TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP.HỒ CHÍ MINH

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO


# BÀI BÁO CÁO

# THỰC HÀNH

**Môn : Máy học sâu**

**Giảng viên HD : Nguyễn Thị Khánh Tiên**

**Sinh viên thực hiện: Huỳnh Thế Hy**

**Mã lớp : DL2301CLCA**

*TP . Hồ Chí Minh, Ngày 26 tháng 11 năm 2025.*

# MỤC LỤC

# TÓM TẮT

Báo cáo này tổng hợp kết quả của chuỗi các bài thực hành về Deep Learning, tập trung vào các kỹ năng nền tảng và nâng cao. Nội dung báo cáo bao gồm: (1) Xây dựng mạng nơ-ron cơ bản để phân loại ảnh trên tập dữ liệu FashionMNIST; (2) Ứng dụng kỹ thuật tinh chỉnh (fine-tuning) trên các kiến trúc pre-trained (ResNet-18, VGG-16) cho bài toán phân loại ảnh CIFAR-10; (3) Khai thác sức mạnh của thư viện Hugging Face cho các tác vụ Xử lý Ngôn ngữ Tự nhiên, từ việc sử dụng pipeline có sẵn đến tinh chỉnh mô hình cho bài toán phân tích cảm xúc; và (4) Tìm hiểu các kỹ thuật biểu diễn dữ liệu chuỗi cho mạng nơ-ron hồi quy. Báo cáo trình bày ngắn gọn mục tiêu, phương pháp, các đoạn mã quan trọng và kết quả đạt được trong mỗi bài thực hành.

# CHƯƠNG 1: PHÂN LOẠI ẢNH THỜI TRANG (FASHIONMNIST) VỚI MẠNG NƠ-RON CƠ BẢN

## 1.1. Giới thiệu và chuẩn bị Dữ liệu

Bài thực hành này tập trung vào việc xây dựng một mạng nơ-ron nhân tạo từ đầu sử dụng PyTorch để giải quyết bài toán phân loại ảnh trên tập dữ liệu FashionMNIST. Dữ liệu được tải về và xử lý thông qua DataLoader, đồng thời áp dụng các kỹ thuật tăng cường dữ liệu như lật ảnh và xoay nhẹ để tăng tính đa dạng cho tập huấn luyện.

```
# Định nghĩa transform cho tập Train (có tăng cường dữ liệu)
train_transform = Compose([
    RandomHorizontalFlip(p=0.5),   # Lật ngang ảnh ngẫu nhiên với xác suất 50%
    RandomRotation(degrees=10),    # Xoay nhẹ ảnh tối đa 10 độ
    ToTensor(),
    Normalize((0.2860,), (0.3530,))
])
# Tạo DataLoaders
train_dataloader     =     DataLoader(training_data,     batch_size=BATCH_SIZE,
shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=BATCH_SIZE, shuffle=False)
```

## 1.2. Xây dựng và Huấn luyện Mô hình

Mô hình được xây dựng gồm các lớp Linear và hàm kích hoạt ReLU. Kỹ thuật Dropout cũng được thêm vào để giảm thiểu hiện tượng overfitting. Quá trình huấn luyện sử dụng hàm mất mát CrossEntropyLoss và trình tối ưu hóa Adam. Đặc biệt, kỹ thuật **Early Stopping** được áp dụng để tự động dừng quá trình huấn luyện khi hiệu suất trên tập validation không còn cải thiện, giúp tiết kiệm thời gian và chọn ra mô hình tốt nhất.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
```

```
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Dropout(p=0.3),  # Tắt ngẫu nhiên 30% nơ-ron ở lớp này
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),  # Tắt tiếp 30% ở lớp này để tránh học vẹt
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

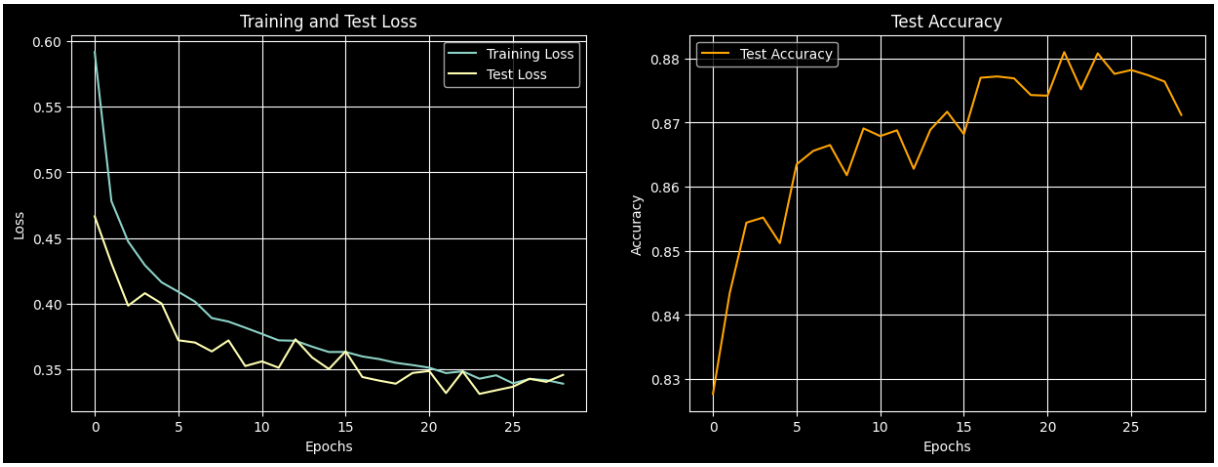
### 1.3. Kết quả và Đánh giá

Sau 29 epochs, quá trình huấn luyện đã dừng lại do Early Stopping. Mô hình tốt nhất đạt được **độ chính xác 88.1%** trên tập kiểm thử.

**Kết quả cuối cùng:**

Test Error:
Accuracy: 88.1%, Avg loss: 0.331350

Biểu đồ dưới đây trực quan hóa quá trình huấn luyện, cho thấy loss giảm dần và accuracy tăng dần, chứng tỏ mô hình đã học hiệu quả.




Trực quan hóa một vài dự đoán trên tập test cho thấy khả năng phân loại tốt của mô hình:

# CHƯƠNG 2: TINH CHỈNH MẠNG NƠ-RON TÍCH CHẬP PRE-TRAINED CHO PHÂN LOẠI ẢNH

## 2.1. Giới thiệu và Chuẩn bị Dữ liệu

Chương này khám phá kỹ thuật tinh chỉnh (fine-tuning) trên các kiến trúc mạng nơ-ron tích chập (CNN) đã được huấn luyện trước trên tập dữ liệu ImageNet. Mục tiêu là so sánh hiệu suất của **ResNet-18** và **VGG-16** khi áp dụng cho bài toán phân loại trên tập **CIFAR-10**. Dữ liệu được tăng cường mạnh mẽ hơn với ColorJitter để mô hình tổng quát hóa tốt hơn.

## 2.2. Thí nghiệm Tinh chỉnh: ResNet-18 và VGG-16

Cả hai mô hình đều được tải về với trọng số đã huấn luyện trước. Lớp phân loại cuối cùng được thay thế để phù hợp với 10 lớp của CIFAR-10.

- **Với ResNet-18:** Do kiến trúc gốc được thiết kế cho ảnh 224x224, lớp tích chập đầu tiên đã được điều chỉnh để phù hợp hơn với ảnh 32x32 của CIFAR-10, bằng cách thay kernel 7x7 (stride 2) bằng kernel 3x3 (stride 1) và loại bỏ lớp MaxPool đầu tiên.

```
model_resnet.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
model_resnet.maxpool = nn.Identity() # Loại bỏ lớp maxpool đầu tiên
```

- **Với VGG-16:** Kiến trúc này phù hợp hơn với ảnh nhỏ, do đó chỉ cần thay đổi lớp classifier cuối cùng.

Quá trình huấn luyện sử dụng trình tối ưu hóa SGD với momentum, CosineAnnealingLR để điều chỉnh learning rate, và hàm mất mátCrossEntropyLoss với label_smoothing để chống overfitting.

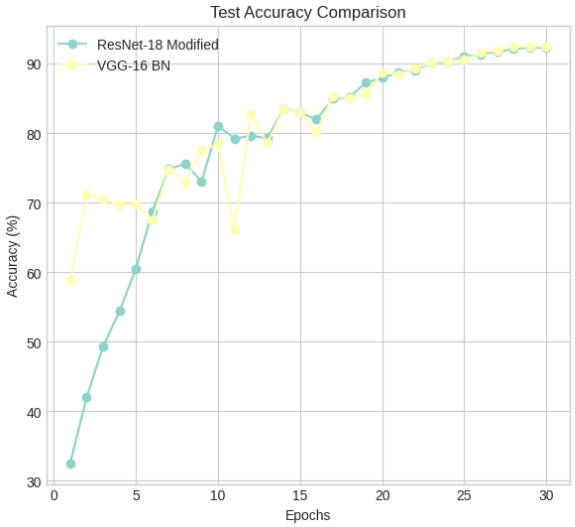
## 2.3. Kết quả và So sánh

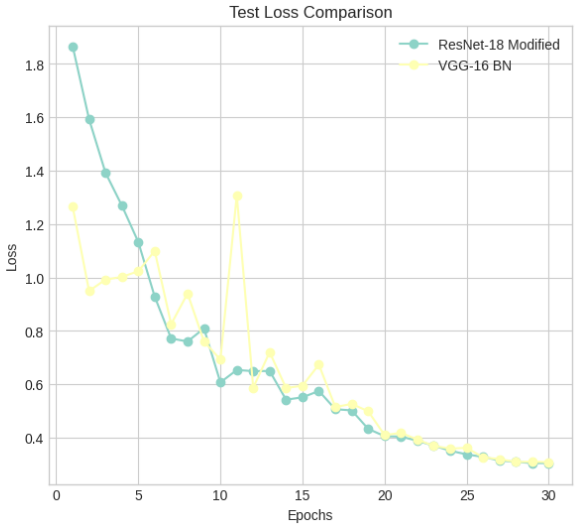
Sau 30 epochs, cả hai mô hình đều đạt hiệu suất rất cao:

- **Best ResNet-18 Accuracy:** 92.35%
- **Best VGG-16 Accuracy:** 92.52%

**Phân tích:**

- **Hiệu suất:** VGG-16 (BN) cho độ chính xác cuối cùng nhỉnh hơn một chút so với ResNet-18.
- **Hiệu quả:** Dù có độ chính xác thấp hơn một chút, ResNet-18 là một kiến trúc nhẹ và huấn luyện nhanh hơn. Việc đạt được kết quả gần tương đương chứng tỏ hiệu quả của kiến trúc residual.
- **Tầm quan trọng của việc tùy chỉnh:** Thí nghiệm với ResNet-18 cho thấy việc điều chỉnh kiến trúc để phù hợp với đặc điểm dữ liệu là cực kỳ quan trọng.



# CHƯƠNG 3: ỨNG DỤNG XỬ LÝ NGÔN NGỮ TỰ NHIÊN VỚI HUGGING FACE

## 3.1. Giới thiệu

Chương này trình bày cách sử dụng thư viện transformers của Hugging Face cho hai tác vụ chính trong NLP: sử dụng pipeline để phân tích cảm xúc một cách nhanh chóng và thực hiện quy trình fine-tuning hoàn chỉnh một mô hình ngôn ngữ cho bài toán phân loại văn bản.

## 3.2. Phân tích Cảm xúc với Pipeline

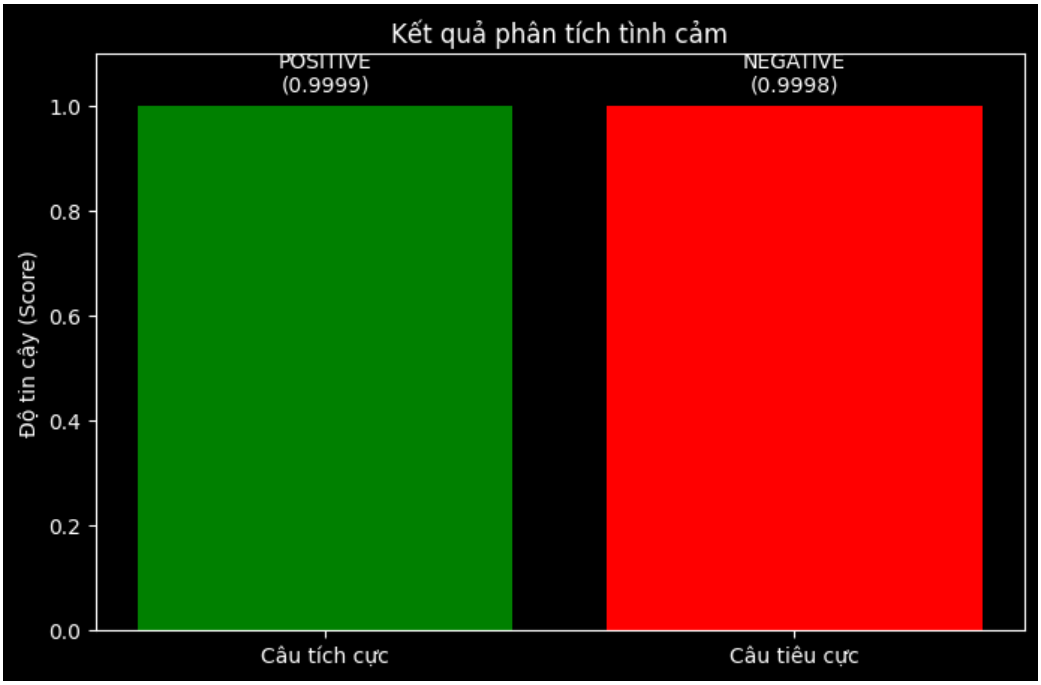
Chức năng pipeline của Hugging Face cho phép thực hiện các tác vụ NLP phức tạp chỉ với vài dòng mã. Bằng cách sử dụng mô hình distilbert-base-uncased-finetuned-sst-2-english, chúng ta có thể dễ dàng phân loại cảm xúc của câu văn.

**Kết quả:**

Câu: 'I absolutely love this course, it is so insightful and well-structured!'
-> Nhãn dự đoán: POSITIVE, với độ tin cậy: 0.9999

Câu: 'The plot of the movie was predictable and the acting was rather disappointing.'
-> Nhãn dự đoán: NEGATIVE, với độ tin cậy: 0.9998

Kết quả cho thấy mô hình hoạt động rất hiệu quả và chắc chắn trong các dự đoán của mình.




### 3.3. Tinh chỉnh mô hình cho phân loại văn bản

Bài thực hành này tiến hành fine-tuning mô hình distilbert-base-uncased trên bộ dữ liệu đánh giá phim IMDB cho bài toán phân loại cảm xúc nhị phân (tích cực/tiêu cực). Quy trình bao gồm các bước: tải và tiền xử lý (tokenize) dữ liệu, định nghĩa tham số huấn luyện, và sử dụng lớp Trainer để tự động hóa quá trình.

### 3.4. Kết quả tinh chỉnh

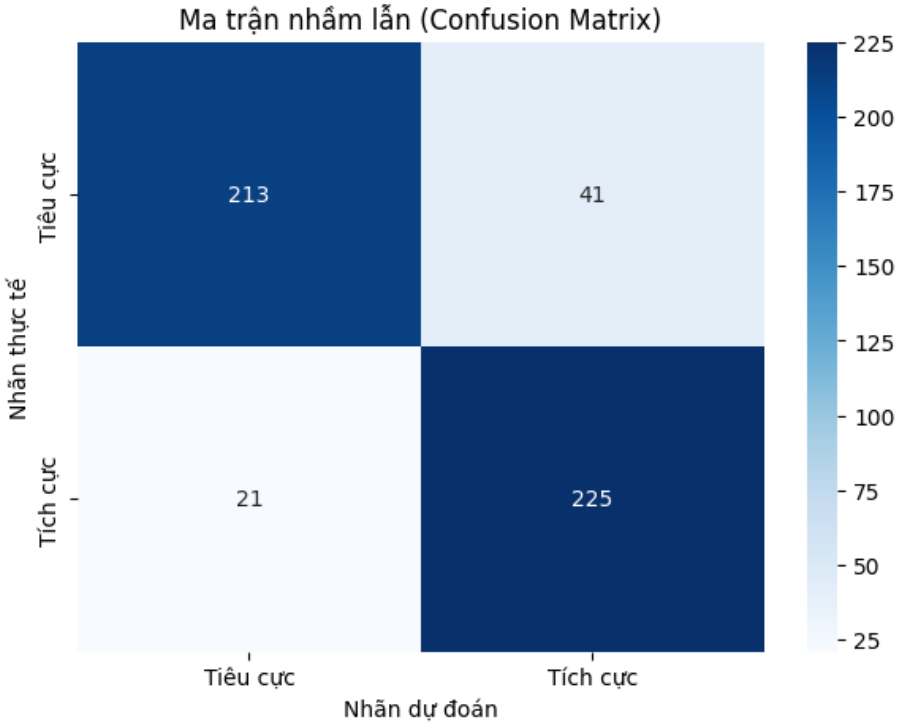
Sau 3 epochs, mô hình đạt được **độ chính xác 87.6%** trên tập kiểm thử con.

**Kết quả đánh giá:**

eval_loss: 0.3079
eval_accuracy: 0.8760

Ma trận nhầm lẫn dưới đây trực quan hóa hiệu suất của mô hình, cho thấy số lượng các dự đoán đúng (đường chéo chính) và sai.

Mô hình phân loại đúng phần lớn các mẫu, chứng tỏ quá trình fine-tuning đã thành công trong việc chuyên biệt hóa mô hình cho tác vụ cụ thể.



# CHƯƠNG 4: CÁC KỸ THUẬT BIỂU DIỄN CHUỖI CHO MẠNG NƠ-RON HỒI QUY

## 4.1. Giới thiệu

Chương cuối cùng mang tính lý thuyết và thực hành cơ bản, trình bày các kỹ thuật thiết yếu để chuẩn bị dữ liệu dạng chuỗi trước khi đưa vào các mô hình như RNN.

## 4.2. Các kỹ thuật mã hóa

1. **Mã hóa số nguyên (Integer Encoding):** Gán một số nguyên duy nhất cho mỗi từ trong từ điển.

```
--- Original Sentence ---
deep learning is an exciting field
--- Câu đã được mã hóa số nguyên ---
```

2. **Mã hóa One-Hot (One-Hot Encoding):** Biểu diễn mỗi từ bằng một vector nhị phân thưa thớt. Kỹ thuật này rõ ràng nhưng không hiệu quả về mặt không gian.

```
# Vector One-Hot cho từ "deep" (ID=3) [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

3. **Nhúng từ (Word Embeddings):** Biểu diễn mỗi từ bằng một vector dày đặc, có số chiều thấp. Phương pháp này không chỉ tiết kiệm không gian mà còn nắm bắt được mối quan hệ ngữ nghĩa giữa các từ. Lớp torch.nn.Embedding được sử dụng để thực hiện việc này.

--- Kích thước đầu ra (chuỗi vector embedding): torch.Size() ---(Mỗi từ giờ được biểu diễn bằng một vector 10 chiều)

### 4.3. Xử lý Chuỗi có Độ dài Thay đổi

1. **Đệm (Padding):** Thêm các giá trị 0 vào các chuỗi ngắn hơn để tất cả các chuỗi trong một batch có cùng độ dài.

```
--- Các chuỗi sau khi padding (đến độ dài 6) ---
[2, 9, 10, 12, 8, 0]
```

2. **Cắt bớt (Truncation):** Cắt bỏ phần cuối của các chuỗi dài hơn một ngưỡng tối đa cho trước.

Việc kết hợp cả Padding và Truncation đảm bảo tất cả các chuỗi đầu vào đều có một kích thước cố định, sẵn sàng cho việc xử lý bởi mô hình.

# TỔNG KẾT

Chuỗi bài thực hành đã cung cấp một cái nhìn tổng quan và thực tiễn về các khía cạnh quan trọng của Deep Learning. Từ việc xây dựng mạng nơ-ron cơ bản, báo cáo đã tiến tới các kỹ thuật nâng cao hơn như tinh chỉnh mô hình pre-trained cho cả bài toán thị giác máy tính và xử lý ngôn ngữ tự nhiên. Kết quả đạt được trên các tập dữ liệu CIFAR-10 và IMDB chứng tỏ sức mạnh của phương pháp transfer learning. Cuối cùng, các kỹ thuật nền tảng về biểu diễn dữ liệu chuỗi đã củng cố kiến thức cần thiết để làm việc với các mô hình xử lý văn bản.