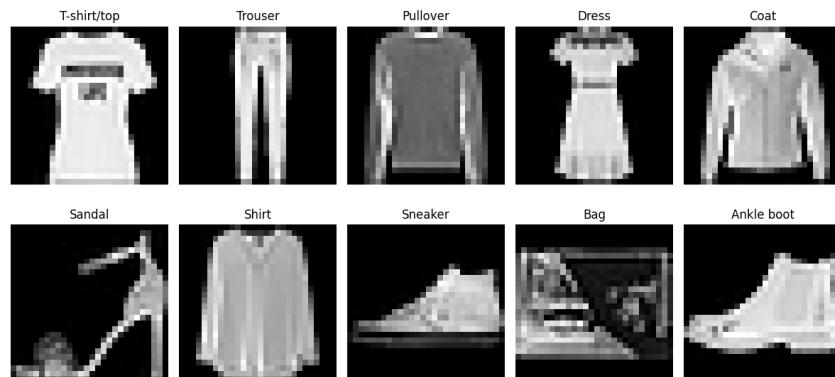
Chương 3

Thực nghiệm - Đánh giá

3.1 Dữ liệu sử dụng

Dữ liệu sử dụng cho các mô hình đã được giới thiệu là bộ **FashionMNIST**. Các đặc điểm của bộ dữ liệu bao gồm:

- 60000 ảnh tập training, 10000 ảnh tập test
- Mỗi ảnh là một ảnh xám kích thước $28 \times 28 \times 1$ (Vì là ảnh xám nên giá trị 1 ở cuối tượng trưng cho 1 kênh màu) với giá trị của từng pixel là số nguyên dương trong khoảng từ 0 đến 255
- Dữ liệu có 10 nhãn
- Được coi là sự thay thế cho bộ dữ liệu MNIST ban đầu, được dùng cho việc kiểm thử các thuật toán học máy



Hình 3.1: Ảnh thực tế của bộ dữ liệu, mỗi ảnh ứng với một nhãn

Bộ dữ liệu này đã được chia thành 3 tập Training, Validation, Test với số lượng mẫu tương ứng là 48000, 12000, 10000. Bảng 3.1 dưới đây thống kê về số lượng mẫu của các nhãn trong 3 tập nói trên:

Vì giá trị của từng pixel nằm trong khoảng từ 0 đến 255 nên ta cần chuẩn hóa, đưa về khoảng từ 0.0 đến 1.0. Kết quả việc **chuẩn hóa dữ liệu** nằm ở bảng 3.2

Nhãn	Training	Validation	Test
T-shirt/top	4800	1200	1000
Trouser	4800	1200	1000
Pullover	4800	1200	1000
Dress	4800	1200	1000
Coat	4800	1200	1000
Sandal	4800	1200	1000
Shirt	4800	1200	1000
Sneaker	4800	1200	1000
Bag	4800	1200	1000
Ankle boot	4800	1200	1000

Bảng 3.1: Bảng thống kê dữ liệu FashionMNIST

	Maximum	Minimum
Trước chuẩn hóa	255	0
Sau chuẩn hóa	1.0	0.0

Bảng 3.2: Giá trị trước và sau chuẩn hóa

Kiểm tra xem có ảnh nào có giá trị NaN, tức là ảnh lỗi không thì trong cả ba phần dữ liệu Training, Validation, Test đều không có ảnh lỗi. Đồng thời, cột mục tiêu để phân loại cũng không có giá trị NaN. Kết quả của việc kiểm tra được minh họa qua bảng 3.3

Bộ dữ liệu	Ma trận dữ liệu(X)	Biến mục tiêu(y)
Train	Không có giá trị NaN	Không có giá trị NaN
Valid	Không có giá trị NaN	Không có giá trị NaN
Test	Không có giá trị NaN	Không có giá trị NaN

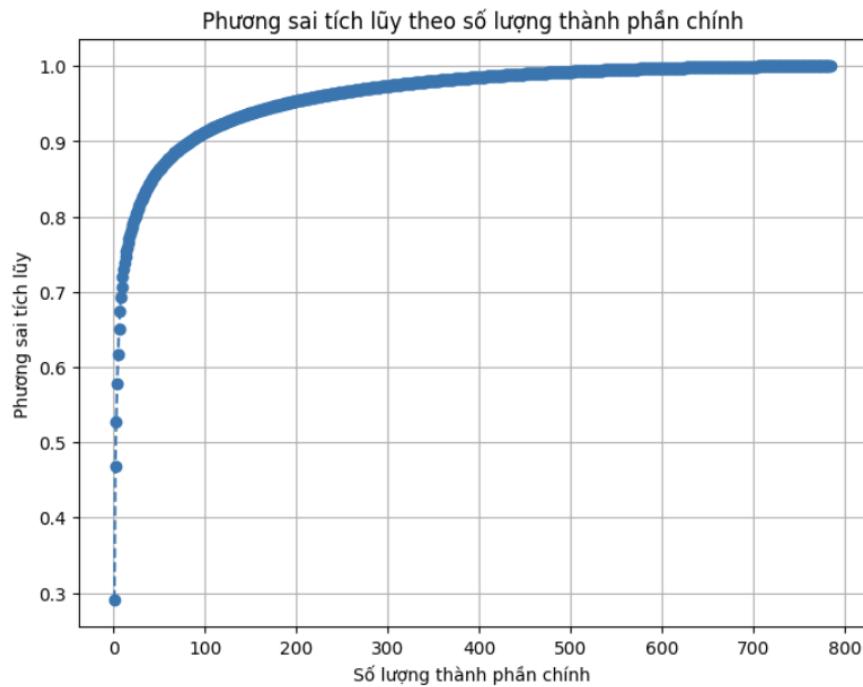
Bảng 3.3: Kiểm tra giá trị NaN trong bộ dữ liệu

Để phục vụ cho các mô hình phân loại như Logistic Regression và Support Vector Machine, từ ảnh kích thước $28 \times 28 \times 1$ chúng tôi sẽ làm phẳng về kích thước 1×784 với 784 cột. Khi đó, bộ dữ liệu được làm phẳng sẽ có kích thước (*Num_Samples*) \times 784 với *Num_Samples* là số mẫu có trong bộ dữ liệu tương ứng.

3.2 Áp dụng các phương pháp giảm chiều và minh họa

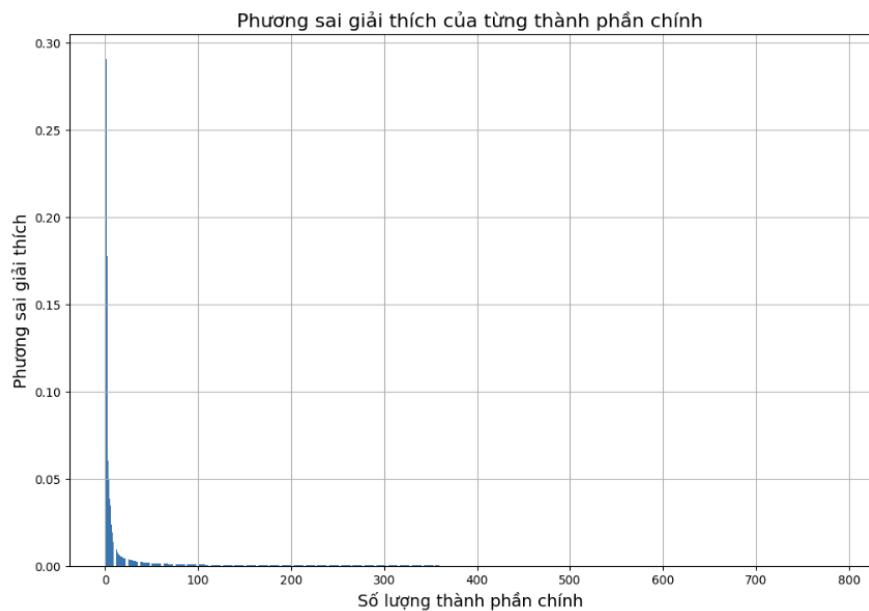
3.2.1 PCA

1. **Phương sai tích lũy của các thành phần:** Dựa trên thực nghiệm PCA trên 48,000 mẫu dữ liệu từ bộ ta có biểu đồ **biểu đồ phương sai tích lũy:**



Hình 3.2: Biểu đồ phương sai tích lũy.

- Đoạn đầu của đường cong, nơi phương sai tích lũy tăng nhanh, cho thấy các thành phần chính đầu tiên đã nắm bắt được phần lớn biến động trong dữ liệu (khoảng 100 thành phần chính đầu tiên).
- Điều này thường xảy ra vì các thành phần chính đầu tiên thường liên quan đến các đặc trưng chung của toàn bộ bộ dữ liệu.



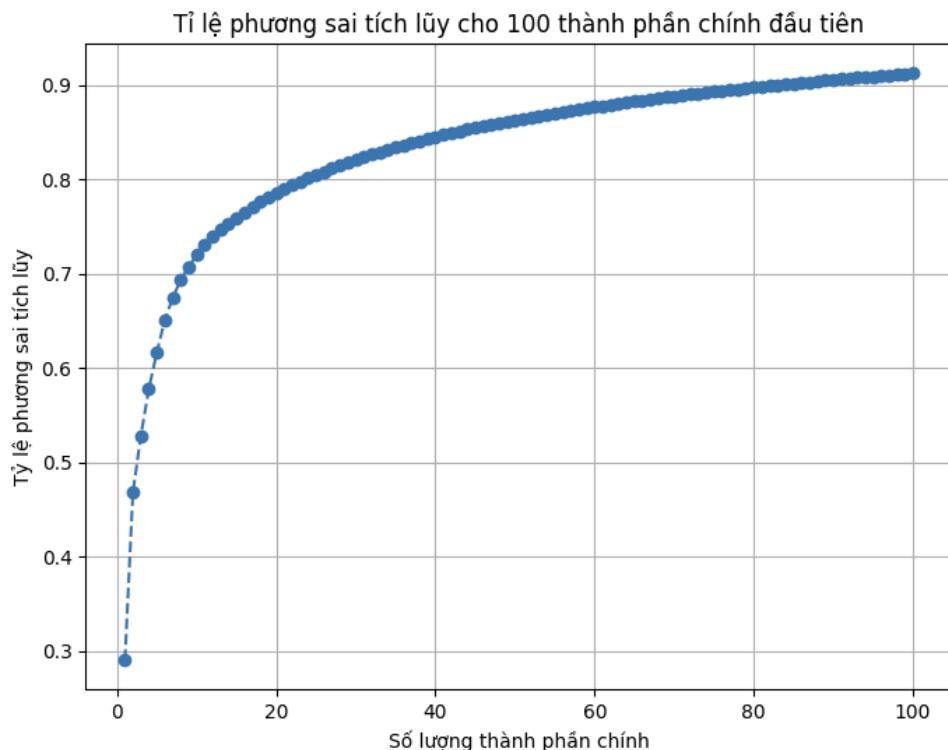
Hình 3.3: Biểu đồ phương sai giải thích.

- Giảm nhanh: Chúng ta thấy ngay từ những thành phần chính đầu tiên, phương

sai giải thích đã giảm rất nhanh. Điều này cho thấy các thành phần chính đầu tiên đã nắm bắt được phần lớn thông tin quan trọng trong dữ liệu.

- **Duôi dài:** Biểu đồ có một "đuôi dài" bắt đầu từ khoảng 50: 50 thành phần chính kéo dài đến các thành phần chính cuối cùng. Biểu đồ cho thấy các thành phần chính cuối cùng chỉ giải thích được một lượng rất nhỏ phương sai, nhưng vẫn đóng góp một phần nhỏ vào việc tái tạo dữ liệu gốc.

2. Khả năng giữ lại thông tin trong các trường đầu tiên Ở đây ta sẽ tiếp tục sử dụng biểu đồ thể hiện hiện tỷ lệ phương sai tích lũy của 100 thành phần đầu tiên để đánh giá kỹ hơn cho việc giảm chiều dữ liệu.



Hình 3.4: Phương sai tích lũy của 100 thành phần chính đầu tiên.

Đặc điểm nổi bật

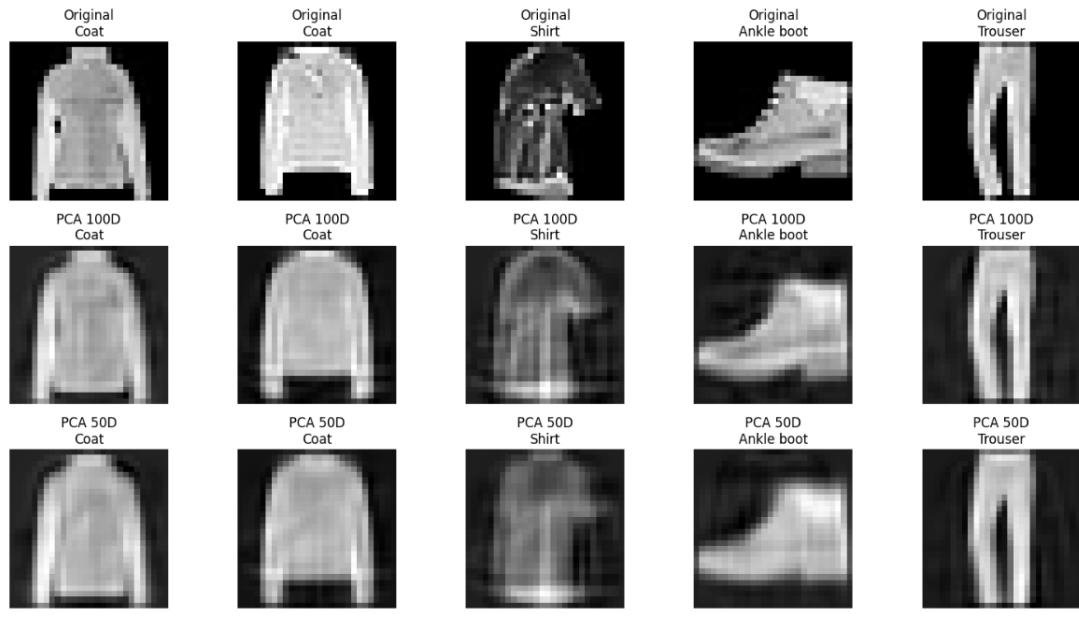
- Tại khoảng **0 - 15 thành phần chính**, tốc độ tăng phương sai tích lũy nhanh
- Tại khoảng **15 - 30 thành phần chính**, tốc độ tăng của tỷ lệ phương sai tích lũy bắt đầu giảm đáng kể, hình thành điểm khuỷu tay rõ ràng trên biểu đồ Elbow.

Tỷ lệ giữ lại thông tin tăng nhanh từ 1 đến 15 thành phần chính và từ 15 thành phần chính thì chậm dần

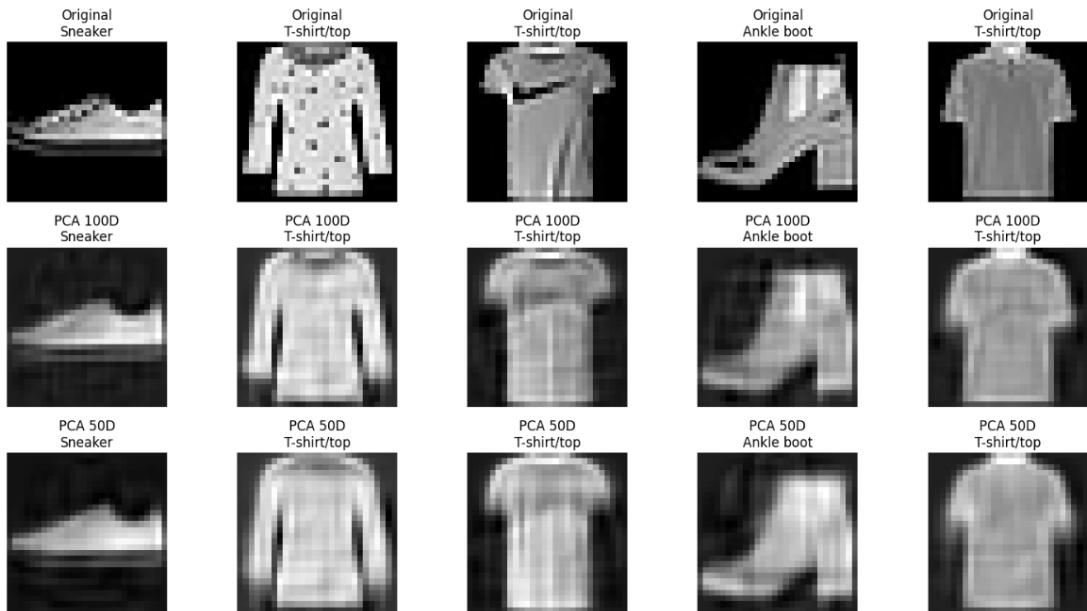
- Khi tăng lên **40 thành phần chính**, tỉ lệ phương sai tích lũy = **80%**, mang lại một cải thiện nhỏ so với 15 thành phần chính.
- Sau mốc **60 thành phần chính**, việc tăng thêm thành phần không mang lại cải thiện đáng kể về tỉ lệ phương sai tích lũy.

- Khi tăng lên khi lớn hơn 100 thành phần chính, tỉ lệ phương sai tích lũy đạt khoảng 91%.

Ảnh giảm chiều thực tế:



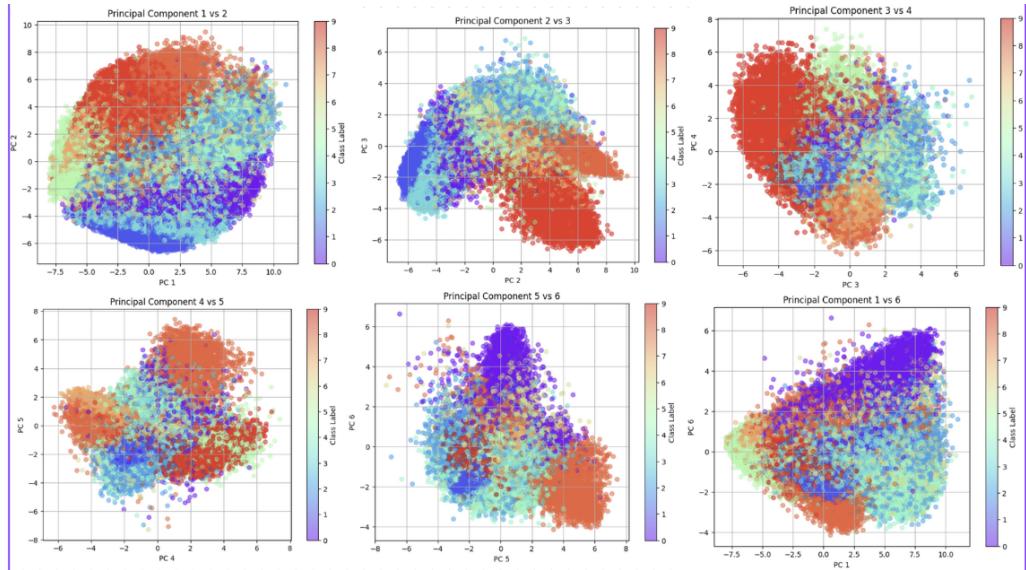
Hình 3.5: 5 ảnh ngẫu nhiên được giảm về 100 và 50 chiều.



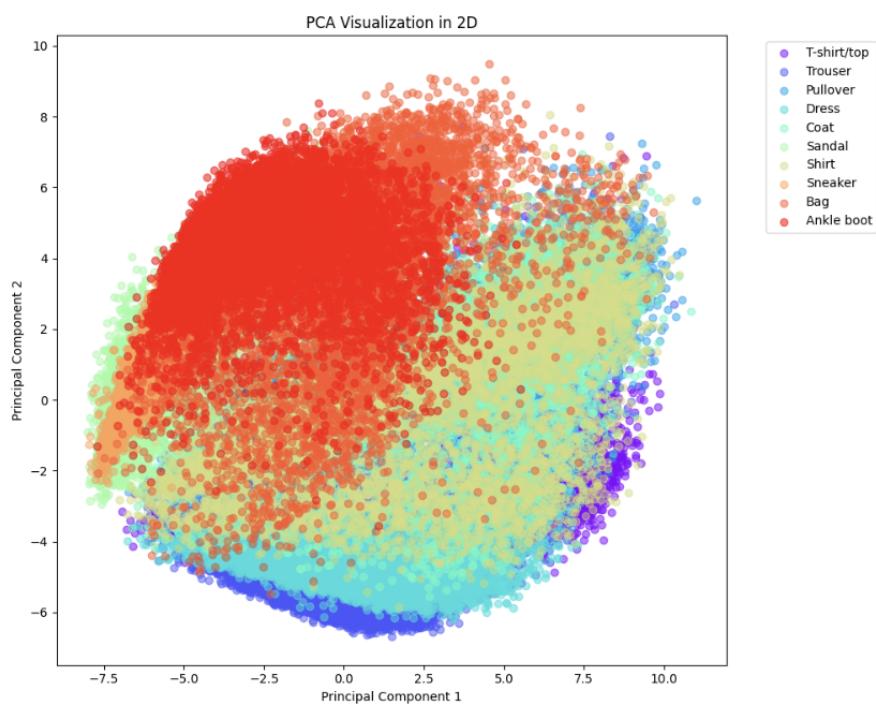
Hình 3.6: 5 ảnh ngẫu nhiên được giảm về 100 và 50 chiều.

- Dựa vào biểu đồ 3.2 và 3.4 ta thấy việc giảm chiều đến mốc 100 và 50 chiều vẫn đảm bảo độ nhận diện ảnh kể cả dùng mắt thường.
- Khi giảm về mốc 100 và 50 thành phần chính thì ảnh đã giảm bớt đi một số chi tiết cho thấy PCA cũng giảm độ nhiễu của ảnh đi khá nhiều.

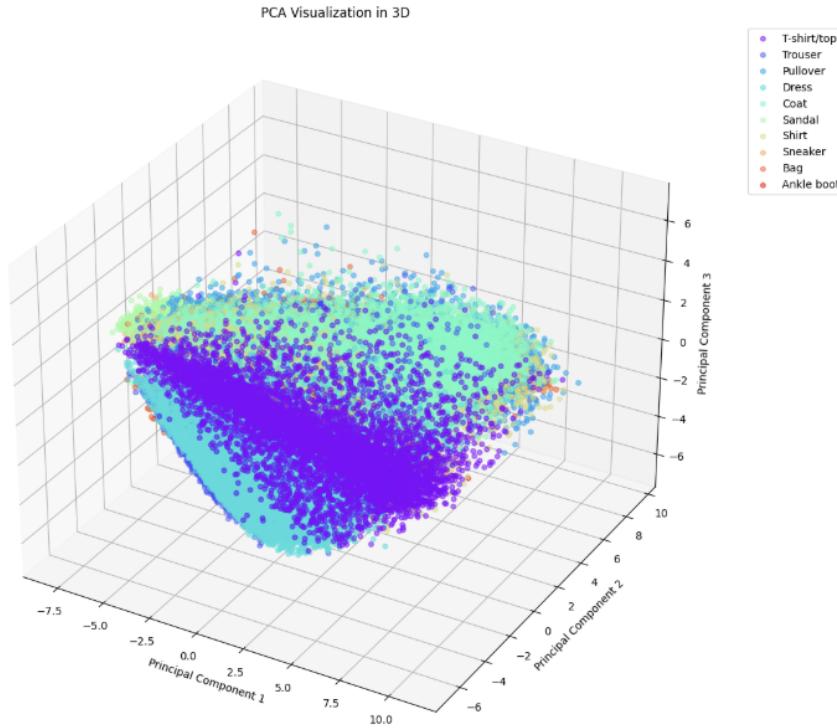
3.Khả năng trực quan hóa : Mặc dù tỷ lệ phương sai tích lũy của các thành phần đầu trong khoảng 100 thành phần đầu đạt khá cao nhưng khả năng về bảo toàn khoảng cách, cũng như lượng thông tin bị mất sau khi giảm chiều cho thấy việc dùng PCA để mà trực quan hóa dữ liệu làm khó khăn trong việc phân tích.



Hình 3.7: 6 cặp ảnh đầu tiên khi trực quan hóa.



Hình 3.8: Trục quan hóa 2D sau khi giảm chiều bằng PCA.



Hình 3.9: Trục quan hóa 3D sau khi giảm chiều bằng PCA.

Hình 3.8 và hình 3.9 cho thấy PCA khi giảm về 2 hoặc 3 để phân cụm quan hóa phân cụm dữ liệu đã không phù hợp.

Việc PCA tối ưu hóa việc giữ lại phương sai của dữ liệu làm cho không đảm bảo rằng các thành phần chính được chọn sẽ phù hợp cho nhiệm vụ cụ thể như trực quan phân cụm hay phân loại.

4. Ý nghĩa và kết luận

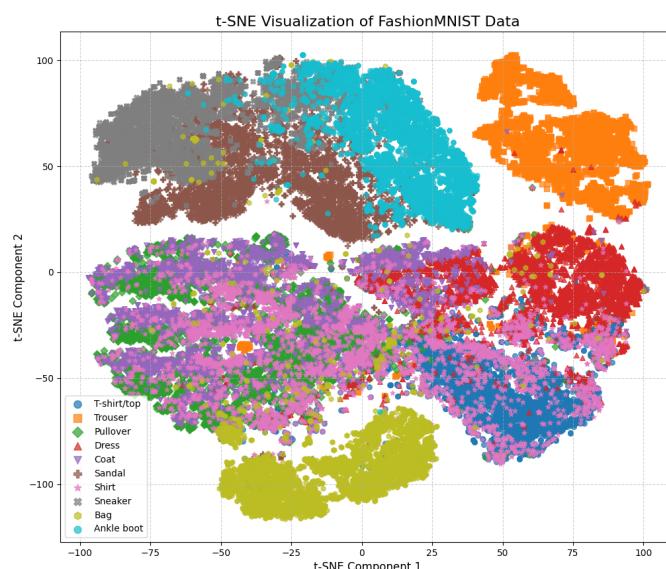
- **Phương sai tích lũy:** Sự tập trung phần lớn phương sai tích lũy ở các thành phần chính đầu tiên cho thấy rằng các đặc trưng quan trọng nhất trong dữ liệu đã được trích xuất và biểu diễn rõ ràng. Điều này phản ánh hiệu quả của PCA trong việc phát hiện các yếu tố chi phối chính của dữ liệu.
- **Phương sai giải thích:** Biểu đồ phương sai giải thích cũng đã nhấn mạnh được: PCA có thể giảm đáng kể số chiều của dữ liệu mà vẫn giữ lại được phần lớn thông tin quan trọng. Hình dạng giảm dần của biểu đồ cho thấy dữ liệu có cấu trúc rõ ràng, tức là có một số đặc trưng quan trọng chi phối phần lớn sự biến thiên của dữ liệu.
- **Hiệu quả trích xuất đặc trưng:** PCA thể hiện khả năng vượt trội trong việc tóm lược các thông tin cốt lõi từ dữ liệu. Tuy nhiên, để tái tạo hoàn toàn dữ liệu gốc mà không mất mát thông tin đáng kể, cần sử dụng một số lượng lớn thành phần chính. Điều này có thể làm tăng đáng kể chi phí tính toán.
- **Giới hạn của việc giảm chiều:** Với tỷ lệ phương sai tích lũy đạt trên **80%** khi sử dụng khoảng **20 thành phần chính**, việc tăng thêm số lượng thành phần không mang lại nhiều lợi ích đáng kể. Điều này chỉ ra rằng 20 thành phần chính đầu tiên đã nắm bắt được phần lớn thông tin quan trọng trong dữ liệu.

- **Khả năng trực quan hóa:** Kết quả trực quan hóa trên không gian 2D và 3D chỉ ra rằng PCA không phù hợp để phân biệt rõ ràng các cụm hoặc lớp trong dữ liệu. Lượng thông tin bị mất trong quá trình giảm chiều làm giảm khả năng phân cụm hiệu quả, đặc biệt khi tỷ lệ giữ lại thông tin và độ khả năng phân cụm thấp khi giảm chiều ảnh về các số chiều nhỏ như 2 hoặc là 3.
- **Số thành phần chính tối ưu:** chọn khoảng **45-50 thành phần chính** giúp tối ưu hóa cả độ chính xác của mô hình lẫn chi phí tính toán, đồng thời đạt được sự cân bằng giữa việc bảo toàn thông tin và hiệu quả của các thuật toán sử dụng.
- **Tổng kết:** PCA là một phương pháp mạnh mẽ để trích xuất và biểu diễn đặc trưng từ dữ liệu, việc áp dụng PCA trong dữ liệu FashionMNIST cũng sẽ giúp các mô hình phân loại giảm overfitting. Tuy nhiên, trong các bài toán đòi hỏi tái tạo dữ liệu chính xác hoặc phân cụm chính xác, PCA có thể không tối ưu nếu không sử dụng đủ số lượng thành phần chính. Do đó, cần cân nhắc chi phí tính toán và yêu cầu bài toán khi lựa chọn số lượng thành phần chính phù hợp.

5. Giải pháp Vì PCA gây khó khăn trong việc trực quan hóa nên để cải thiện, nên sử dụng các kỹ thuật giảm chiều nâng cao như **t-SNE** hoặc **UMAP**.

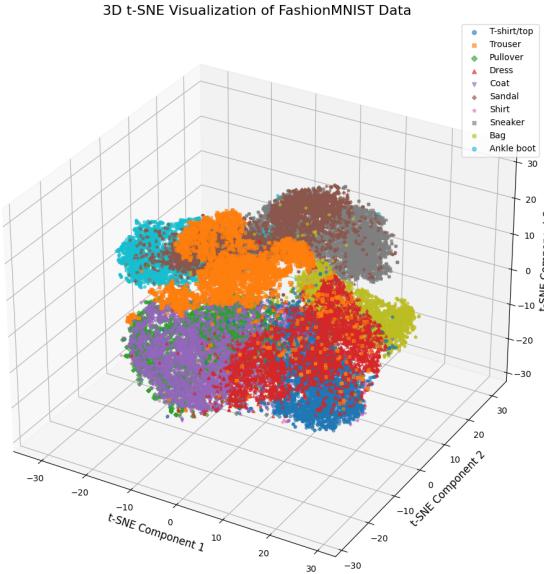
3.2.2 T-SNE

Đầu tiên, chúng tôi sẽ giảm chiều dữ liệu của tập Training(48000 mẫu) về 2 chiều và minh họa kết quả sử dụng các tham số như sau: n_components = 2(hoặc 3), learning_rate = 200, init = "random", perplexity = 30, random_state = 42, metric = "euclidean". Hình 3.10 trực quan hóa dữ liệu mà thuật toán t-SNE giảm về 2 chiều trả lại.



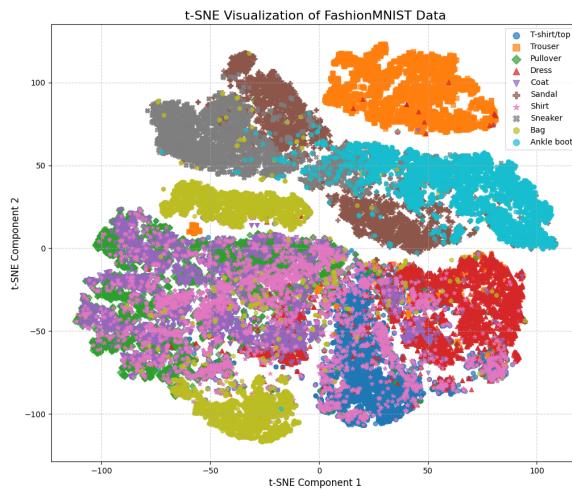
Hình 3.10: Trực quan hóa dữ liệu gốc đã được giảm xuống 2 chiều qua thuật toán t-SNE

Tiếp theo, chúng tôi cũng sẽ giảm chiều dữ liệu nhưng về 3 chiều và minh họa kết quả sử dụng cùng một bộ tham số như trên. Hình 3.11 cho thấy những thuộc tính mà thuật toán t-SNE giảm về 3 chiều trả lại.

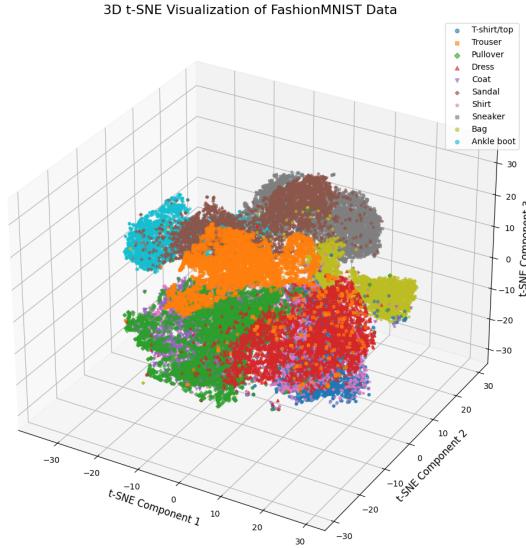


Hình 3.11: Trực quan hóa dữ liệu gốc đã được giảm xuống 3 chiều qua thuật toán t-SNE

Vì dữ liệu ban đầu có 784 cột, chúng tôi sẽ áp dụng PCA giảm về 50 chiều trước, sau đó mới áp dụng t-SNE với bộ tham số tương tự như hai ví dụ trên. Kết quả minh họa được biểu diễn với ảnh 3.12(về 2 chiều) và ảnh 3.13(về 3 chiều).



Hình 3.12: Trực quan hóa dữ liệu giảm chiều bằng PCA trước, đã được giảm xuống 2 chiều qua thuật toán t-SNE



Hình 3.13: Trực quan hóa dữ liệu giảm chiều bằng PCA trước, đã được giảm xuống 3 chiều qua thuật toán t-SNE

3.3 Sử dụng Softmax Regression

Trong báo cáo này, phương pháp Softmax Regression được sử dụng cho 2 loại dữ liệu đầu vào bao gồm: bộ dữ liệu chưa giảm chiều (là bộ dữ liệu Fashion MNIST được chuyển về dạng $N \times 784$) và bộ dữ liệu giảm chiều bằng phương pháp PCA (là bộ dữ liệu Fashion MNIST giảm còn 46 chiều, có dạng $N \times 46$), ở đây N là số ảnh. Để sử dụng phương pháp này, chúng tôi sử dụng thư viện scikit-learn trong Python. Trên cả 2 loại dữ liệu, chúng tôi triển khai model Logistic Regression trong thư viện này với các tham số chung: `multi_class = 'multinomial'`, `max_iter = 10000`.

Tiến hành thực nghiệm trên cả 2 loại dữ liệu trên với các tỷ lệ `train:validation` như sau: 4:1, 7:3, 6:4.

Các chỉ số đánh giá bao gồm: **Accuracy** (Training / Validation), **Precision** (Training / Validation), **Recall** (Training / Validation).

3.3.1 Áp dụng Softmax Regression với dữ liệu chưa giảm chiều

STT	Train/Validation	Bộ tham số	Accuracy	Precision	Recall
1	4:1	<code>solver='lbfgs'</code>	0.889 / 0.850	0.888 / 00.	0.889 / 0.850
2	4:1	<code>solver='saga'</code>	0.885 / 0.859	0.884 / 0.858	0.885 / 0.859
3	4:1	<code>solver='sag'</code>	0.887 / 0.858	0.886 / 0.857	0.887 / 0.858
4	7:3	<code>solver='lbfgs'</code>	0.892 / 0.840	0.891 / 0.838	0.892 / 0.840
5	7:3	<code>solver='saga'</code>	0.887 / 0.853	0.886 / 0.852	0.887 / 0.853
6	7:3	<code>solver='sag'</code>	0.889 / 0.851	0.888 / 0.850	0.889 / 0.851
7	6:4	<code>solver='lbfgs'</code>	0.898 / 0.836	0.897 / 0.834	0.898 / 0.836
8	6:4	<code>solver='saga'</code>	0.892 / 0.848	0.891 / 0.847	0.892 / 0.848
9	6:4	<code>solver='sag'</code>	0.893 / 0.845	0.893 / 0.844	0.893 / 0.845

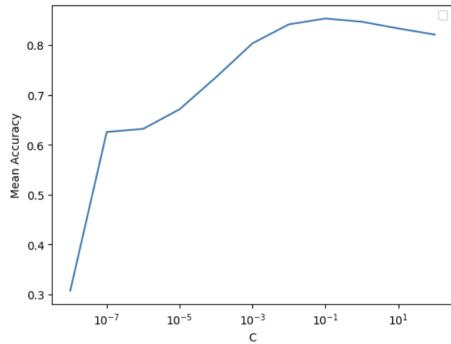
Bảng 3.4: Thực nghiệm với các bộ dữ liệu và không dùng hiệu chỉnh L2.

Từ bảng 3.4, nhận thấy: Các mô hình 4, 7, 8 và 9 có dấu hiệu bị overfit vì chỉ số trên tập huấn luyện cao hơn chỉ số trên tập đánh giá 5% - 6%. Vì vậy, chúng tôi

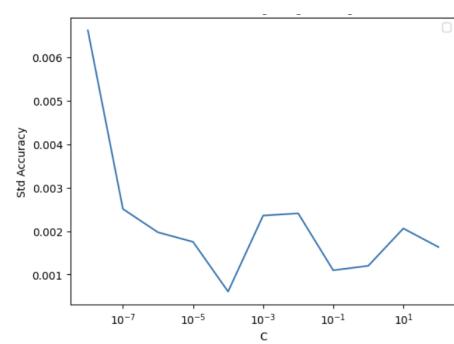
áp dụng hiệu chỉnh L2 với các mô hình này.

Xác định tham số C thông qua lớp `model_selection.GridSearchCV`: mỗi `solver` được thử nghiệm với $C \in \{10^{-8}, 10^{-7}, 10^{-6}, \dots, 100\}$; đánh giá chéo với tham số `cv = 3`; với bộ dữ liệu của mô hình 4 cho `solver = 'lbfgs'`, bộ dữ liệu của mô hình 8 cho `solver = 'saga'` và bộ dữ liệu của mô hình 9 cho `solver = 'sag'`.

(i) **Với `solver = 'lbfgs'`:**



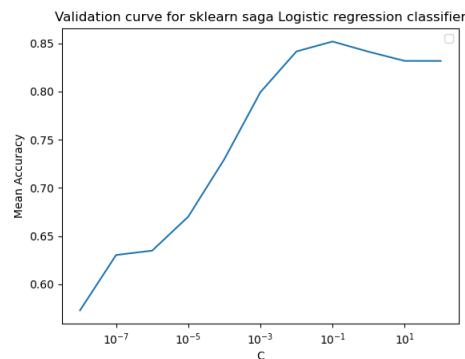
(a) Hệ số C và chỉ số mean accuracy (lbfgs).



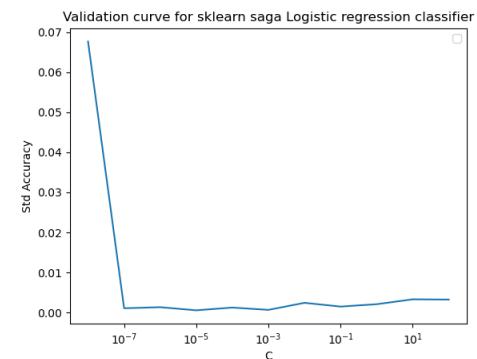
(b) Hệ số C và chỉ số std accuracy (lbfgs).

Hình 3.14: So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.

(ii) **Với `solver = 'saga'`:**



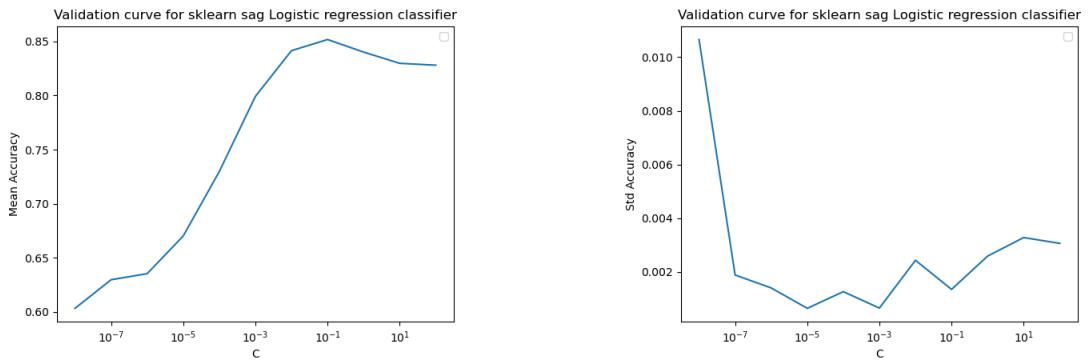
(a) Hệ số C và chỉ số mean accuracy (saga).



(b) Hệ số C và chỉ số std accuracy (saga).

Hình 3.15: So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.

(iii) **Với `solver = 'sag'`:**



(a) Hệ số C và chỉ số mean accuracy (sag).

(b) Hệ số C và chỉ số std accuracy (sag).

Hình 3.16: So sánh mối quan hệ giữa hệ số C với mean accuracy và std accuracy.

Từ các hình 3.14, 3.16, 3.15, ta thấy rằng tại $C = 0.1$ chỉ số mean accuracy lớn nhất và chỉ số std accuracy đủ nhỏ. Điều này cho thấy với $C = 0.1$ mô hình có độ chính xác tốt và ổn định. Vậy nên, chỉ số $C = 0.1$ phù hợp với các `solver` = 'lbfgs', 'saga', 'sag'. Đặc biệt, mô hình với `solver` = 'lbfgs' và $C = 0.1$ có chỉ số mean accuracy cao nhất, std accuracy nhỏ nhất.

Sau khi áp dụng hiệu chỉnh L2, ta có bảng tổng hợp chỉ số của các mô hình:

STT	train/validation	Bộ tham số	Accuracy	Precision	Recall
1	4:1	<code>solver='lbfgs'</code>	0.888 / 0.850	0.887 / 0.848	0.888 / 0.850
2	4:1	<code>solver='saga'</code>	0.885 / 0.859	0.884 / 0.858	0.885 / 0.859
3	4:1	<code>solver='sag'</code>	0.887 / 0.858	0.886 / 0.857	0.887 / 0.858
4	7:3	<code>solver='lbfgs', C=0.1</code>	0.873 / 0.861	0.872 / 0.860	0.873 / 0.861
5	7:3	<code>solver='saga'</code>	0.887 / 0.847	0.886 / 0.846	0.887 / 0.847
6	7:3	<code>solver='sag'</code>	0.889 / 0.851	0.888 / 0.850	0.889 / 0.851
7	6:4	<code>solver='lbfgs', C=0.1</code>	0.874 / 0.860	0.873 / 0.860	0.874 / 0.860
8	6:4	<code>solver='saga', C=0.1</code>	0.874 / 0.861	0.873 / 0.860	0.874 / 0.861
9	6:4	<code>solver='sag', C=0.1</code>	0.874 / 0.861	0.873 / 0.860	0.874 / 0.861

Bảng 3.5: Thực nghiệm với các bộ dữ liệu sau khi dùng hiệu chỉnh L2.

Nhận xét:

- Các mô hình được áp dụng hiệu chỉnh L2 đều có cải thiện rõ ràng với các chỉ số khá cao, không bị chênh lệch nhiều giữa tập training và validation.
- Các chỉ số Precision và Recall khá cân bằng ở mức cao chứng tỏ mô hình vừa có khả năng hạn chế dự đoán sai, vừa có khả năng hạn chế bỏ sót các giá trị quan trọng.
- Các mô hình 4, 7, 8 và 9 có chỉ số trên tập training và validation vừa cao vừa cân bằng, nghĩa là khả năng tổng quát hóa của các mô hình này tốt và việc áp dụng hiệu chỉnh L2 đã mang lại kết quả tốt.
- Mô hình 4 đảm bảo: chỉ số trên hai tập đủ cao và chênh lệch nhau ít nhất
 \Rightarrow Chọn mô hình 4 là mô hình phù hợp với dữ liệu.

3.3.2 Áp dụng Softmax Regression với dữ liệu đã giảm chiều

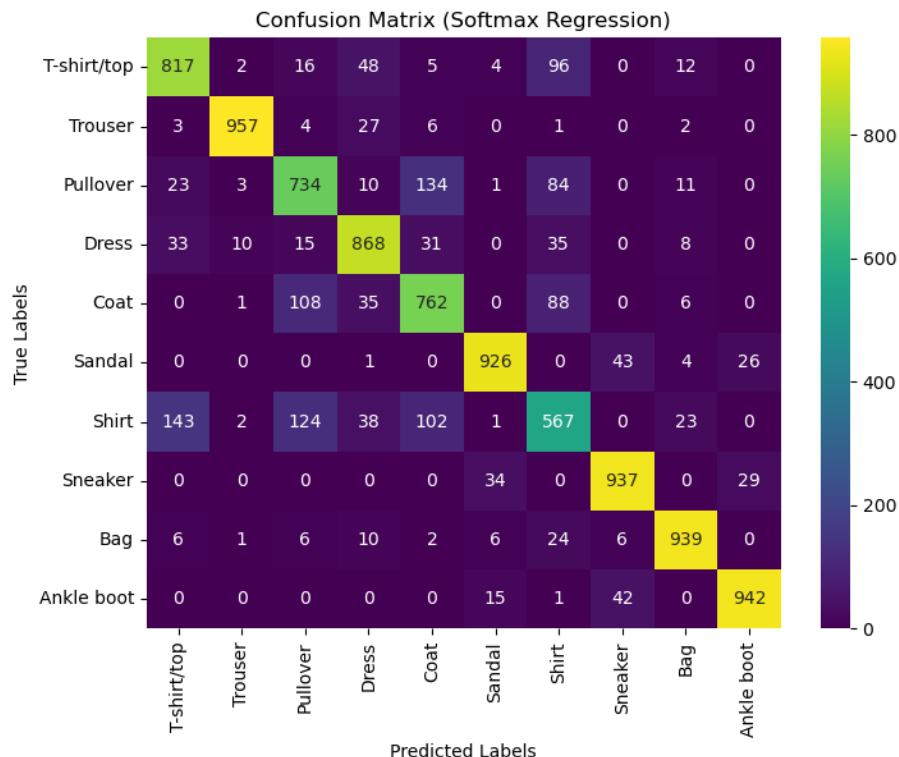
STT	train/validation	Bộ tham số	Accuracy	Precision	Recall
1	4:1	solver='lbfgs'	0.841 / 0.846	0.839 / 0.845	0.841 / 0.846
2	4:1	solver='saga'	0.840 / 0.845	0.839 / 0.844	0.840 / 0.845
3	4:1	solver='sag'	0.840 / 0.846	0.839 / 0.845	0.840 / 0.846
4	7:3	solver='lbfgs'	0.841 / 0.843	0.839 / 0.842	0.841 / 0.843
5	7:3	solver='saga'	0.840 / 0.843	0.839 / 0.842	0.840 / 0.843
6	7:3	solver='sag'	0.841 / 0.843	0.839 / 0.842	0.841 / 0.843
7	6:4	solver='lbfgs'	0.840 / 0.843	0.839 / 0.842	0.840 / 0.843
8	6:4	solver='saga'	0.840 / 0.843	0.838 / 0.842	0.840 / 0.843
9	6:4	solver='sag'	0.840 / 0.843	0.893 / 0.842	0.840 / 0.843

Bảng 3.6: Thực nghiệm với các bộ dữ liệu và không dùng hiệu chỉnh L2.

Từ bảng 3.6 có thể thấy rằng các mô hình đều không bị overfit nhưng độ chính xác thấp hơn từ 3% đến 5% so với khi chưa giảm chiều, nên việc giảm chiều dữ liệu đã làm mất một số thông tin cần thiết. Tuy các chỉ số không quá cao nhưng sau khi giảm chiều, mô hình rất ổn định với các solver khác nhau.

3.3.3 Lựa chọn mô hình phù hợp và đánh giá

Sau quá trình thử nghiệm, chúng tôi chọn bộ dữ liệu có tỷ lệ train:validation = 7:3 và các tham số phù hợp (solver = 'lbfgs', penalty='l2', C = 0.1), chúng tôi tiến hành huấn luyện trên tập train và đánh giá mô hình trên tập test. Kết quả thu được như sau: **Accuracy score:** 0.845, **Precision score:** 0.844, **Recall score:** 0.845.



Hình 3.17: Ma trận nhầm lẫn của phương pháp Softmax Regression.

Một số đánh giá về phương pháp được Softmax Regression được chọn:

1. Tổng quan về hiệu suất: Phương pháp này hoạt động tốt trên một số lớp như Trouser, Sandal, Sneaker, Bag và Ankle boot với lượng dự đoán chính xác trên 95% (các ô chéo màu vàng sáng). Các lớp khác như Shirt, Pullover và Coat có sự nhầm lẫn đáng kể. Với các chỉ số đánh giá như trên, nhìn chung mô hình đạt hiệu quả ở mức chưa quá tốt.
2. Các lớp có kết quả tốt nhất:
 - **Trouser:** 957 mẫu đúng trên tổng số 1000 mẫu thực tế, đạt Recall = 95.7%. Mô hình dự đoán 976 mẫu thuộc lớp Trouser với 957 dự đoán đúng, đạt Precision = 98%.
 - **Sandal:** 926 mẫu đúng trên tổng số 1000 mẫu thực tế, đạt Recall = 92.6%. Mô hình dự đoán 987 mẫu thuộc lớp Sandal với 926 dự đoán đúng, đạt Precision = 94%.
 - **Sneaker:** 937 mẫu đúng trên tổng số 1000 mẫu thực tế, đạt Recall = 93.7%. Mô hình dự đoán 1028 mẫu thuộc lớp Sneaker với 937 dự đoán đúng, đạt Precision = 91.1%.
 - **Bag:** 939 mẫu đúng trên tổng số 1000 mẫu thực tế, đạt Recall = 93.9%. Mô hình dự đoán 1005 mẫu thuộc lớp Bag với 939 dự đoán đúng, đạt Precision = 93.4%.
 - **Ankle boot:** 942 mẫu đúng trên tổng số 1000 mẫu thực tế, đạt Recall = 94.2%. Mô hình dự đoán 997 mẫu thuộc lớp Ankle boot với 942 dự đoán đúng, đạt Precision = 94.5%.
3. Các lớp có nhiều nhầm lẫn:
 - **Shirt:** Đây là lớp có nhiều nhầm lẫn nhất, chỉ 567 mẫu được dự đoán đúng, nhiều mẫu của lớp này bị nhầm lẫn với T-shirt/top (143 mẫu), Pullover (124 mẫu), Coat (102 mẫu),... Đạt Recall = 57%, Precision = 63%.
 - **Pullover:** Các mẫu của lớp này bị nhầm lẫn nhiều nhất với Coat (134 mẫu) và Shirt (84 mẫu), đạt Recall = 73%. Mô hình dự đoán 1007 mẫu thuộc lớp Pullover nhưng chỉ có 734 dự đoán đúng, đạt Precision = 73%.
 - **Coat:** Các mẫu của lớp này bị nhầm lẫn nhiều nhất với Pullover (108 mẫu), Shirt (88 mẫu), đạt Recall = 76%. Mô hình dự đoán 1042 mẫu thuộc lớp Coat nhưng chỉ có 762 dự đoán đúng, đạt Precision = 73%.
4. Nguyên nhân nhầm lẫn:
 - Các nhãn như T-shirt/top, Shirt, Pullover, và Coat có sự giống nhau về mặt hình dạng và cấu trúc, khiến mô hình đơn giản như Softmax Regression khó phân biệt được.
 - Các lớp có đặc trưng cụ thể hơn như Trouser và Sneaker ít bị nhầm lẫn vì đặc tính của chúng dễ nhận diện hơn.
5. Phương án cải thiện: Tăng cường dữ liệu hoặc sử dụng các mô hình phức tạp hơn như: SVM, CNN...

3.4 Sử dụng Convolutional Neural Network

Chúng tôi sẽ xây dựng mô hình CNN với cấu trúc như sau:

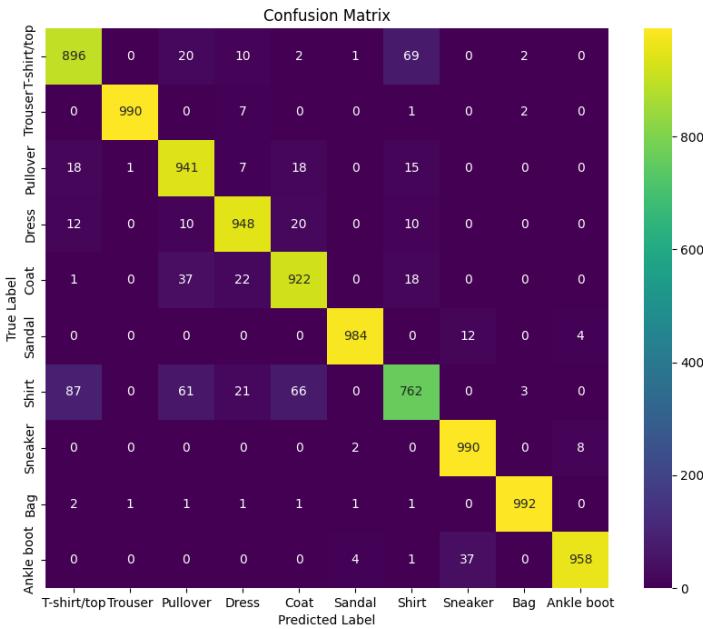
- Trước hết, tất cả các bộ lọc của tầng Convolution sẽ có kích thước 3×3 . Ảnh đầu vào của các tầng Convolution sẽ bổ sung các pixel ở rìa để giữ nguyên kích thước ảnh ở đầu ra với tham số padding = "same".
- Hai phần đầu tiên có tầng Convolution với 32 bộ lọc. Tiếp theo sau là tầng Batch Normalization và tầng cho hàm kích hoạt ReLU cho đầu ra của tầng BatchNormalization. Tầng Dropout = 0.25 được sử dụng để 25% số node sẽ bị giảm đi.
- Phần tiếp theo có tầng Convolution tuy nhiên với 64 bộ lọc. Tầng Batch Normalization và hàm kích hoạt ReLU vẫn được sử dụng. Tuy nhiên trước khi đi qua tầng Dropout với tham số 0.25, các ảnh đầu vào sẽ phải qua tầng MaxPooling với kích thước bộ lọc chạy qua ảnh là 2×2 .
- Phần tiếp theo cũng sẽ là cấu trúc: Convolution - Batch Normalization - ReLU - Dropout(0.25). Tuy nhiên, số bộ lọc của Convolution được tăng lên 128.
- Kết quả của 4 phần trên sẽ được làm phẳng để làm đầu vào của Phần MLP. MLP được sử dụng có 2 tầng Hidden với số unit là: 512 - 128. Xen giữa các tầng Hidden có các tầng Batch Normalization và Dropout(0.5). Kết quả cuối cùng được đưa ra bởi tầng kích hoạt với hàm kích hoạt Softmax(cho bài toán phân loại nhiều lớp).

Các tầng Batch Normalization và Dropout được sử dụng để tránh tình trạng Overfit. Đối với phần MLP, tham số Dropout được khuyến cáo là 0.5[2] còn ở các tầng Convolution thì tham số sẽ được chọn dựa trên quá trình thực nghiệm.

Chúng tôi sẽ huấn luyện mô hình dựa trên tập Training đã được xáo trộn, Tập Validation sẽ được sử dụng để đánh giá quá trình huấn luyện và tập Test sẽ được sử dụng để đánh giá mô hình dựa trên các độ đo khác nhau.

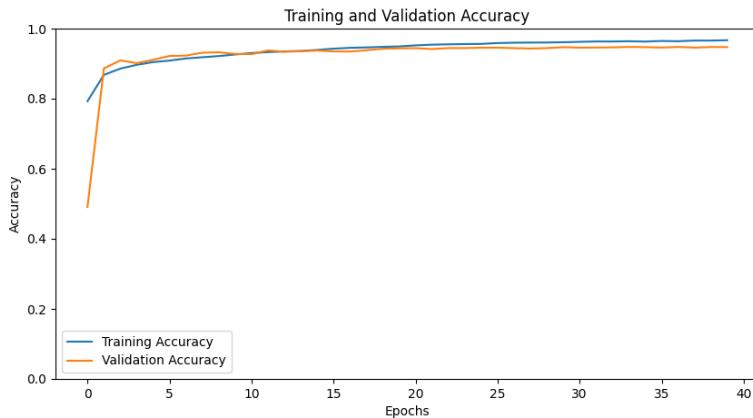
Mô hình được xây dựng sử dụng Tensorflow với hàm mất mát là SparseCategoricalEntropy(cho bài toán phân loại nhiều lớp) với hàm tối ưu là Adam để tối ưu mô hình với learning rate = 0.001. Chúng tôi áp dụng việc giảm learning rate theo hàm mũ với công thức $10^{-3} \times 0.9^{\text{learning_rate}}$. Cuối cùng, chúng tôi huấn luyện với 40 epochs và batch size là 128.

Sau khi kết thúc quá trình huấn luyện, chúng tôi đánh giá mô hình trên tập test và đạt được các kết quả theo độ đo như sau: **Accuracy** = 93.83%, **Precision** = 93.83%, **Recall** = 93.83%. Dưới đây là hình ảnh của **ma trận Confusion**



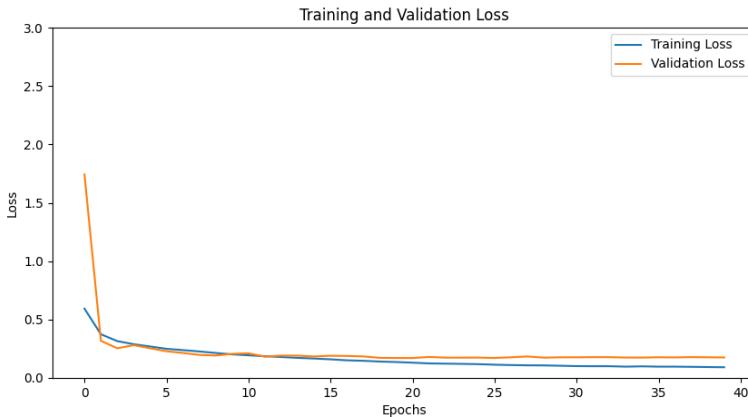
Hình 3.18: Ma trận Confusion của mô hình CNN

Dưới đây là hai hình ảnh mô tả **quá trình Training và Validation**.



Hình 3.19: Hình ảnh mô tả sự thay đổi của Accuracy khi huấn luyện mô hình

Từ hình ảnh quá trình Training và Validation, ta có thể thấy mô hình khởi đầu với Accuracy trung bình(xấp xỉ 70%) và Loss thấp(Xấp xỉ 1.0). Chỉ cần sau 5 epoch đầu, mô hình đã đạt được Accuracy 90% và Loss dưới 0.5. Kể từ sau epoch thứ 5, Accuracy và Loss cho việc Training và Validation tăng dần dần - điều này ứng với việc learning rate được cài đặt để giảm dần để tránh tình trạng đi khỏi điểm hội tụ. Điều này cho thấy quyết định chọn số epoch = 40 là hợp lý, đủ để mô hình có khả năng phân loại chính xác nhất. Đồng thời, với 3 độ đo chính xác đạt kết quả



Hình 3.20: Hình ảnh mô tả sự thay đổi của Loss khi huấn luyện mô hình

như trên, CNN là mô hình tốt hơn so với Softmax Regression để phân loại ảnh thời trang.

Phân tích chi tiết kết quả của CNN

Các nhãn được dự đoán gần hoàn hảo:

- Trouser.
- Sandal.
- Sneaker.
- Bag.

Các nhãn có nhiều sai sót trong dự đoán:

- T-shirt/Top.
- Pullover
- Dress.
- Coat.
- Shirt.

Nhãn "Shirt" bị nhầm lẫn nhiều nhất với các nhãn:

- T-shirt/Top.
- Pullover.
- Coat.

Chương 4

Kết luận

Về 2 mô hình giảm chiều PCA và t-SNE:

- Thực nghiệm PCA luôn hoàn thành với tốc độ nhanh hơn thực nghiệm t-SNE(do t-SNE chỉ xét từng cặp dữ liệu một)
- Đối với kết quả trực quan hóa dữ liệu của t-SNE(Hình 3.10 và hình 3.11 đều thể hiện các phân cụm rõ ràng hơn so với kết quả trực quan hóa dữ liệu của PCA(Hình 3.8 và hình 3.9).
- Điều này chứng tỏ khả năng hình thành những cấu trúc hình học ở không gian đã giảm chiều của thuật toán t-SNE tốt hơn so với thuật toán PCA

Về 2 mô hình phân loại Softmax Regression và CNN:

- Nhờ có sự hỗ trợ của GPU, CNN có thể hoàn thiện quá trình huấn luyện nhanh hơn so với Softmax Regression sử dụng CPU mặc dù CNN sử dụng nhiều dữ liệu hơn.
- Các độ đo kết quả của CNN đạt gần 94%, trong khi kết quả tốt nhất của Softmax Regression(solver = 'lbfgs', penalty = 'l2', C = 0.1) chỉ đạt 84.5% trên các độ đo kết quả.
- Các lớp mà mô hình Softmax Regression có hiệu quả chưa cao thì mô hình CNN vẫn đảm bảo hiệu quả.
- Cả 2 mô hình phân loại đều có sự nhầm lẫn trong việc dự đoán ảnh với nhãn "Shirt"
- Softmax dự đoán tốt nhất với 5 nhãn: "Trouser", "Sandal", "Sneaker", "Bag", "Ankle Boot".
- CNN dự đoán gần hoàn hảo với 4 nhãn "Trouser", "Sandal", "Bag", "Sneaker".

Tài liệu tham khảo

- [1] T. Vu, “Machine learning cơ bản - bài 20: Principal component analysis,” *Machine Learning cơ bản*, 2017.
- [2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.
- [3] K. D. N., “Understanding batch normalization,” *Medium*, 2018. Accessed: 2024-12-14.
- [4] L. van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.
- [5] R. A. Jacobs, “Increased rates of convergence through learning rate adaptation,” *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.
- [6] L. van der Maaten, “Accelerating t-sne using tree-based algorithms,” *Journal of Machine Learning Research*, vol. 15, no. 93, pp. 3221–3245, 2014.
- [7] Dive into Deep Learning, “Why convolutional neural networks?,” 2021. Accessed: December 7, 2024.