ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

-----***-----

BÁO CÁO FINAL PROJECT DEEP LEARNING

ĐỀ TÀI

Phân loại văn bản nhiều nhãn với các bài đăng facebook

Học phần: Học Sâu (AIT3001_37)

Giảng Viên Hướng Dẫn: TS. Triệu Hải Long

Ngành: Trí tuệ nhân tạo - QH-2022-I/CQ-A-AI

Nhóm: 26

Tên thành viên	Mã sinh viên
Phạm Đăng Phong	22022614
Chu Thân Nhất	22022578
Bàn Hoàng Sơn	22022651
Nguyễn Bảo Sơn	22022613
Cao Đặng Quốc Vương	22022601
Lê Nguyên Vũ	22022544

Mục Lục

Phân chia công việc

Họ và tên	MSSV	Công Việc
Phạm Đăng Phong (Nhóm trưởng)	**22022614**	**Tìm hiểu model BERT, cài đặt model Xử lí, trực quan hóa dữ liệu. Viết báo cáo**
Chu Thân Nhất	**22022578**	**Tìm hiểu model BERT, cài đặt model. Viết báo cáo**
Bàn Hoàng Sơn	**22022651**	**Tìm hiểu, cài đặt và thử nghiệm mô hình BiLSTM+CNN. Viết báo cáo**
Nguyễn Bảo Sơn	**22022613**	**Tìm hiểu, cài đặt và thử nghiệm mô hình BiLSTM+CNN. Viết báo cáo**
Cao Đặng Quốc Vương	**22022601**	**Tìm hiểu, cài đặt và huấn luyện mô hình LSTM. Viết báo cáo**
Lê Nguyên Vũ	**22022544**	**Tìm hiểu, cài đặt và huấn luyện mô hình LSTM. Viết báo cáo**

Link code: Github

I. Tổng quan về bài toán

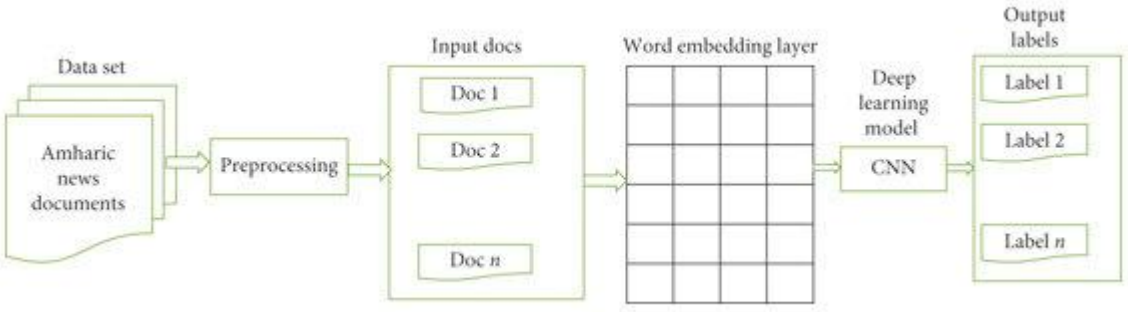
1. Mục đích và ý nghĩa của bài toán

Bài toán phân loại văn bản nhằm tự động gán nhãn các tài liệu vào các danh mục cụ thể dựa trên nội dung của chúng. Điều này giúp tiết kiệm thời gian và công sức, nâng cao hiệu quả xử lý thông tin. Ứng dụng của bài toán này rất rộng, từ lọc email, phân tích cảm xúc đến đề xuất nội dung.

2. Phạm vi và ứng dụng thực tiễn

Báo cáo tập trung vào phân loại các văn bản tiếng Việt, áp dụng các phương pháp truyền thống (LSTM) và hiện đại (BERT) để đạt kết quả tốt nhất. Ứng dụng thực tiễn bao gồm quản lý thông tin, tối ưu hóa quy trình kinh doanh, hỗ trợ giáo dục và y tế.

3. Mô tả bài toán



Input: Đoạn văn bản với chủ đề có sẵn

Output: Chủ đề văn bản sau khi phân loại

II. Dữ liệu

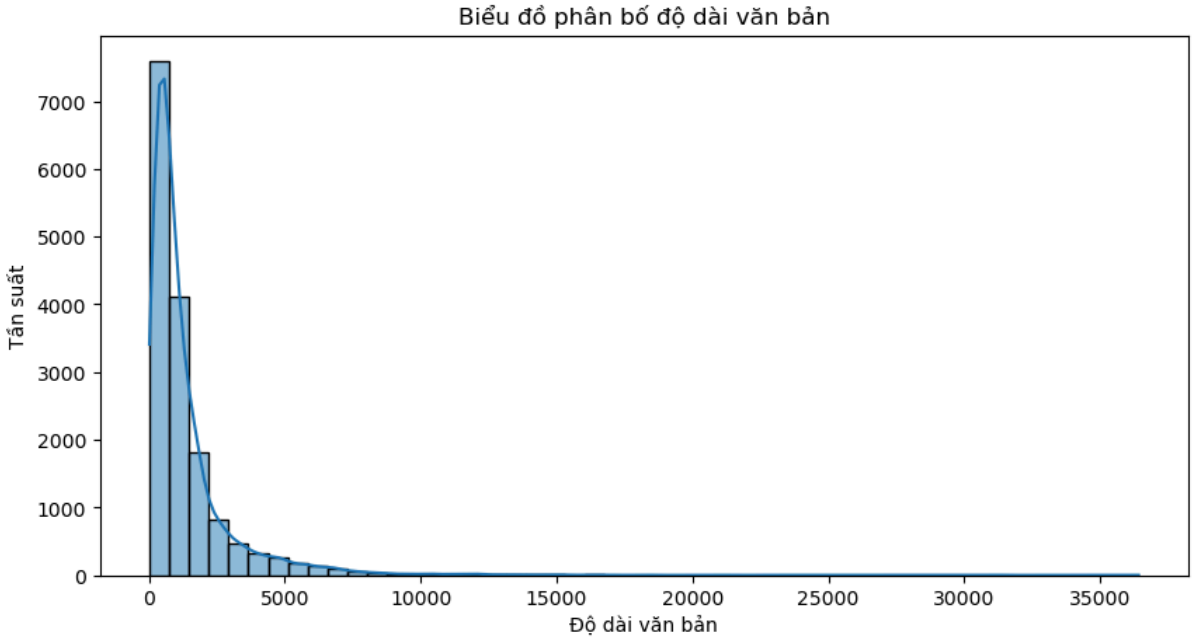
1. Thông tin về bộ dữ liệu

- Bộ dữ liệu cho bài toán được thu thập từ mạng xã hội Facebook, nó bao gồm các đoạn văn bản thuộc các hội nhóm khác nhau về chủ đề, nội dung bài viết
- Bộ dữ liệu huấn luyện bao gồm 16000 bài viết đã được gán 23 loại nhãn khác nhau đã được thu thập và được lưu dưới dạng file .txt. Tương tự, bộ dữ liệu kiểm thử có 23 loại nhãn khác nhau với 10017 bài viết

Du_lich	Chinh_tri	Sach	May_tinh_va_thiet_bi_dien_tu
Nha_dat	Giao_duc	Do_an_va_do_uong	Mang_internet_va_vien_thong
Mua_sam	Giao_thong	Lam_dep_va_the_hinh	Suc_khoe_va_benh_tat
Tai_chinh	Phap_luat	Con_nguoi_va_xa_hoi	Thoi_quen_va_so_thich
Khoa_hoc	Giai_tri	Nghe_thuat	Kinh_doanh_va_Cong_nghiep
Nha_va_vuon	The_thao	Cong_nghe_moi	

Dưới đây là một số thông tin chi tiết về bộ dữ liệu:

- **Số lượng nhãn**: 23
- **Độ dài văn bản**: Đa dạng, thường từ 21 - 1200 từ và ký tự
- **Số lượng văn bản**: 16000 bài viết trên tập huấn luyện và 10017 bài viết trên tập kiểm thử



2. Xử lý dữ liệu

- Các bài viết trong bộ dữ liệu được thu thập từ Facebook nên tồn tại rất nhiều loại ký tự, văn bản nhiễu không thích hợp cho bài toán phân loại chủ đề ví dụ như số điện thoại, đường link, icon, hashtag… nên các ký tự này sẽ được làm sạch. Hơn nữa để đồng nhất dữ liệu thì việc lowercase cũng rất quan trọng.
- Khi mô tả dữ liệu, nhóm nhận thấy dữ liệu có một số hàng bị trùng nên cũng cần được tiến hành loại bỏ và áp dụng tiền xử lý

	text	label
count	16000	16000
unique	15223	23

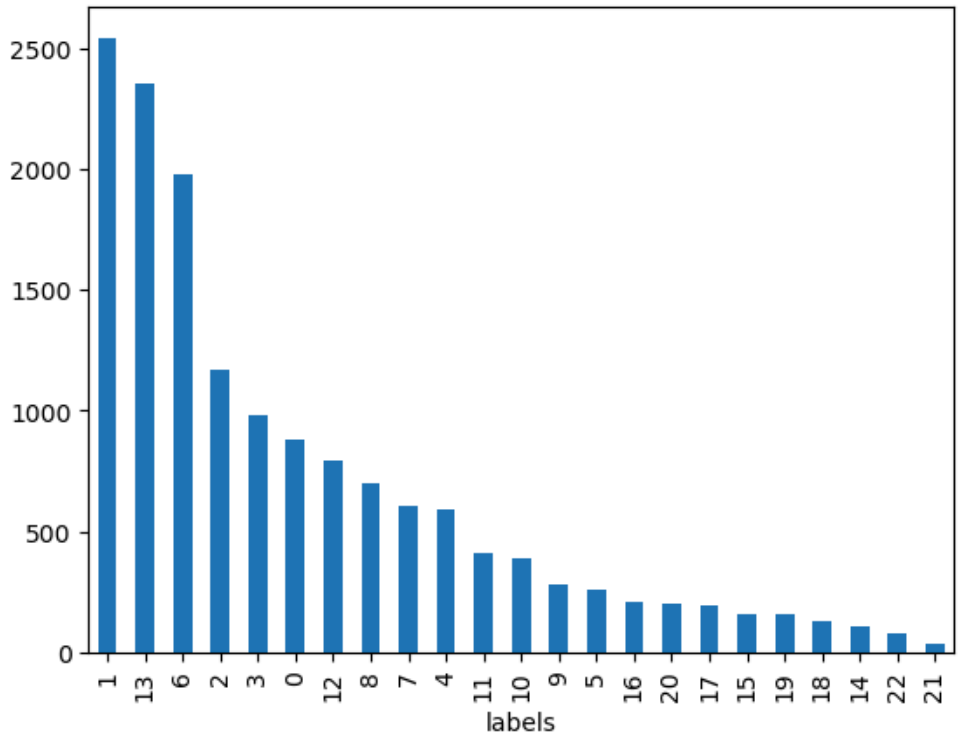
- Sau khi hoàn thành các bước làm sạch dữ liệu, đây là kết quả thu được:

🌈 CHÀO NOEL ĐÓN MƯA QUÀ TẶNG . 😍 Nhân dịp Noel, Viettel triển khai chương trình khuyến mại với nhiều ưu đãi hấp dẫn hơn dành cho Khách hàng đăng ký lắp đặt dịch vụ Internet từ ngày 23/12 đến hết ngày 31/12 . 👉 Đăng ký ngay internet tại Hà Nội và HCM: shop.viettel.vn/landing/ftth1 . 👉 Đăng ký ngay tại 61 tỉnh thành: shop.viettel.vn/landing/ftth2 . ☑ Đóng trước 3 tháng - tặng ngay 1 tháng . ☑ Đóng trước 6 tháng - tặng ngay 2 tháng . ☑ Đóng trước 12 tháng - tặng ngay 4 tháng . ☑Đóng trước 18 tháng - tặng ngay 6 tháng . 📲 Truy cập ngay https://shop.viettel.vn/fp và ứng dụng My Viettel: https://viettel.vn/app15 để xem và tham khảo các gói cước . ☎ Liên hệ: 18008168 để được tư vấn và hỗ trợ . 🎯 Chi tiết chương trình xem tại link: https://goo.gl/12emJd

=> chào noel đón mưa quà tặng nhân dịp noel viettel triển khai chương trình khuyến mại với nhiều ưu đãi hấp dẫn hơn dành cho khách hàng đăng ký lắp đặt dịch vụ internet từ ngày đến hết ngày đăng ký ngay internet tại hà nội và hcm shopviettelvnlandingftth đăng ký ngay tại tỉnh thành shopviettelvnlandingftth đóng trước tháng tặng ngay tháng đóng trước tháng tặng ngay tháng đóng trước tháng tặng ngay tháng đóng trước tháng tặng ngay tháng truy cập ngay và ứng dụng my viettel để xem và tham khảo các gói cước liên hệ để được tư vấn và hỗ trợ chi tiết chương trình xem tại link

3. Thống kê dữ liệu

- Tiếp theo để xem phân bố của dữ liệu, chúng ta sẽ thống kê lại dữ liệu trên từng nhãn để kiểm tra độ chênh lệch về dữ liệu:

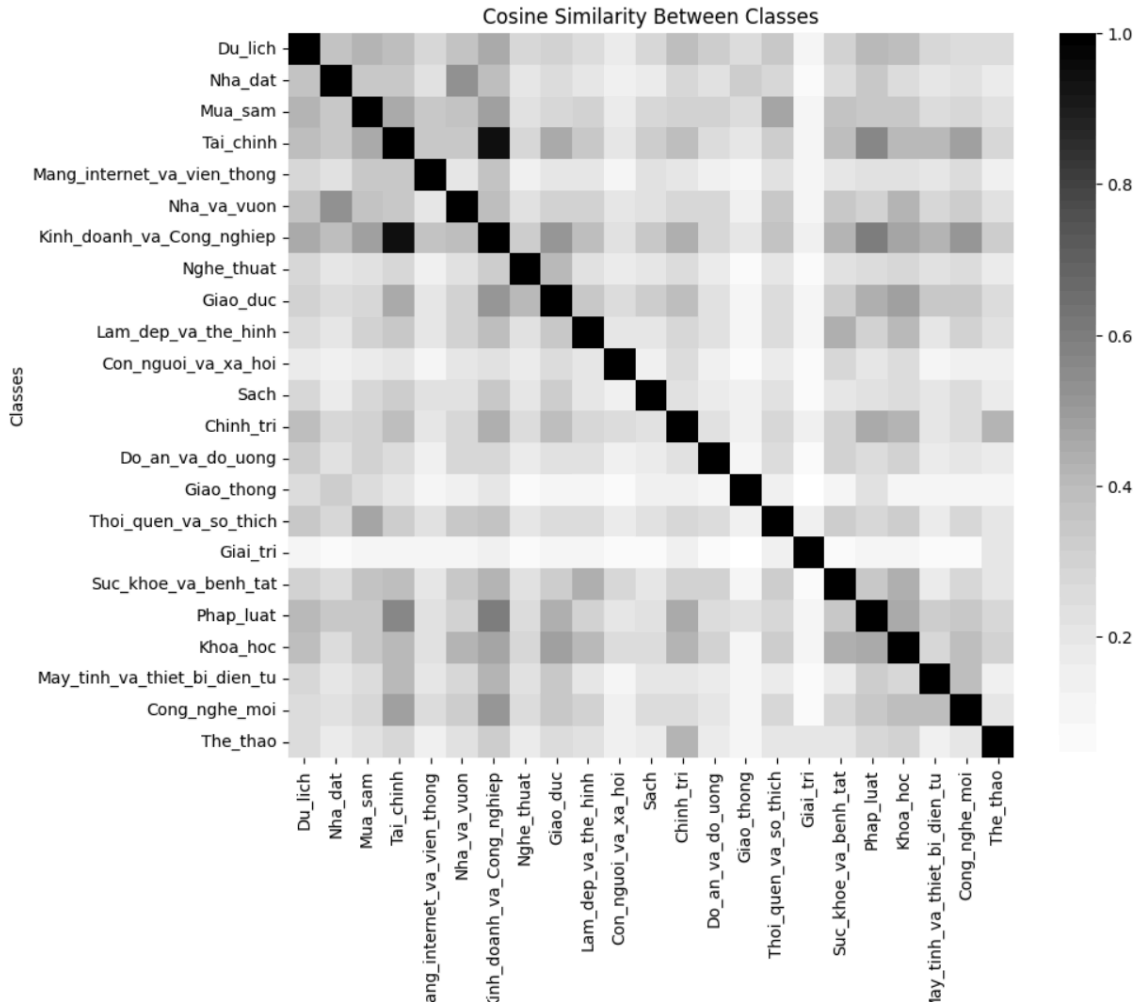


Biểu đồ cho thấy rằng số lượng nhãn chênh lệch là rất lớn, điều này sẽ làm phát sinh một số vấn đề:

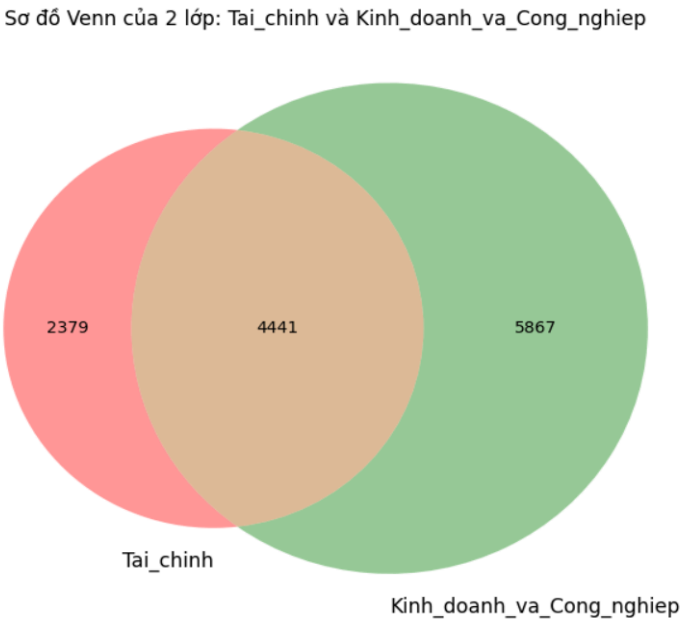
- **Hiệu suất không đồng đều:** Mô hình sẽ có xu hướng dự đoán nhãn có nhiều dữ liệu hơn. Điều này làm cho mô hình kém hiệu quả trong việc dự đoán nhãn có ít dữ liệu.
- **Độ chính xác bị lệch:** Các chỉ số đánh giá như độ chính xác (accuracy) sẽ bị lệch nếu không có sự cân bằng giữa các nhãn. Mô hình có thể đạt độ chính xác cao đơn giản vì nó đoán đúng nhiều mẫu từ nhãn có nhiều dữ liệu.

4. Phân tích dữ liệu

- Trước khi đưa vào mô hình, ta cần phải hiểu rõ về cấu trúc và đặc điểm của dữ liệu, từ đó xác định được các vấn đề tiềm ẩn như sự trùng lặp của các lớp, bước chuyển đổi dữ liệu sai sót,…
- Ở đây ta sẽ sử dụng kỹ thuật như cosine similarity để kiểm tra độ giống nhau giữa các lớp và trực quan hóa sử dụng heatmap



- Ở 2 lớp Tai_chinh và Kinh_doanh_va_cong_nghiep của biểu đồ, ta thấy được sự tương đồng cao với ô vuông màu rất đậm. Và để hiểu rõ nguyên nhân, nhóm sẽ tiếp tục tìm hiểu vấn đề thông qua biểu đồ venn:



- Tại sơ đồ ta thấy được có tới 4441 từ ngữ bị trùng lặp giữa 2 nhãn, chiếm tới hơn một nửa dữ liệu của lớp Tai_chinh, để giải quyết vấn đề này, chúng tôi sẽ thử nghiệm các phương pháp tối ưu tham số đối với các mô hình được sử dụng.

5. Xử lí dữ liệu mất cân bằng

Để xử lý vấn đề mất cân bằng dữ liệu, chúng ta có thể sử dụng các phương pháp:

- Re-sampling: Kết hợp 2 phương pháp undersample và oversample, nhóm lựa chọn số lượng dữ liệu n_samples = median số lượng mẫu trong các label giúp phân bổ dữ liệu được đồng đều.

```python
from sklearn.utils import resample

def oversample(df, label, n_samples):
    label_df = df[df['label'] == label]
    return resample(label_df,
                    replace=True,
                    n_samples=n_samples,
                    random_state=42)

def undersample(df, label, n_samples):
    label_df = df[df['label'] == label]
    return resample(label_df,
                    replace=False,
                    n_samples=n_samples,
                    random_state=42)

label_counts = train_df['label'].value_counts() # label count distribution

n_samples = int(label_counts.median()) # number of samples for each label

resampled_dfs = []

for i in range(23):
    label = id2label[i]
    label_df = train_df[train_df['label'] == label]

    if len(label_df) < n_samples:
        # Oversample minority class
        resampled_df = oversample(train_df, label, n_samples)
    elif len(label_df) > n_samples:
        # Undersample majority class
        resampled_df = undersample(train_df, label, n_samples)
    else:
        # Keep the same if it's already at the desired number of samples
        resampled_df = label_df

    resampled_dfs.append(resampled_df)

balanced_df = pd.concat(resampled_dfs)
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

- Sinh dữ liệu: Với bài toán phân loại văn bản này, ta có thể lựa chọn sinh dữ liệu sử dụng các mô hình ngôn ngữ lớn như các nghiên cứu gần đây. Trong phạm vi của thí nghiệm này, nhóm quyết định lựa chọn một giải pháp tiết kiệm thời gian hơn là sử dụng Markov chain để sinh dữ liệu.

```python
import random
from collections import defaultdict

def build_markov_chain(corpus, order):
    chain = defaultdict(list)
    for text in corpus:
        words = text.split()
        for i in range(len(words) - order):
            key = tuple(words[i:i+order])
            chain[key].append(words[i+order])
    return chain

def generate_text(chain, order, length=100):
    text = ""
    if not chain:
        return text

    current = random.choice(list(chain.keys()))
    result = list(current)
    for _ in range(length - order):
        if current not in chain or not chain[current]:
            break
        next_word = random.choice(chain[current])
        result.append(next_word)
        current = tuple(result[-order:])
    return ' '.join(result)

label_to_gen = ['Cong_nghe_moi', 'Kinh_doanh_va_Cong_nghiep', 'Phap_luat', 'Suc_khoe_va_benh_tat', 'Tai_chinh']

for label in label_to_gen:
    order = 3
    corpus = train[train['label'] == label]['cleaned_text'].tolist()
    chain = build_markov_chain(corpus, order)
    generated_text = generate_text(chain, order)
    print(f"Generated text for label '{label}': {generated_text}")

"""
Generated text for label 'Cong_nghe_moi': nhân tạo big data hot
Generated text for label 'Phap_luat': thứ trưởng bắt giữ cáo buộc hối lộ reuters cựu tổng thống peru bắn cảnh sát bắ
Generated text for label 'Tai_chinh': dẫn viên nội mà còn ảnh hưởng hình ảnh du lịch đe dọa điện thoại chức năng chụ
"""
```

1 số kết quả sau khi sinh dữ liệu:

Label	Dữ liệu ban đầu	Dữ liệu được sinh ra
Sach	Lịch đăng sách hôm nay 184 nếu bạn mới mua lần đầu vui lòng xem qua cách thức mua sách thanh toán giao hàng	Tôi yêu sách vì chúng là cửa sổ mở ra thế giới cho tâm hồn tôi. Sách là nguồn cảm hứng vô tận, nơi tôi tìm thấy sự an ủi và tri thức
Cong_nghe_moi	4 điểm chính trong xu hướng công nghệ 2017 là 1 Ứng dụng Machine Learning AI sâu rộng vào nhiều lĩnh vực 2 Ứng dụng app thông minh hơn cho nhu cầu thông tin của con người 3 Digital mọi thứ mọi dịch vụ liên quan đến đời sống con người	Sử dụng công nghệ mới giúp chúng ta nâng cao hiệu suất làm việc và sáng tạo trong công việc hàng ngày.
The_thao	Trong bóng đá mọi cầu thủ đều không thể tránh khỏi những áp lực Bản thân chúng em cũng đã phải trải qua rất nhiều áp lực	Bóng đá, cầu lông, bóng chuyền và các môn thể thao khác là những hoạt động thú vị và lành mạnh. Thể thao giúp tăng cường sức khỏe, rèn luyện tinh thần và tạo ra cơ hội giao lưu xã hội. Điều này rất quan trọng đối với mọi người, đặc biệt là những người trẻ tuổi.

III. Các phương pháp tiếp cận bài toán

1. LSTM (baseline):

1.1 Lý thuyết chung về mô hình

LSTM (Long Short-Term Memory) là một loại mạng truyền hồi (Recurrent Neural Network – RNN) được thiết kế để xử lý các dữ liệu tuần tự, như chuỗi văn bản hoặc tín hiệu thời gian. Khác với RNN truyền thống, LSTM giải quyết được vấn đề "vanishing gradient" nhờ các cấu trúc gồm cửa và trạng thái nhớ.

Các đặc điểm chính:

- **Cửa nhớ (Forget Gate):** Xác định những thông tin nào nên lưu giữ hay loại bỏ.
- **Cửa nhập (Input Gate):** Quyết định những thông tin nào được cập nhật vào trạng thái nhớ.
- **Cửa xuất (Output Gate):** Xác định phần thông tin được sử dụng cho dự đoán cuối.

Nhờ vào các cửa này, LSTM có khả năng ghi nhớ các mối quan hệ xa trong chuỗi dài hơn RNN truyền thống.

1.2 Kiến trúc mô hình sử dụng

Kiến trúc mô hình trong bài toán bao gồm các thành phần sau:

```
# Số lớp nhãn
num_classes = len(label_encoder.classes_)

# Xây dựng mô hình
model = Sequential([
    Embedding(input_dim=max_vocab_size, output_dim=128, input_length=max_sequence_length),
    LSTM(128, return_sequences=False),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

1. **Embedding Layer:**

- Biểu diễn từ vựng trong dạng vector dẫn dài, chuyển từ thành số để dòng máy có thể hiểu.
- **input_dim:** 10,000 (kích thước từ vựng).
- **output_dim:** 128 (kích thước vector embedding).
- **input_length:** 100 (độ dài tối đa của chuỗi).

2. **LSTM Layer:**
   - Số neuron: 128.
   - Kết xuất duy nhất tại cuối chuỗi (return_sequences=False).
3. **Dropout Layer:**
   - Giảm overfitting bằng cách bỏ bớt một số neuron trong khi huấn luyện.
   - Tỷ lệ dropout: 20%.
4. **Dense Layer:**
   - Lớp 1: 64 neuron, hàm kích hoạt ReLU.
   - Lớp cuối: Số neuron bằng số lớp nhãn (num_classes), hàm kích hoạt Softmax (để sinh xác suất cho từng lớp).

1.3 Cách triển khai mô hình

```
# Tokenizer
max_vocab_size = 10000
max_sequence_length = 100  # Độ dài tối đa cho mỗi câu
tokenizer = Tokenizer(num_words=max_vocab_size, oov_token='<UNK>')
tokenizer.fit_on_texts(train_texts)  # Chỉ fit trên tập train

# Chuyển văn bản thành chuỗi số
train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)

# Padding
train_padded = pad_sequences(train_sequences, maxlen=max_sequence_length, padding='post', truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=max_sequence_length, padding='post', truncating='post')
```

- Văn bản được mã hóa thành chuỗi số dựa trên từ điển từ vựng.

```
train_sequences
```

```
226,
88,
117,
55,
5,
133,
391,
374,
75,
55,
250,
346,
65,
217,
284,
66,
219,
1,
3823,
1696,
9,
419,
```

- Padding để đảm bảo tất cả các chuỗi có độ dài đồng nhất (100 từ).

```
train_padded

array([[  45,   94,  105, ...,  321,   18,  105],
       [  46,  108,  322, ...,    0,    0,    0],
       [   7,  271,   24, ...,    0,    0,    0],
       ...,
       [ 305,  105,   20, ...,   98,  103,  143],
       [ 420,  260,   66, ...,  340,  420,  260],
       [ 749, 1481,   47, ...,  137,  963,  173]], dtype=int32)
```

- Nhãn được mã hóa thành số nguyên bằng LabelEncoder.

```
[9]  train_labels_encoded

array([ 4, 15, 15, ...,  9, 16,  9])
```

Xây dựng và biên dịch mô hình:

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Hiện kết quả đã ẩn

[]  model.summary()

Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(32, 100, 128)	1,280,000
lstm (LSTM)	(32, 128)	131,584
dropout (Dropout)	(32, 128)	0
dense (Dense)	(32, 64)	8,256
dropout_1 (Dropout)	(32, 64)	0
dense_1 (Dense)	(32, 23)	1,495

```
Total params: 4,264,007 (16.27 MB)
Trainable params: 1,421,335 (5.42 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,842,672 (10.84 MB)
```

- **Optimizer:** Adam (tối ưu hóa hiệu quả và nhanh chóng).
- **Loss:** Sparse Categorical Crossentropy (phù hợp với nhãn được mã hóa dưới dạng số nguyên).
- **Metric:** Accuracy.

Chia dữ liệu:

Dữ liệu liệu huấn luyện được chia với tỉ lệ các tập train, validation lần lượt là: 80% và 20%

```
from sklearn.model_selection import train_test_split

# Chia dữ liệu
X_train, X_val, y_train, y_val = train_test_split(train_padded, train_labels_encoded, test_size=0.2, random_state=42)

# In số lượng mẫu
print("Số mẫu trong tập huấn luyện:", len(X_train))
print("Số mẫu trong tập validation:", len(X_val))
```

```
Số mẫu trong tập huấn luyện: 12800
Số mẫu trong tập validation: 3200
```

1.4 Huấn luyện

Tập huấn luyện:

```
# Huấn luyện mô hình
history = model.fit(
    train_padded, train_labels_encoded,
    validation_split=0.2,
    epochs=200,
    batch_size=32
)
```

- **Validation split:** 20% dữ liệu huấn luyện được sử dụng để đánh giá trong quá trình huấn luyện.
- **Epochs:** 200 (số lần lặp qua toàn bộ dữ liệu).
- **Batch size:** 32 (số mẫu được xử lý trong mỗi lần cập nhật trọng số).

Đánh giá trên tập test:

```
[ ]  # Đánh giá trên tập test
     loss, accuracy = model.evaluate(test_padded, test_labels_encoded)
     print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

- Kết quả đánh giá được đo bằng loss và accuracy trên tập test.

Báo cáo chi tiết:

```
# Tạo Classification Report
from sklearn.metrics import classification_report

test_predictions = model.predict(test_padded)
test_predicted_labels = test_predictions.argmax(axis=1)

report = classification_report(test_labels_encoded, test_predicted_labels, target_names=label_encoder.classes_)
print(report)
```

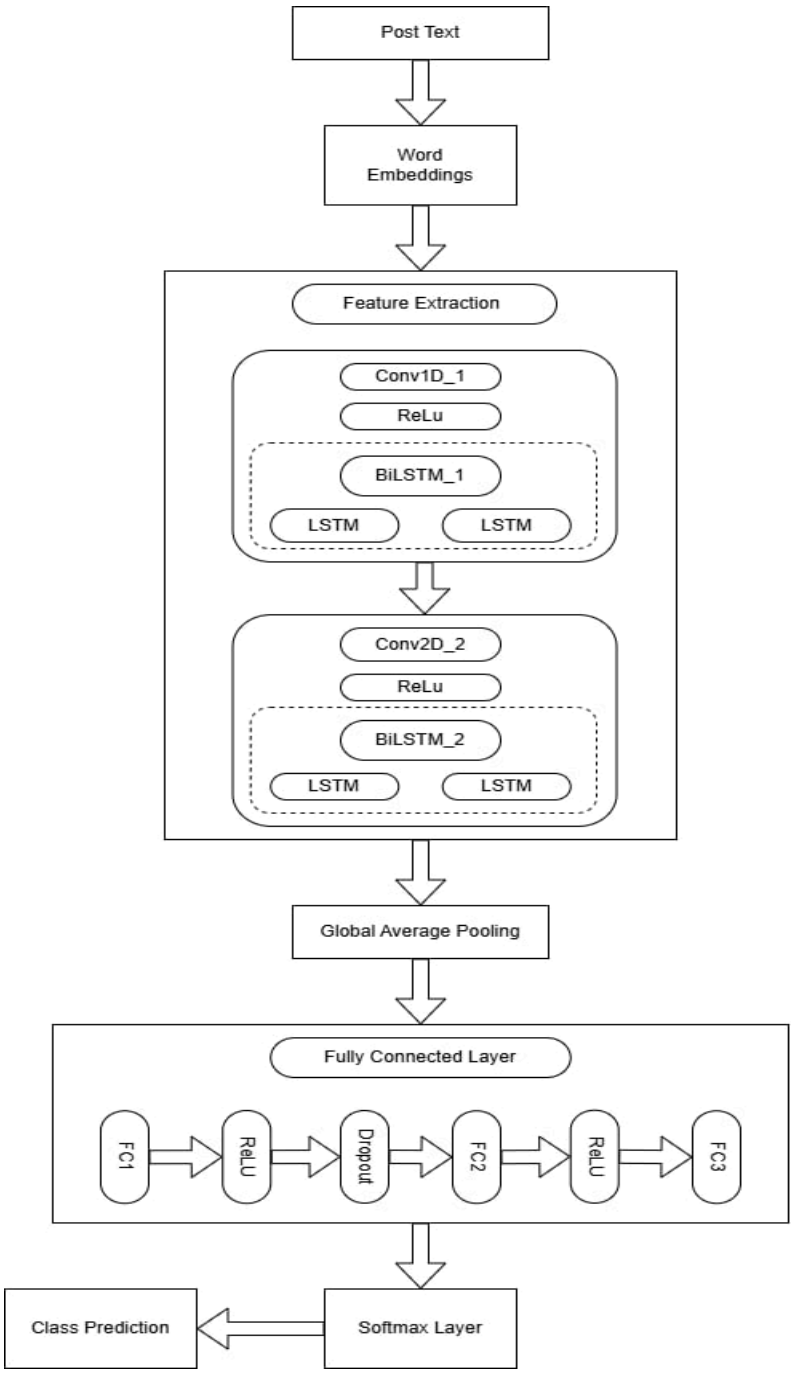
- Báo cáo phân loại cung cấp độ chính xác (precision), độ nhạy (recall), và điểm F1 cho từng nhãn.

2. BiLSTM+CNN:

2.1 Kiến trúc mô hình:

- Mô hình được xây dựng với các lớp:
    - **BiLSTM**: Cải thiện so với LSTM nhờ học được ngữ cảnh theo 2 chiều. Điều này cho phép mô hình nắm bắt thông tin từ cả trước và sau mỗi từ trong văn bản, giúp hiểu rõ hơn về ngữ cảnh tổng thể, tối ưu hóa khả năng nắm bắt ngữ cảnh dài hạn.
    - **Conv1D**: Được áp dụng trực tiếp trên các embeddings của văn bản để phát hiện các mẫu đặc trưng cục bộ (local patterns). Kết quả từ Conv1D là một tập hợp các bản đồ đặc trưng (feature maps) giàu thông tin ngữ cảnh cục bộ. CNN giảm khối lượng tính toán bằng cách trích xuất trước các đặc trưng quan trọng, làm giảm khả năng overfitting của mô hình.
    - **Global Average Pooling (GAP)**: Giúp tóm tắt thông tin toàn bộ chuỗi bằng cách tính trung bình giá trị trên chiều không gian, giảm kích thước đầu ra từ ma trận xuống vector, từ đó giảm độ phức tạp, giảm nguy cơ overfitting và giữ lại thông tin ngữ cảnh tổng thể, đồng thời linh hoạt với chuỗi có độ dài khác nhau.
    - **Fully Connected (FC):** biến đổi các đặc trưng trích xuất thành biểu diễn dạng vector, giúp mô hình học mối quan hệ phi tuyến giữa các đặc trưng và đầu ra. FC kết nối các đặc trưng với nhãn phân loại, tăng khả năng biểu diễn, tổng quát hóa và hoàn thiện dự đoán cuối cùng.

- Sơ đồ mô hình:



2.2 Cài đặt mô hình:

1. Cài đặt mô hình pre-trained Word2Vec cho tiếng việt - PhoW2V:

- **PhoW2V** cung cấp các bộ các từ nhúng cấp độ từ và âm tiết **Word2Vec** được đào tạo trước cho tiếng Việt, mô hình được đào tạo trước trên kho dữ liệu 20GB văn bản tiếng Việt và được sử dụng cho bài báo "Anh Tuan Nguyen, Mai Hoang Dao, and Dat Quoc Nguyen. 2020. A Pilot Study of Text-to-SQL Semantic Parsing for Vietnamese. In Findings of the Association for Computational Linguistics: EMNLP 2020".
- Mô hình gồm 1587505 từ với mỗi từ được embedding với 2 phiên bản số chiều 100 và 300.

- Dữ liệu mô hình được lưu vào 2 file .txt: word2vec_vi_words_100dims.txt và word2vec_vi_words_300dims.txt
- Tạo vocab từ mô hình:

```
[ ]  from gensim.models import KeyedVectors

     word2vec_file = "/content/gdrive/MyDrive/DL/word2vec_vi_words_300dims.txt"
     embed_lookup = KeyedVectors.load_word2vec_format(word2vec_file, binary=False)
```

```
pretrained_words = embed_lookup.index_to_key
pretrained_words = pretrained_words[2:]
print(pretrained_words[:10])
print("Size of Vocab: {}\n".format(len(pretrained_words)))
print("Embedding dim: {}\n".format(len(embed_lookup[pretrained_words[0]])))
```

```
['và', 'của', 'là', 'các', 'có', 'được', 'trong', 'cho', 'đã', 'với']
Size of Vocab: 1587505

Embedding dim: 300
```

- Thử nghiệm tìm các từ tương đồng với mô hình:

```
find_similar_to = 'bóng_đá'
print('Similar words to ' + find_similar_to + ': \n')

for similar_word in embed_lookup.similar_by_word(find_similar_to):
    print("Word: {0}, Similarity: {1:.3f}".format(similar_word[0], similar_word[1]))
```

```
Similar words to bóng_đá:

Word: Bóng_đá, Similarity: 0.747
Word: cầu_thủ, Similarity: 0.679
Word: cá_độ, Similarity: 0.654
Word: cá_cược, Similarity: 0.625
Word: đội_tuyển, Similarity: 0.608
Word: đấu, Similarity: 0.606
Word: thi_đấu, Similarity: 0.606
Word: World_Cup, Similarity: 0.606
Word: Oballa, Similarity: 0.605
Word: sân_cỏ, Similarity: 0.599
```

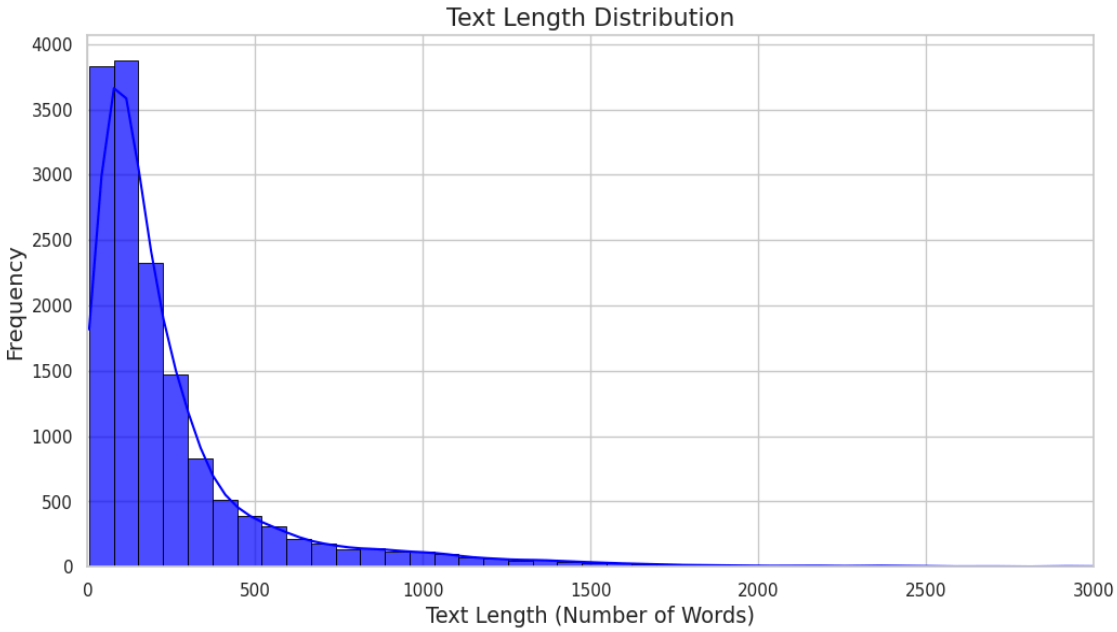
2. Tokenization và padding dữ liệu:

- Phân tích độ dài các văn bản:

```
Zero-length texts: 0
Minimum text length: 5
Maximum text length: 7349
Average text length: 262.33
Median text length: 147.0


Zero-length texts: 0
Minimum text length: 8
Maximum text length: 6940
Average text length: 270.00
Median text length: 150.0
```

- Không chứa đoạn text trống, độ dài văn bản trung bình với dữ luyện huấn luyện khoảng 262 từ => độ dài tối ưu sử dụng cho padding là từ 262 - 300 từ.

- Tách từ của mỗi văn bản thành các từ sử dụng thư viện 2 thư viện nltk và underthesea, một thư viện chuyên dùng để hỗ trợ xử lý ngôn ngữ Tiếng Việt. Nó giúp các từ Tiếng Việt một cách chính xác hơn. VD: bóng đá -> bóng_đá thay vì tách thành 2 từ bóng và đá, giúp các từ được ánh xạ đến từ điển pr



e-trained w2v chính xác hơn.

```python
from underthesea import word_tokenize

def tokenize(embed_lookup, texts):
    tokenized_texts = []

    for text in texts:
        tokens = word_tokenize(text)

        text_indices = []
        for word in tokens:
            try:
                word_idx = embed_lookup.key_to_index[word]
            except KeyError:
                word_idx = 0
            text_indices.append(word_idx)

        tokenized_texts.append(text_indices)

    return tokenized_texts
```

- Các đoạn văn bản được thiết lập với độ dài tối đa đã được thiết lập, tiến hành cắt ngắn các văn bản vượt quá độ dài trên và padding với các đoạn ngắn hơn. Các từ không có trong vocab được chuyển thành 0 (trong vocab không chứa chỉ mục này) và thêm padding là 1 ở đầu văn bản với các đoạn ngắn hơn độ dài tối đa.

```python
def pad_features(tokenized_texts, seq_length):

    features = np.ones((len(tokenized_texts), seq_length), dtype=int)

    for i, row in enumerate(tokenized_texts):
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features
```

```python
seq_length = 300

train_features = pad_features(tokenized_text_train, seq_length=seq_length)

assert len(train_features)==len(tokenized_text_train)
assert len(train_features[0])==seq_length

test_features = pad_features(tokenized_text_test, seq_length=seq_length)

assert len(test_features)==len(tokenized_text_test)
assert len(test_features[0])==seq_length

print(train_features[:20,:10])
print("\n")
print(test_features[:20,:10])
```

=> Kết quả các đoạn văn bản có độ dài cố định với các từ thành chuyển thành các chỉ mục với thiết lập bên trên.

```
[     1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      1      1      1      1      1      1      1      1      1
      1      0      0  17663  16602    129  61198    193     46  61198
   3641    159   4948    409   2090     38    822   1433    528    932
   6109   8257    356   1976   3137   3137   2100   6451    860   2418
    932    567   1820   2642   5779    331   3931   3879   1331      3
      0    356   1976      6   2727    331  97082     20    159   4948
      6   2727    458      7  12248    332    220   2156      0     20
    331  97082      9    562    280  42384    356   1976      6   4099
     20    159  11547      2  42384      2  42384   8526   2531      5
    123      0   1270   1456   4342      6   1338   5872   1870    502
      0   2727     67      6     32   2727    234    140      8   2727
     49   3641   4948      6  12310 103179   2338    331   1952    383
     49      6    454  12310 220686 193564   2338   1126   1300    478
      7  12248   3720     59   4099    822     14  11223    551  14856
   5075    303   7283  14856   2518   1905   1370    756     43      9
   2100   6451  21249      7   1227    128      9     89   1775     86
    695     14    153   4574    103     55   1042  54700   1193   2582
     25     31      0    891  95392   2464   2144      0 146634    147]
```

3. Chia dữ liệu:

- Dữ liệu liệu huấn luyện được chia với tỉ lệ các tập train, validation lần lượt là: 80% và 20%
- Dữ liệu thử nghiệm vẫn dữ nguyên số lượng sau khi tiền xử lý

```
                        Feature Shapes:
Train set:              (12047, 300)
Validation set:         (3012, 300)
Test set:               (9642, 300)
```

- Cấu hình các loader:

```python
import torch
from torch.utils.data import TensorDataset, DataLoader

batch_size = 128

train_data = TensorDataset(torch.from_numpy(train_x), torch.from_numpy(train_y))
valid_data = TensorDataset(torch.from_numpy(val_x), torch.from_numpy(val_y))
test_data = TensorDataset(torch.from_numpy(test_x), torch.from_numpy(test_y))

trainloader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
valloader = DataLoader(valid_data, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
testloader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)
```

2.3 Huấn luyện mô hình:

- **Các tham số của mô hình:**
  - **vocab_size**: Kích thước của vocab (số từ trong pre-trained model)

    - **embedding_dim**: Số chiều embedding (số chiều embedding của pre-trained model)
    - **num_classes**: Số lượng nhãn đầu ra.
    - **embed_model:** Mô hình phoW2V.
    - **freeze_embeddings=True**: Trọng số của embedding layer được cố định (không thay đổi) trong suốt quá trình huấn luyện.
- **Các tham số của các lớp trong mô hình:**
    - **Lớp Conv1D_1**: in_channels=embedding_dim, out_channels=512, kernel_size=3, padding=1)
    - **Lớp BiLSTM_1**: input_size=512, hidden_size=128, num_layers=1, bidirectional=True, batch_first=True
    - **Lớp Conv1D_2:** in_channels=256, out_channels=512, kernel_size=3, padding=1)
    - **Lớp BiLSTM_2:** input_size=512, hidden_size=128, num_layers=1, bidirectional=True, batch_first=True
    - **Lớp Dropout**: 0.6.
    - **Lớp FC1**: (256, 512)
    - **Lớp FC2**: (512, 128)
    - **Lớp FC3:** (128, 23)

  => Các lớp Conv1D có đầu ra lớn hơn đầu vào giúp mở rộng các đặc trưng của văn bản, giúp lớp BiLSTM phía sau dễ dàng nắm bắt các đặc trưng quan trọng. Các lớp BiLSTM có đầu ra nhỏ hơn đầu vào giúp mô hình tránh overfitting và nén các đặc trưng thông tin theo chuỗi dài. Lớp Fully Connected đầu tiên mở rộng đặc trưng từ lớp BiLSTM cuối cùng trước khi trích xuất các đặc trưng quan trọng trong các lớp tiếp theo giúp mô hình đưa ra kết quả đầu ra chính xác nhất.
- **Hàm tối ưu và đánh giá:**
    - **optimizer = Adam:** Hàm tối ưu phổ biến.
    - **criterion = CrossEntropyLoss**: Hàm mất mát phù hợp với bài toán phân loại nhiều lớp
- **Các siêu tham số của mô hình:**
    - **Batch_size = 128 (baseline)**: Số lượng batch_size vừa đủ lớn để tối ưu tốc độ của mô hình mà vẫn đủ để trích xuất các đặc trưng.
    - **optimizer = Adam**: Hàm tối ưu phổ biến.
    - **learning rate = 0.001:** Tốc độ học phổ biến.
    - **epochs = 25 (baseline)**: Số lượng epoch vừa đủ để mô hình hội tụ.
    - **seq_length** = 270 (baseline): Số lượng độ dài văn bản phù hợp.
- **Các thiết lập khác:**
    - **device = cuda**: Huấn luyện mô hình trên GPU vì BiLSTM huấn luyện rất lâu trên CPU.
- **Các phương pháp huấn luyện:**
    - **Early Stopping:** Sử dụng để kết thúc huấn luyện khi mô hình có dấu hiệu overfitting.
    - **Lowest validation loss saving**: Lưu lại mô hình trong epoch có validation loss thấp nhất.

- **Train Loss và Train Accuracy**: Theo dõi quá trình học của mô hình với tập huấn luyện.

3. BERT

3.1 Kiến trúc mô hình sử dụng:

- **Tokenization (Phân tách từ)**: Văn bản được chia thành các token (từ hoặc các phần của từ). BERT sử dụng một kỹ thuật gọi là WordPiece tokenization để chia nhỏ các từ không phổ biến thành các phần nhỏ hơn.
- **Adding Special Tokens (Thêm các token đặc biệt)**: BERT yêu cầu thêm hai token đặc biệt vào văn bản: [CLS] (Classification) ở đầu văn bản và [SEP] (Separator) ở cuối câu hoặc giữa các câu.
- **Token Embeddings (Tạo các biểu diễn cho token)**: Mỗi token được chuyển đổi thành một vector có chiều dài cố định (thường là 768 hoặc 1024 chiều). Các vector này đại diện cho ý nghĩa ngữ nghĩa của các token trong ngữ cảnh của câu.
- **Positional Embeddings (Biểu diễn vị trí)**: Thêm thông tin về vị trí của các token trong câu để mô hình hiểu được thứ tự từ.
- **Segment Embeddings (Biểu diễn đoạn)**: Nếu văn bản bao gồm nhiều câu, BERT thêm thông tin để xác định mỗi câu thuộc về đoạn nào.

3.2 Cách triển khai mô hình

- Xây dựng hàm BERTembedding để embedding cho các token đầu vào

```
class BERTEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_size, max_len, segment_size=2):
        super(BERTEmbedding, self).__init__()
        self.token_embedding = nn.Embedding(vocab_size, embed_size)
        self.position_embedding = nn.Embedding(max_len, embed_size)
        self.segment_embedding = nn.Embedding(segment_size, embed_size)
        self.layer_norm = nn.LayerNorm(embed_size)
```

Token Embedding: Biểu diễn các từ bằng vector.
Position Embedding: Sử dụng chỉ số của từ trong câu để mã hóa vị trí.
Segment Embedding: Mã hóa xem token thuộc câu nào.

- Thực hiện chú ý nhiều đầu để học các mối quan hệ giữa các từ trong câu

```
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, embed_size, num_heads):
        super(MultiHeadSelfAttention, self).__init__()
        self.query = nn.Linear(embed_size, embed_size)
        self.key = nn.Linear(embed_size, embed_size)
        self.value = nn.Linear(embed_size, embed_size)
        self.fc_out = nn.Linear(embed_size, embed_size)
```

- Lớp FeedForwart để tăng cường khả năng học biểu diễn phi tuyến của mô hình

```
class FeedForward(nn.Module):
    def __init__(self, embed_size, hidden_size):
        super(FeedForward, self).__init__()
        self.fc1 = nn.Linear(embed_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, embed_size)
        self.relu = nn.ReLU()
```

- Kết hợp các thành phần để tạo ra mô hình BERT

```
class BERTModel(nn.Module):
    def __init__(self, vocab_size, embed_size, max_len, num_heads, hidden_size, num_layers
        super(BERTModel, self).__init__()
        self.embedding = BERTEmbedding(vocab_size, embed_size, max_len)
        self.layers = nn.ModuleList([
            TransformerEncoderLayer(embed_size, num_heads, hidden_size) for _ in range(num
        ])
        self.fc = nn.Linear(embed_size, len(label_map))
```

dự đoán đầu ra từ token

```
def forward(self, token_ids, segment_ids, mask=None):
    x = self.embedding(token_ids, segment_ids)
    for layer in self.layers:
        x = layer(x, mask)
    logits = self.fc(x[:, 0, :])
    return logits
```

IV. Kết quả và đánh giá

1. LSTM (baseline):

Kết quả trên tập kiểm tra:

```
# Đánh giá trên tập test
loss, accuracy = model.evaluate(test_padded, test_labels_encoded)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7844 - loss: 2.7936
Test Loss: 2.8399405479431152, Test Accuracy: 0.7801517844200134
```

- **Độ mất mát (Loss):** Giá trị mất mát trên tập kiểm tra là thước đo lỗi dự đoán của mô hình.
- **Độ chính xác (Accuracy):** Độ chính xác đo tỷ lệ dự đoán đúng trên tổng số mẫu.

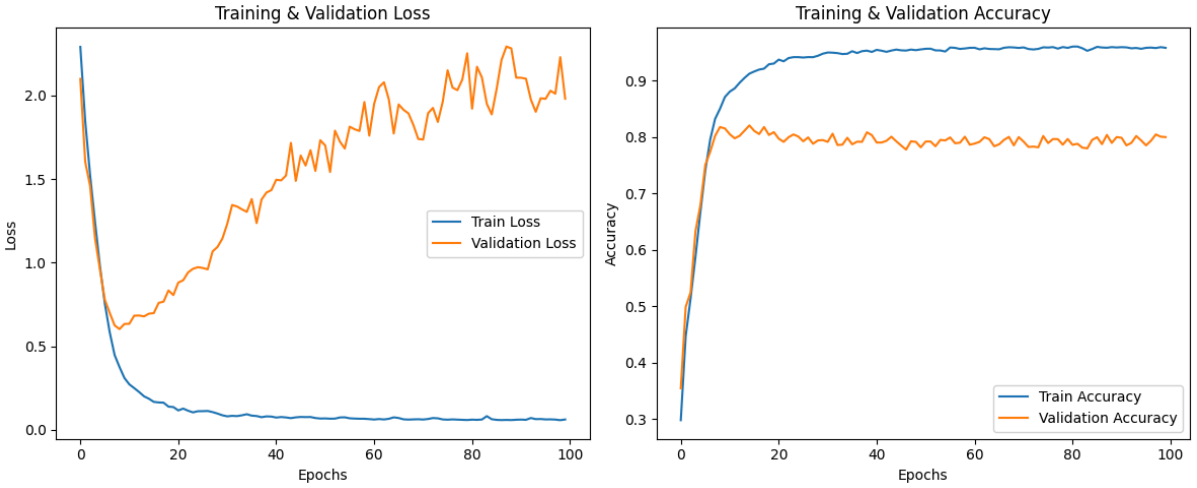
Báo cáo chi tiết:

```
from sklearn.metrics import classification_report

test_predictions = model.predict(test_padded)
test_predicted_labels = test_predictions.argmax(axis=1)

# Tạo Classification Report
report = classification_report(test_labels_encoded, test_predicted_labels)
print(report)
```

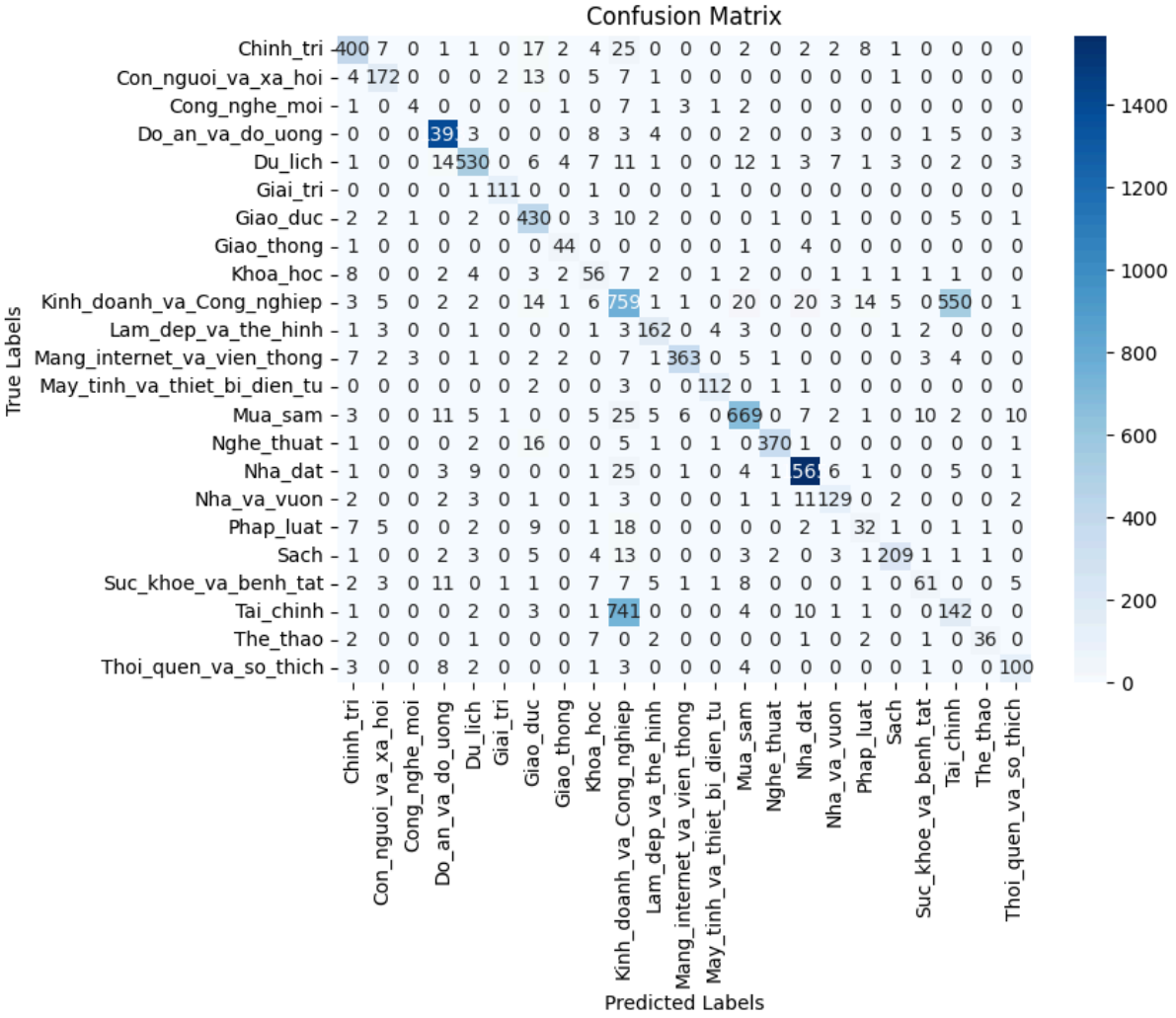
```
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
              precision    recall  f1-score   support

           0       0.92      0.85      0.88       472
           1       0.89      0.74      0.81       205
           2       0.67      0.50      0.57        20
           3       0.97      0.98      0.97      1425
           4       0.84      0.90      0.87       606
           5       0.93      0.98      0.96       114
           6       0.87      0.90      0.88       460
           7       0.78      0.94      0.85        50
           8       0.55      0.48      0.51        92
           9       0.45      0.52      0.48      1407
          10       0.88      0.91      0.89       181
          11       0.94      0.93      0.93       401
          12       0.96      0.92      0.94       119
          13       0.88      0.86      0.87       762
          14       0.95      0.93      0.94       398
          15       0.97      0.97      0.97      1623
          16       0.74      0.81      0.77       158
          17       0.42      0.47      0.44        80
          18       0.91      0.89      0.90       249
          19       0.62      0.62      0.62       114
          20       0.17      0.13      0.15       906
          21       0.85      0.77      0.81        52
          22       0.78      0.83      0.80       122

    accuracy                           0.78     10016
   macro avg       0.78      0.77      0.78     10016
weighted avg       0.78      0.78      0.78     10016
```

- **Precision (Độ chính xác):** Tỷ lệ dự đoán đúng trên tổng số dự đoán cho một nhãn.
- **Recall (Độ nhạy):** Tỷ lệ dự đoán đúng trên tổng số mẫu thực sự thuộc về nhãn đó.
- **F1-score:** Trung bình điều hòa giữa precision và recall, đo lường hiệu quả tổng thể.

- **Biểu đồ Training and Validation Loss và Training and Validation Accuracy theo Epoch:**



- **Confusion matrix**



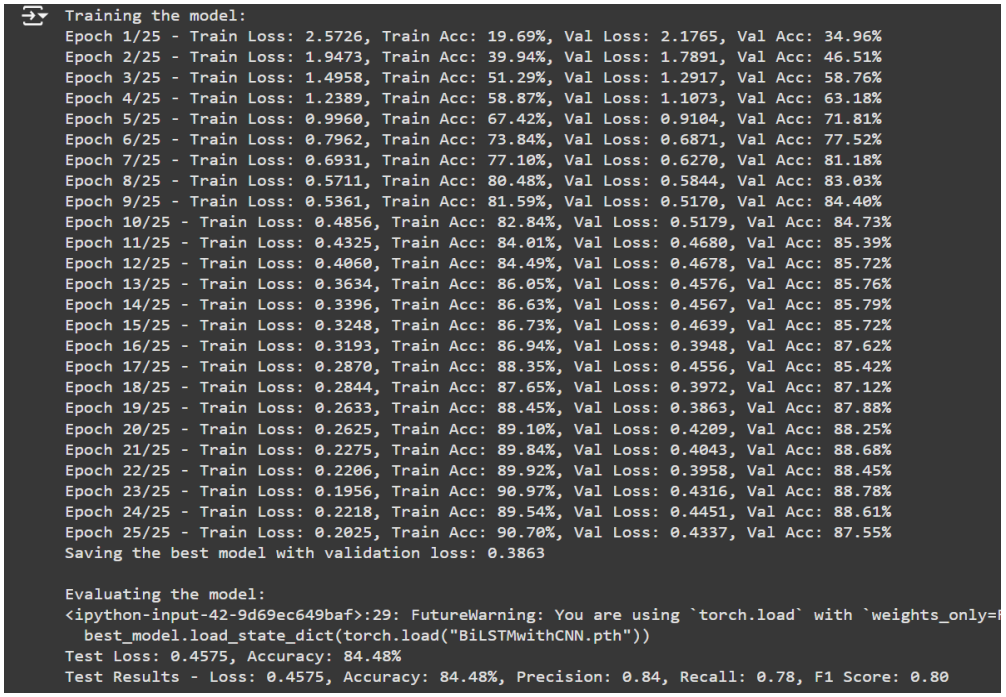
Đánh giá:

- **Ưu điểm:**
    - Mô hình đạt độ chính xác cao trên tập kiểm tra.
    - Báo cáo chi tiết cho thấy sự cân bằng tốt giữa các nhãn.
    - LSTM xử lý tốt các mối quan hệ ngữ nghĩa dài trong văn bản.
- **Nhược điểm:**
    - Thời gian huấn luyện lâu với số epoch lớn.
    - Yêu cầu tài nguyên tính toán cao.

2. BiLSTM+CNN:

- **Các phương pháp đánh giá:**
    - CrossEntropyLoss: Hàm mất mát phù hợp với bài toán phân loại nhiều lớp
    - Validation Loss: Kiểm tra loss trong quá trình huấn luyện để đánh giá overfitting của mô hình và hỗ trợ lưu mô hình tốt nhất.
    - Accuracy Loss: Kiểm tra độ chính xác của mô hình trên tập validation.
    - Test Loss, Test Accuracy, Precision, Recall, F1, Score: Đánh giá mức độ chính xác và tổng quát với dữ liệu chưa nhìn thấy.
- **Các phương pháp trực quan hóa đánh giá:**
    - Vẽ biểu đồ Train Loss vs Validation Loss và Train Accuracy vs Validation Accuracy.
    - Vẽ confusion matrix cho test data.
- **Đánh giá baseline của mô hình:**
    - Thiết lập mặc định:
        - Batch_size = 128
        - embed model = 100w2v
        - seq_length = 270word
        - Pooling layer: Global average pooling
        - optimizer = Adam
        - tokenizer = nltk
        - DropOut = 0.6

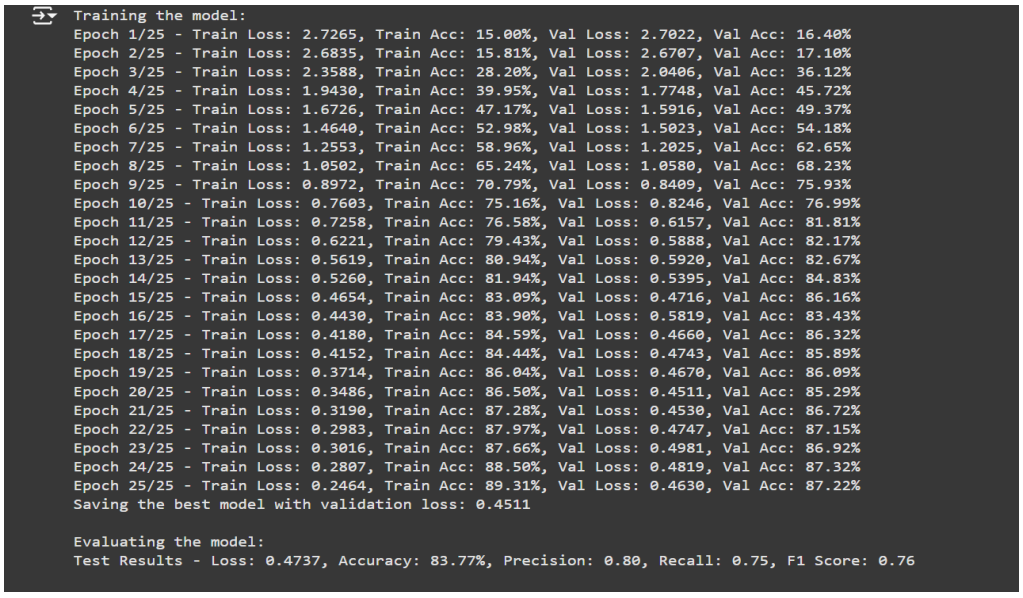
- Baseline đầu tiên cho hiệu quả khá tốt trên tập test với thông tin quá trình huấn luyện và đánh giá như sau:



```
Training the model:
Epoch 1/25 - Train Loss: 2.5726, Train Acc: 19.69%, Val Loss: 2.1765, Val Acc: 34.96%
Epoch 2/25 - Train Loss: 1.9473, Train Acc: 39.94%, Val Loss: 1.7891, Val Acc: 46.51%
Epoch 3/25 - Train Loss: 1.4958, Train Acc: 51.29%, Val Loss: 1.2917, Val Acc: 58.76%
Epoch 4/25 - Train Loss: 1.2389, Train Acc: 58.87%, Val Loss: 1.1073, Val Acc: 63.18%
Epoch 5/25 - Train Loss: 0.9960, Train Acc: 67.42%, Val Loss: 0.9104, Val Acc: 71.81%
Epoch 6/25 - Train Loss: 0.7962, Train Acc: 73.84%, Val Loss: 0.6871, Val Acc: 77.52%
Epoch 7/25 - Train Loss: 0.6931, Train Acc: 77.10%, Val Loss: 0.6270, Val Acc: 81.18%
Epoch 8/25 - Train Loss: 0.5711, Train Acc: 80.48%, Val Loss: 0.5844, Val Acc: 83.03%
Epoch 9/25 - Train Loss: 0.5361, Train Acc: 81.59%, Val Loss: 0.5170, Val Acc: 84.40%
Epoch 10/25 - Train Loss: 0.4856, Train Acc: 82.84%, Val Loss: 0.5179, Val Acc: 84.73%
Epoch 11/25 - Train Loss: 0.4325, Train Acc: 84.01%, Val Loss: 0.4680, Val Acc: 85.39%
Epoch 12/25 - Train Loss: 0.4060, Train Acc: 84.49%, Val Loss: 0.4678, Val Acc: 85.72%
Epoch 13/25 - Train Loss: 0.3634, Train Acc: 86.05%, Val Loss: 0.4576, Val Acc: 85.76%
Epoch 14/25 - Train Loss: 0.3396, Train Acc: 86.63%, Val Loss: 0.4567, Val Acc: 85.79%
Epoch 15/25 - Train Loss: 0.3248, Train Acc: 86.73%, Val Loss: 0.4639, Val Acc: 85.72%
Epoch 16/25 - Train Loss: 0.3193, Train Acc: 86.94%, Val Loss: 0.3948, Val Acc: 87.62%
Epoch 17/25 - Train Loss: 0.2870, Train Acc: 88.35%, Val Loss: 0.4556, Val Acc: 85.42%
Epoch 18/25 - Train Loss: 0.2844, Train Acc: 87.65%, Val Loss: 0.3972, Val Acc: 87.12%
Epoch 19/25 - Train Loss: 0.2633, Train Acc: 88.45%, Val Loss: 0.3863, Val Acc: 87.88%
Epoch 20/25 - Train Loss: 0.2625, Train Acc: 89.10%, Val Loss: 0.4209, Val Acc: 88.25%
Epoch 21/25 - Train Loss: 0.2275, Train Acc: 89.84%, Val Loss: 0.4043, Val Acc: 88.68%
Epoch 22/25 - Train Loss: 0.2206, Train Acc: 89.92%, Val Loss: 0.3958, Val Acc: 88.45%
Epoch 23/25 - Train Loss: 0.1956, Train Acc: 90.97%, Val Loss: 0.4316, Val Acc: 88.78%
Epoch 24/25 - Train Loss: 0.2218, Train Acc: 89.54%, Val Loss: 0.4451, Val Acc: 88.61%
Epoch 25/25 - Train Loss: 0.2025, Train Acc: 90.70%, Val Loss: 0.4337, Val Acc: 87.55%
Saving the best model with validation loss: 0.3863

Evaluating the model:
<ipython-input-42-9d69ec649baf>:29: FutureWarning: You are using `torch.load` with `weights_only=
  best_model.load_state_dict(torch.load("BiLSTMwithCNN.pth"))
Test Loss: 0.4575, Accuracy: 84.48%
Test Results - Loss: 0.4575, Accuracy: 84.48%, Precision: 0.84, Recall: 0.78, F1 Score: 0.80
```

=> Các thông số của mô hình khả quan trên tập test.

- **Các điều chỉnh mô hình với embed_model 100:**
  - **Điều chỉnh lớp BiLSTM:**
  - Tăng output_size 2 lớp BiLSTM: 128 -> 256 với thông tin quá trình huấn luyện và đánh giá như sau:



```
Training the model:
Epoch 1/25 - Train Loss: 2.7265, Train Acc: 15.00%, Val Loss: 2.7022, Val Acc: 16.40%
Epoch 2/25 - Train Loss: 2.6835, Train Acc: 15.81%, Val Loss: 2.6707, Val Acc: 17.10%
Epoch 3/25 - Train Loss: 2.3588, Train Acc: 28.20%, Val Loss: 2.0406, Val Acc: 36.12%
Epoch 4/25 - Train Loss: 1.9430, Train Acc: 39.95%, Val Loss: 1.7748, Val Acc: 45.72%
Epoch 5/25 - Train Loss: 1.6726, Train Acc: 47.17%, Val Loss: 1.5916, Val Acc: 49.37%
Epoch 6/25 - Train Loss: 1.4640, Train Acc: 52.98%, Val Loss: 1.5023, Val Acc: 54.18%
Epoch 7/25 - Train Loss: 1.2553, Train Acc: 58.96%, Val Loss: 1.2025, Val Acc: 62.65%
Epoch 8/25 - Train Loss: 1.0502, Train Acc: 65.24%, Val Loss: 1.0580, Val Acc: 68.23%
Epoch 9/25 - Train Loss: 0.8972, Train Acc: 70.79%, Val Loss: 0.8409, Val Acc: 75.93%
Epoch 10/25 - Train Loss: 0.7603, Train Acc: 75.16%, Val Loss: 0.8246, Val Acc: 76.99%
Epoch 11/25 - Train Loss: 0.7258, Train Acc: 76.58%, Val Loss: 0.6157, Val Acc: 81.81%
Epoch 12/25 - Train Loss: 0.6221, Train Acc: 79.43%, Val Loss: 0.5888, Val Acc: 82.17%
Epoch 13/25 - Train Loss: 0.5619, Train Acc: 80.94%, Val Loss: 0.5920, Val Acc: 82.67%
Epoch 14/25 - Train Loss: 0.5260, Train Acc: 81.94%, Val Loss: 0.5395, Val Acc: 84.83%
Epoch 15/25 - Train Loss: 0.4654, Train Acc: 83.09%, Val Loss: 0.4716, Val Acc: 86.16%
Epoch 16/25 - Train Loss: 0.4430, Train Acc: 83.90%, Val Loss: 0.5819, Val Acc: 83.43%
Epoch 17/25 - Train Loss: 0.4180, Train Acc: 84.59%, Val Loss: 0.4660, Val Acc: 86.32%
Epoch 18/25 - Train Loss: 0.4152, Train Acc: 84.44%, Val Loss: 0.4743, Val Acc: 85.89%
Epoch 19/25 - Train Loss: 0.3714, Train Acc: 86.04%, Val Loss: 0.4670, Val Acc: 86.09%
Epoch 20/25 - Train Loss: 0.3486, Train Acc: 86.50%, Val Loss: 0.4511, Val Acc: 85.29%
Epoch 21/25 - Train Loss: 0.3190, Train Acc: 87.28%, Val Loss: 0.4530, Val Acc: 86.72%
Epoch 22/25 - Train Loss: 0.2983, Train Acc: 87.97%, Val Loss: 0.4747, Val Acc: 87.15%
Epoch 23/25 - Train Loss: 0.3016, Train Acc: 87.66%, Val Loss: 0.4981, Val Acc: 86.92%
Epoch 24/25 - Train Loss: 0.2807, Train Acc: 88.50%, Val Loss: 0.4819, Val Acc: 87.32%
Epoch 25/25 - Train Loss: 0.2464, Train Acc: 89.31%, Val Loss: 0.4630, Val Acc: 87.22%
Saving the best model with validation loss: 0.4511

Evaluating the model:
Test Results - Loss: 0.4737, Accuracy: 83.77%, Precision: 0.80, Recall: 0.75, F1 Score: 0.76
```

  - Tăng num layer từ 1 -> 2:

```
Training the model:
Epoch 1/25 - Train Loss: 2.7105, Train Acc: 15.23%, Val Loss: 2.6767, Val Acc: 16.40%
Epoch 2/25 - Train Loss: 2.6039, Train Acc: 18.54%, Val Loss: 2.3596, Val Acc: 30.78%
Epoch 3/25 - Train Loss: 2.1405, Train Acc: 36.38%, Val Loss: 1.9041, Val Acc: 42.80%
Epoch 4/25 - Train Loss: 1.8037, Train Acc: 44.90%, Val Loss: 1.6391, Val Acc: 48.51%
Epoch 5/25 - Train Loss: 1.5613, Train Acc: 50.34%, Val Loss: 1.7799, Val Acc: 48.11%
Epoch 6/25 - Train Loss: 1.3165, Train Acc: 56.50%, Val Loss: 1.1510, Val Acc: 63.01%
Epoch 7/25 - Train Loss: 1.1019, Train Acc: 62.63%, Val Loss: 1.2178, Val Acc: 61.55%
Epoch 8/25 - Train Loss: 0.9998, Train Acc: 65.91%, Val Loss: 0.9834, Val Acc: 69.12%
Epoch 9/25 - Train Loss: 0.8569, Train Acc: 71.22%, Val Loss: 0.7781, Val Acc: 74.93%
Epoch 10/25 - Train Loss: 0.7352, Train Acc: 75.35%, Val Loss: 0.6756, Val Acc: 80.15%
Epoch 11/25 - Train Loss: 0.6739, Train Acc: 78.13%, Val Loss: 0.6522, Val Acc: 80.78%
Epoch 12/25 - Train Loss: 0.5810, Train Acc: 79.97%, Val Loss: 0.6848, Val Acc: 79.75%
Epoch 13/25 - Train Loss: 0.5520, Train Acc: 80.83%, Val Loss: 0.5459, Val Acc: 83.76%
Epoch 14/25 - Train Loss: 0.5134, Train Acc: 82.18%, Val Loss: 0.5351, Val Acc: 84.50%
Epoch 15/25 - Train Loss: 0.4858, Train Acc: 82.78%, Val Loss: 0.4944, Val Acc: 84.56%
Epoch 16/25 - Train Loss: 0.4481, Train Acc: 83.75%, Val Loss: 0.5509, Val Acc: 84.33%
Epoch 17/25 - Train Loss: 0.4312, Train Acc: 83.94%, Val Loss: 0.4821, Val Acc: 85.86%
Epoch 18/25 - Train Loss: 0.3798, Train Acc: 85.56%, Val Loss: 0.5232, Val Acc: 84.63%
Epoch 19/25 - Train Loss: 0.3722, Train Acc: 85.58%, Val Loss: 0.4792, Val Acc: 84.59%
Epoch 20/25 - Train Loss: 0.3329, Train Acc: 86.87%, Val Loss: 0.4938, Val Acc: 84.10%
Epoch 21/25 - Train Loss: 0.3403, Train Acc: 86.43%, Val Loss: 0.4500, Val Acc: 86.22%
Epoch 22/25 - Train Loss: 0.3115, Train Acc: 87.13%, Val Loss: 0.4733, Val Acc: 86.12%
Epoch 23/25 - Train Loss: 0.3041, Train Acc: 87.67%, Val Loss: 0.4242, Val Acc: 86.85%
Epoch 24/25 - Train Loss: 0.2764, Train Acc: 88.15%, Val Loss: 0.4482, Val Acc: 86.55%
Epoch 25/25 - Train Loss: 0.2543, Train Acc: 89.08%, Val Loss: 0.4580, Val Acc: 86.09%
Saving the best model with validation loss: 0.4242

Evaluating the model:
Test Results - Loss: 0.4930, Accuracy: 83.19%, Precision: 0.79, Recall: 0.75, F1 Score: 0.75
```

=>> Các điều chỉnh về lớp BiLSTM gây giảm ít nhiều đến hiệu suất mô hình.

- **Điều chỉnh lớp Conv1D:**
- Lớp Convo1D sẽ không điều chỉnh các thông số đầu vào và đầu ra vị có liên quan đến lớp Embedding layer và BiLSTM.
- Tăng kernel size từ 3 -> 5:

```
Training the model:
Epoch 1/25 - Train Loss: 2.6817, Train Acc: 17.68%, Val Loss: 2.4391, Val Acc: 30.68%
Epoch 2/25 - Train Loss: 2.3125, Train Acc: 31.51%, Val Loss: 2.1546, Val Acc: 36.19%
Epoch 3/25 - Train Loss: 1.9107, Train Acc: 43.70%, Val Loss: 1.8803, Val Acc: 44.75%
Epoch 4/25 - Train Loss: 1.6657, Train Acc: 48.34%, Val Loss: 1.5081, Val Acc: 52.92%
Epoch 5/25 - Train Loss: 1.4287, Train Acc: 54.36%, Val Loss: 1.3189, Val Acc: 59.59%
Epoch 6/25 - Train Loss: 1.2263, Train Acc: 60.15%, Val Loss: 1.1356, Val Acc: 64.77%
Epoch 7/25 - Train Loss: 1.0571, Train Acc: 64.77%, Val Loss: 1.0378, Val Acc: 67.53%
Epoch 8/25 - Train Loss: 0.9612, Train Acc: 68.45%, Val Loss: 0.9319, Val Acc: 72.14%
Epoch 9/25 - Train Loss: 0.8352, Train Acc: 71.75%, Val Loss: 0.7839, Val Acc: 76.33%
Epoch 10/25 - Train Loss: 0.7370, Train Acc: 75.60%, Val Loss: 0.6976, Val Acc: 79.65%
Epoch 11/25 - Train Loss: 0.6297, Train Acc: 78.51%, Val Loss: 0.6974, Val Acc: 79.95%
Epoch 12/25 - Train Loss: 0.5786, Train Acc: 80.33%, Val Loss: 0.5833, Val Acc: 82.97%
Epoch 13/25 - Train Loss: 0.5260, Train Acc: 81.80%, Val Loss: 0.5924, Val Acc: 79.78%
Epoch 14/25 - Train Loss: 0.5058, Train Acc: 81.87%, Val Loss: 0.5932, Val Acc: 83.00%
Epoch 15/25 - Train Loss: 0.4710, Train Acc: 83.15%, Val Loss: 0.5234, Val Acc: 84.16%
Epoch 16/25 - Train Loss: 0.4205, Train Acc: 84.32%, Val Loss: 0.5118, Val Acc: 84.56%
Epoch 17/25 - Train Loss: 0.3891, Train Acc: 85.39%, Val Loss: 0.4762, Val Acc: 85.46%
Epoch 18/25 - Train Loss: 0.3704, Train Acc: 85.64%, Val Loss: 0.4642, Val Acc: 86.09%
Epoch 19/25 - Train Loss: 0.3514, Train Acc: 86.54%, Val Loss: 0.4613, Val Acc: 84.23%
Epoch 20/25 - Train Loss: 0.3268, Train Acc: 87.18%, Val Loss: 0.4818, Val Acc: 85.62%
Epoch 21/25 - Train Loss: 0.3116, Train Acc: 87.59%, Val Loss: 0.4284, Val Acc: 86.42%
Epoch 22/25 - Train Loss: 0.2912, Train Acc: 87.97%, Val Loss: 0.4665, Val Acc: 85.92%
Epoch 23/25 - Train Loss: 0.2689, Train Acc: 88.59%, Val Loss: 0.4896, Val Acc: 86.79%
Epoch 24/25 - Train Loss: 0.2859, Train Acc: 88.15%, Val Loss: 0.4522, Val Acc: 87.38%
Epoch 25/25 - Train Loss: 0.2691, Train Acc: 88.68%, Val Loss: 0.5194, Val Acc: 84.36%
Saving the best model with validation loss: 0.4284

Evaluating the model:
Test Results - Loss: 0.5684, Accuracy: 81.65%, Precision: 0.77, Recall: 0.74, F1 Score: 0.74
```

- Tăng kernel size từ 3 -> 5 và thêm lớp max pooling 1D sau các lớp conv1D:

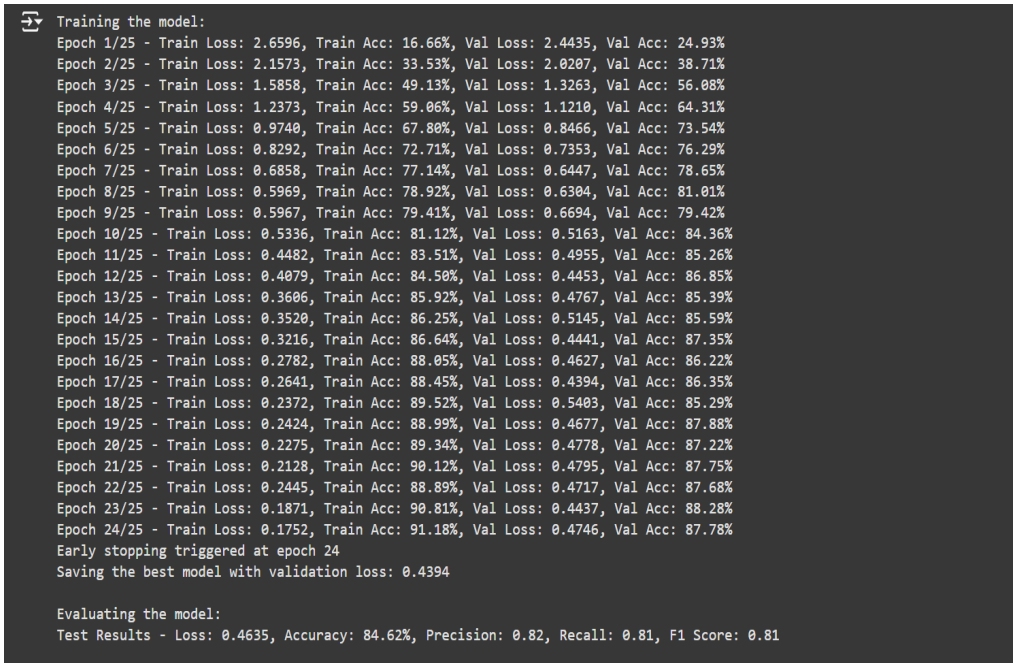
```
Training the model:
Epoch 1/25 - Train Loss: 2.6882, Train Acc: 16.31%, Val Loss: 2.6102, Val Acc: 25.40%
Epoch 2/25 - Train Loss: 2.3211, Train Acc: 30.61%, Val Loss: 2.2182, Val Acc: 33.33%
Epoch 3/25 - Train Loss: 2.2391, Train Acc: 33.36%, Val Loss: 2.2715, Val Acc: 33.76%
Epoch 4/25 - Train Loss: 2.0287, Train Acc: 40.08%, Val Loss: 1.8625, Val Acc: 44.62%
Epoch 5/25 - Train Loss: 1.6974, Train Acc: 47.95%, Val Loss: 1.7480, Val Acc: 45.95%
Epoch 6/25 - Train Loss: 1.4849, Train Acc: 51.93%, Val Loss: 1.3677, Val Acc: 56.51%
Epoch 7/25 - Train Loss: 1.2930, Train Acc: 57.33%, Val Loss: 1.2020, Val Acc: 61.49%
Epoch 8/25 - Train Loss: 1.1325, Train Acc: 62.17%, Val Loss: 1.0930, Val Acc: 63.71%
Epoch 9/25 - Train Loss: 1.0281, Train Acc: 65.39%, Val Loss: 0.9766, Val Acc: 69.32%
Epoch 10/25 - Train Loss: 0.9195, Train Acc: 68.95%, Val Loss: 0.9799, Val Acc: 70.72%
Epoch 11/25 - Train Loss: 0.8258, Train Acc: 71.77%, Val Loss: 0.8428, Val Acc: 75.00%
Epoch 12/25 - Train Loss: 0.7438, Train Acc: 74.87%, Val Loss: 0.7183, Val Acc: 78.45%
Epoch 13/25 - Train Loss: 0.6502, Train Acc: 77.99%, Val Loss: 0.6597, Val Acc: 81.01%
Epoch 14/25 - Train Loss: 0.6167, Train Acc: 79.21%, Val Loss: 0.6470, Val Acc: 81.08%
Epoch 15/25 - Train Loss: 0.5675, Train Acc: 80.31%, Val Loss: 0.6022, Val Acc: 82.84%
Epoch 16/25 - Train Loss: 0.5028, Train Acc: 82.21%, Val Loss: 0.5516, Val Acc: 83.83%
Epoch 17/25 - Train Loss: 0.4668, Train Acc: 83.46%, Val Loss: 0.6180, Val Acc: 81.27%
Epoch 18/25 - Train Loss: 0.4565, Train Acc: 83.58%, Val Loss: 0.6001, Val Acc: 83.13%
Epoch 19/25 - Train Loss: 0.4067, Train Acc: 84.84%, Val Loss: 0.5601, Val Acc: 84.73%
Epoch 20/25 - Train Loss: 0.3798, Train Acc: 85.61%, Val Loss: 0.4929, Val Acc: 85.69%
Epoch 21/25 - Train Loss: 0.3679, Train Acc: 85.98%, Val Loss: 0.5145, Val Acc: 84.59%
Epoch 22/25 - Train Loss: 0.3662, Train Acc: 85.81%, Val Loss: 0.5458, Val Acc: 84.79%
Epoch 23/25 - Train Loss: 0.3267, Train Acc: 86.98%, Val Loss: 0.4902, Val Acc: 86.32%
Epoch 24/25 - Train Loss: 0.2952, Train Acc: 87.98%, Val Loss: 0.5122, Val Acc: 86.12%
Epoch 25/25 - Train Loss: 0.3067, Train Acc: 87.17%, Val Loss: 0.5128, Val Acc: 85.52%
Saving the best model with validation loss: 0.4902

Evaluating the model:
Test Results - Loss: 0.5223, Accuracy: 82.09%, Precision: 0.77, Recall: 0.73, F1 Score: 0.74
```

=> Các điều chỉnh về thông số của lớp Conv1D gây giảm đáng kể hiệu suất của mô hình.

- **Điều chỉnh lớp Pooling:**
- Sử dụng Gobal max pooling thay cho Global average pooling:

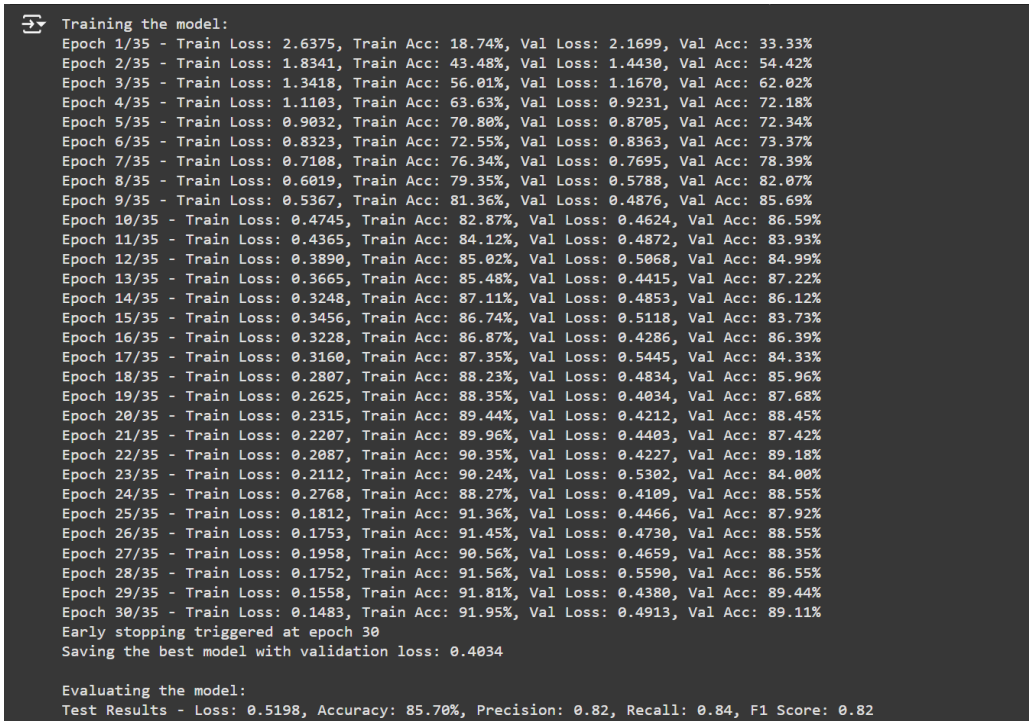
```
Training the model:
Epoch 1/25 - Train Loss: 2.6687, Train Acc: 17.49%, Val Loss: 2.4099, Val Acc: 33.40%
Epoch 2/25 - Train Loss: 2.1897, Train Acc: 35.97%, Val Loss: 2.0493, Val Acc: 40.44%
Epoch 3/25 - Train Loss: 1.8010, Train Acc: 44.95%, Val Loss: 1.7415, Val Acc: 47.58%
Epoch 4/25 - Train Loss: 1.5177, Train Acc: 51.79%, Val Loss: 1.3855, Val Acc: 58.60%
Epoch 5/25 - Train Loss: 1.2488, Train Acc: 59.73%, Val Loss: 1.2208, Val Acc: 62.85%
Epoch 6/25 - Train Loss: 1.0229, Train Acc: 66.61%, Val Loss: 0.8507, Val Acc: 74.37%
Epoch 7/25 - Train Loss: 0.8095, Train Acc: 73.12%, Val Loss: 0.7501, Val Acc: 77.16%
Epoch 8/25 - Train Loss: 0.6763, Train Acc: 77.49%, Val Loss: 0.6579, Val Acc: 81.34%
Epoch 9/25 - Train Loss: 0.5774, Train Acc: 80.43%, Val Loss: 0.5799, Val Acc: 82.57%
Epoch 10/25 - Train Loss: 0.5159, Train Acc: 81.70%, Val Loss: 0.5708, Val Acc: 83.50%
Epoch 11/25 - Train Loss: 0.4675, Train Acc: 83.41%, Val Loss: 0.5546, Val Acc: 84.13%
Epoch 12/25 - Train Loss: 0.4224, Train Acc: 84.06%, Val Loss: 0.5100, Val Acc: 85.09%
Epoch 13/25 - Train Loss: 0.3860, Train Acc: 85.02%, Val Loss: 0.4857, Val Acc: 86.02%
Epoch 14/25 - Train Loss: 0.3586, Train Acc: 85.86%, Val Loss: 0.4286, Val Acc: 87.55%
Epoch 15/25 - Train Loss: 0.3266, Train Acc: 86.96%, Val Loss: 0.5106, Val Acc: 81.04%
Epoch 16/25 - Train Loss: 0.3571, Train Acc: 85.93%, Val Loss: 0.4530, Val Acc: 85.99%
Epoch 17/25 - Train Loss: 0.3066, Train Acc: 87.52%, Val Loss: 0.4506, Val Acc: 86.85%
Epoch 18/25 - Train Loss: 0.2692, Train Acc: 88.30%, Val Loss: 0.4269, Val Acc: 87.88%
Epoch 19/25 - Train Loss: 0.2496, Train Acc: 89.07%, Val Loss: 0.4582, Val Acc: 87.85%
Epoch 20/25 - Train Loss: 0.2235, Train Acc: 89.83%, Val Loss: 0.4559, Val Acc: 87.82%
Epoch 21/25 - Train Loss: 0.2286, Train Acc: 89.44%, Val Loss: 0.4393, Val Acc: 88.01%
Epoch 22/25 - Train Loss: 0.2132, Train Acc: 90.27%, Val Loss: 0.5220, Val Acc: 85.62%
Epoch 23/25 - Train Loss: 0.2206, Train Acc: 89.57%, Val Loss: 0.4959, Val Acc: 86.02%
Epoch 24/25 - Train Loss: 0.2108, Train Acc: 90.20%, Val Loss: 0.4699, Val Acc: 87.72%
Early stopping triggered at epoch 24
Saving the best model with validation loss: 0.4269

Evaluating the model:
Test Results - Loss: 0.4624, Accuracy: 84.64%, Precision: 0.80, Recall: 0.79, F1 Score: 0.79
```

- Sử dụng kết hợp cả Global Average Pooling và Gobal Max Pooling:

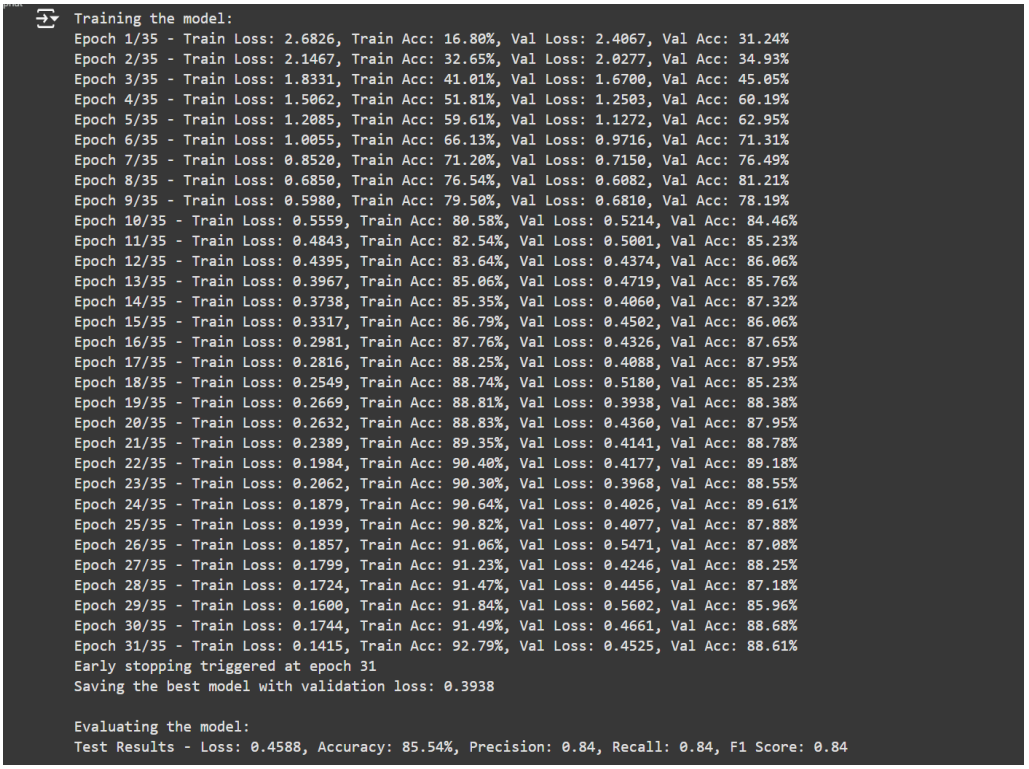


=> Global Max Pooling giảm về các chỉ số đánh giá nhưng lại hơn về accuracy, với kết hợp 2 pooling, các chỉ số đánh giá có tính đồng đều hơn trong khi (chỉ loss tăng nhưng khá nhỏ) và accuracy có cải thiện nhỏ.

- **Các điều chỉnh mô hình với embed_model 300:**
  - **Thay đổi embed_model:**



  - **Tăng độ dài văn bản tối đa lên 300 từ:**



=> Khi sử dụng embed_model 300 với cài đặt mặc định mô hình có hiệu suất giảm so với mô hình embed_model 100 nhưng khi tăng số lượng từ lên 300 mô hình cho thấy độ chính xác được cải thiện đáng kể ở các chỉ số như Precision, Recall, Scorce và có cải thiện nhỏ Accuracy. Mô hình đã được thử nghiệm với các độ dài tối đa văn bản khác nhau và 300 từ là tối đa văn bản tối ưu.

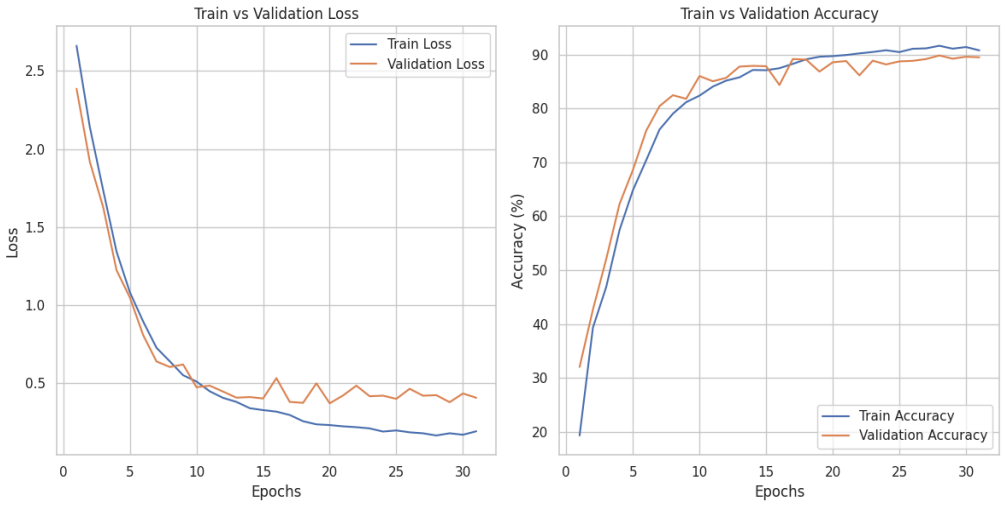
- **Các thử nghiệm điều chỉnh batch_size:**
    - Với các thử nghiệm tăng giảm batch_size, mô hình hoạt động tốt nhất với thiết lập mặc định là 128, với các thiết lập khác như: 64 và 32 mô hình bị giảm hiệu suất nhẹ và tốc độ huấn luyện tăng đáng kể.
- **Sử dụng thư viện underthesea:**
    - Mặc dù là thư viện chuyên sử dụng cho xử lý Tiếng Việt và có những ưu điểm rất tiềm năng nhưng khi sử dụng thư viện để tokenize văn bản. Hiệu suất mô hình bị giảm rất mạnh.

```
Training the model:
Epoch 1/35 - Train Loss: 2.7187, Train Acc: 15.02%, Val Loss: 2.6798, Val Acc: 11.75%
Epoch 2/35 - Train Loss: 2.4445, Train Acc: 23.90%, Val Loss: 2.0878, Val Acc: 33.07%
Epoch 3/35 - Train Loss: 1.8979, Train Acc: 41.71%, Val Loss: 1.7325, Val Acc: 47.08%
Epoch 4/35 - Train Loss: 1.6730, Train Acc: 48.52%, Val Loss: 1.5790, Val Acc: 50.60%
Epoch 5/35 - Train Loss: 1.4604, Train Acc: 53.89%, Val Loss: 1.3163, Val Acc: 59.79%
Epoch 6/35 - Train Loss: 1.2677, Train Acc: 58.74%, Val Loss: 1.4122, Val Acc: 56.77%
Epoch 7/35 - Train Loss: 1.1031, Train Acc: 64.32%, Val Loss: 1.0267, Val Acc: 68.23%
Epoch 8/35 - Train Loss: 0.9490, Train Acc: 68.68%, Val Loss: 0.9641, Val Acc: 71.35%
Epoch 9/35 - Train Loss: 0.8523, Train Acc: 71.80%, Val Loss: 0.9346, Val Acc: 71.78%
Epoch 10/35 - Train Loss: 0.8055, Train Acc: 73.01%, Val Loss: 0.7840, Val Acc: 76.83%
Epoch 11/35 - Train Loss: 0.7374, Train Acc: 75.45%, Val Loss: 0.8803, Val Acc: 73.11%
Epoch 12/35 - Train Loss: 0.7369, Train Acc: 75.45%, Val Loss: 0.7553, Val Acc: 77.46%
Epoch 13/35 - Train Loss: 0.6334, Train Acc: 78.34%, Val Loss: 0.8306, Val Acc: 76.00%
Epoch 14/35 - Train Loss: 0.6148, Train Acc: 78.74%, Val Loss: 0.6785, Val Acc: 79.08%
Epoch 15/35 - Train Loss: 0.5636, Train Acc: 80.38%, Val Loss: 0.7222, Val Acc: 79.25%
Epoch 16/35 - Train Loss: 0.5227, Train Acc: 81.41%, Val Loss: 0.7096, Val Acc: 79.78%
Epoch 17/35 - Train Loss: 0.5069, Train Acc: 81.64%, Val Loss: 0.6906, Val Acc: 79.88%
Epoch 18/35 - Train Loss: 0.4921, Train Acc: 82.80%, Val Loss: 0.6826, Val Acc: 80.81%
Epoch 19/35 - Train Loss: 0.4545, Train Acc: 83.76%, Val Loss: 0.7040, Val Acc: 80.08%
Epoch 20/35 - Train Loss: 0.4102, Train Acc: 84.58%, Val Loss: 0.6744, Val Acc: 82.04%
Epoch 21/35 - Train Loss: 0.3773, Train Acc: 85.80%, Val Loss: 0.6458, Val Acc: 81.31%
Epoch 22/35 - Train Loss: 0.3663, Train Acc: 85.91%, Val Loss: 0.6592, Val Acc: 81.54%
Epoch 23/35 - Train Loss: 0.3454, Train Acc: 86.53%, Val Loss: 0.6715, Val Acc: 82.01%
Epoch 24/35 - Train Loss: 0.3131, Train Acc: 87.81%, Val Loss: 0.6351, Val Acc: 82.30%
Epoch 25/35 - Train Loss: 0.3044, Train Acc: 88.13%, Val Loss: 0.6530, Val Acc: 83.47%
Epoch 26/35 - Train Loss: 0.2904, Train Acc: 88.22%, Val Loss: 0.7251, Val Acc: 82.04%
Epoch 27/35 - Train Loss: 0.2832, Train Acc: 88.40%, Val Loss: 0.7128, Val Acc: 81.34%
Epoch 28/35 - Train Loss: 0.3147, Train Acc: 87.42%, Val Loss: 0.6554, Val Acc: 83.10%
Epoch 29/35 - Train Loss: 0.2707, Train Acc: 88.95%, Val Loss: 0.7008, Val Acc: 83.10%
Epoch 30/35 - Train Loss: 0.2446, Train Acc: 89.76%, Val Loss: 0.7014, Val Acc: 83.43%
Epoch 31/35 - Train Loss: 0.2256, Train Acc: 90.26%, Val Loss: 0.6755, Val Acc: 83.13%
Epoch 32/35 - Train Loss: 0.2100, Train Acc: 90.97%, Val Loss: 0.8026, Val Acc: 81.37%
Epoch 33/35 - Train Loss: 0.2407, Train Acc: 89.62%, Val Loss: 0.6796, Val Acc: 82.93%
Epoch 34/35 - Train Loss: 0.2261, Train Acc: 90.54%, Val Loss: 0.8415, Val Acc: 82.01%
Early stopping triggered at epoch 34
Saving the best model with validation loss: 0.6351

Evaluating the model:
Test Results - Loss: 0.8097, Accuracy: 78.98%, Precision: 0.75, Recall: 0.71, F1 Score: 0.72
```

- **Đánh giá cuối cùng:**
    - Mô hình đạt hiệu suất cao nhất với thiết lập mặc định khi sử dụng embed_model 300. Tuy Test Loss và Accuracy chỉ thay đổi nhỏ so với khi sử dụng embed_model 100 nhưng độ tổng quát của mô hình với giữ liệu mới tăng đáng kể, thể hiện sự ổn định khi dự đoán các nhãn dễ gây nhầm lẫn. Các lần huấn luyện có thể chênh lệch nhau không quá 0.02 loss.
    - **Training log:**

```
Training the model:
Epoch 1/35 - Train Loss: 2.6616, Train Acc: 19.32%, Val Loss: 2.3864, Val Acc: 32.04%
Epoch 2/35 - Train Loss: 2.1392, Train Acc: 39.33%, Val Loss: 1.9163, Val Acc: 42.66%
Epoch 3/35 - Train Loss: 1.7349, Train Acc: 46.88%, Val Loss: 1.6269, Val Acc: 52.06%
Epoch 4/35 - Train Loss: 1.3429, Train Acc: 57.45%, Val Loss: 1.2234, Val Acc: 62.28%
Epoch 5/35 - Train Loss: 1.0821, Train Acc: 64.80%, Val Loss: 1.0483, Val Acc: 68.56%
Epoch 6/35 - Train Loss: 0.8943, Train Acc: 70.38%, Val Loss: 0.8072, Val Acc: 75.86%
Epoch 7/35 - Train Loss: 0.7260, Train Acc: 76.08%, Val Loss: 0.6383, Val Acc: 80.41%
Epoch 8/35 - Train Loss: 0.6391, Train Acc: 79.00%, Val Loss: 0.6029, Val Acc: 82.44%
Epoch 9/35 - Train Loss: 0.5498, Train Acc: 81.15%, Val Loss: 0.6188, Val Acc: 81.77%
Epoch 10/35 - Train Loss: 0.5097, Train Acc: 82.36%, Val Loss: 0.4731, Val Acc: 85.99%
Epoch 11/35 - Train Loss: 0.4469, Train Acc: 84.04%, Val Loss: 0.4832, Val Acc: 85.03%
Epoch 12/35 - Train Loss: 0.4048, Train Acc: 85.14%, Val Loss: 0.4451, Val Acc: 85.66%
Epoch 13/35 - Train Loss: 0.3791, Train Acc: 85.76%, Val Loss: 0.4063, Val Acc: 87.75%
Epoch 14/35 - Train Loss: 0.3388, Train Acc: 87.12%, Val Loss: 0.4103, Val Acc: 87.88%
Epoch 15/35 - Train Loss: 0.3265, Train Acc: 87.08%, Val Loss: 0.4012, Val Acc: 87.82%
Epoch 16/35 - Train Loss: 0.3168, Train Acc: 87.45%, Val Loss: 0.5315, Val Acc: 84.33%
Epoch 17/35 - Train Loss: 0.2953, Train Acc: 88.25%, Val Loss: 0.3791, Val Acc: 89.14%
Epoch 18/35 - Train Loss: 0.2553, Train Acc: 89.09%, Val Loss: 0.3733, Val Acc: 89.04%
Epoch 19/35 - Train Loss: 0.2353, Train Acc: 89.57%, Val Loss: 0.4979, Val Acc: 86.82%
Epoch 20/35 - Train Loss: 0.2307, Train Acc: 89.69%, Val Loss: 0.3702, Val Acc: 88.55%
Epoch 21/35 - Train Loss: 0.2226, Train Acc: 89.90%, Val Loss: 0.4202, Val Acc: 88.78%
Epoch 22/35 - Train Loss: 0.2172, Train Acc: 90.20%, Val Loss: 0.4832, Val Acc: 86.12%
Epoch 23/35 - Train Loss: 0.2092, Train Acc: 90.45%, Val Loss: 0.4154, Val Acc: 88.84%
Epoch 24/35 - Train Loss: 0.1890, Train Acc: 90.78%, Val Loss: 0.4190, Val Acc: 88.15%
Epoch 25/35 - Train Loss: 0.1963, Train Acc: 90.44%, Val Loss: 0.3987, Val Acc: 88.71%
Epoch 26/35 - Train Loss: 0.1841, Train Acc: 91.05%, Val Loss: 0.4632, Val Acc: 88.81%
Epoch 27/35 - Train Loss: 0.1779, Train Acc: 91.13%, Val Loss: 0.4190, Val Acc: 89.14%
Epoch 28/35 - Train Loss: 0.1639, Train Acc: 91.61%, Val Loss: 0.4222, Val Acc: 89.81%
Epoch 29/35 - Train Loss: 0.1780, Train Acc: 91.07%, Val Loss: 0.3774, Val Acc: 89.21%
Epoch 30/35 - Train Loss: 0.1687, Train Acc: 91.38%, Val Loss: 0.4331, Val Acc: 89.58%
Epoch 31/35 - Train Loss: 0.1908, Train Acc: 90.75%, Val Loss: 0.4057, Val Acc: 89.48%
Early stopping triggered at epoch 31
Saving the best model with validation loss: 0.3702

Evaluating the model:
Test Results - Loss: 0.4319, Accuracy: 86.27%, Precision: 0.85, Recall: 0.83, F1 Score: 0.84
```

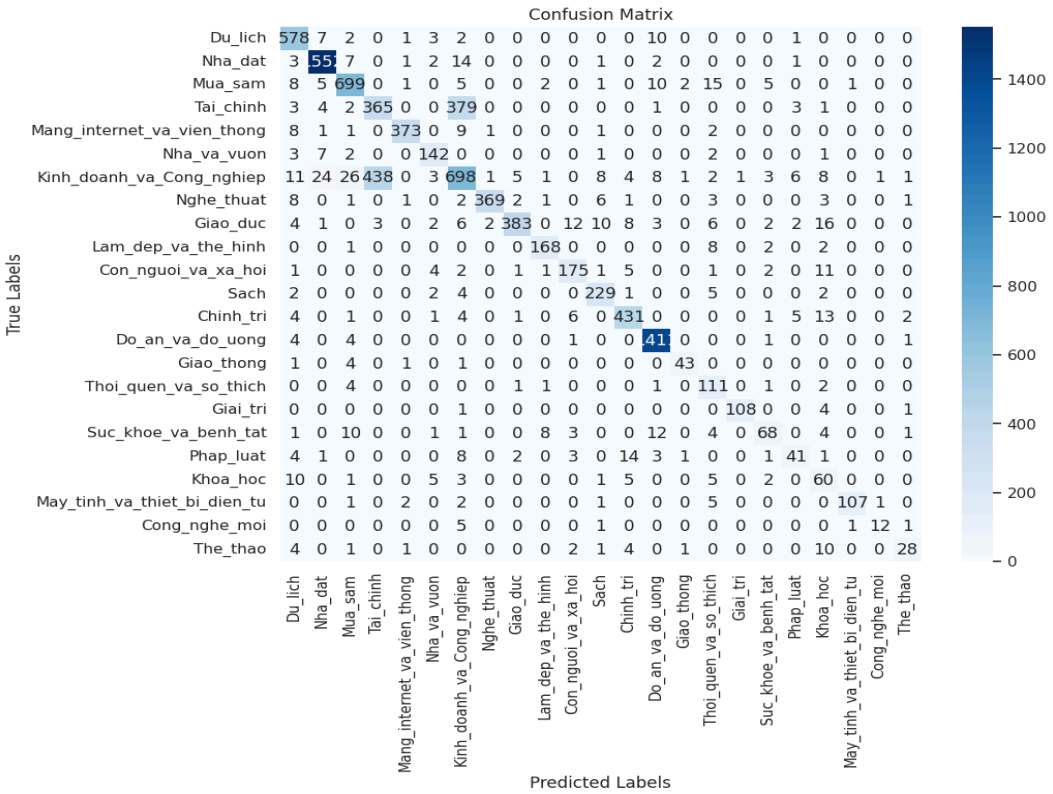
- **Biểu đồ Train Loss vs Validation Loss và Train Accuracy vs Validation theo Epoch:**



- **Confusion matrix:**

Classification Report:

	precision	recall	f1-score	support
Du_lich	0.88	0.96	0.92	604
Nha_dat	0.97	0.98	0.97	1583
Mua_sam	0.91	0.93	0.92	754
Tai_chinh	0.45	0.48	0.47	758
Mang_internet_va_vien_thong	0.98	0.94	0.96	396
Nha_va_vuon	0.86	0.90	0.88	158
Kinh_doanh_va_Cong_nghiep	0.61	0.56	0.58	1250
Nghe_thuat	0.99	0.93	0.96	398
Giao_duc	0.97	0.83	0.90	460
Lam_dep_va_the_hinh	0.92	0.93	0.93	181
Con_nguoi_va_xa_hoi	0.87	0.86	0.86	204
Sach	0.87	0.93	0.90	245
Chinh_tri	0.91	0.92	0.92	469
Do_an_va_do_uong	0.97	0.99	0.98	1422
Giao_thong	0.90	0.86	0.88	50
Thoi_quen_va_so_thich	0.66	0.92	0.77	121
Giai_tri	0.99	0.95	0.97	114
Suc_khoe_va_benh_tat	0.77	0.60	0.68	113
Phap_luat	0.69	0.52	0.59	79
Khoa_hoc	0.43	0.65	0.52	92
May_tinh_va_thiet_bi_dien_tu	0.98	0.90	0.94	119
Cong_nghe_moi	0.86	0.60	0.71	20
The_thao	0.78	0.54	0.64	52
accuracy			0.85	9642
macro avg	0.84	0.81	0.82	9642
weighted avg	0.85	0.85	0.84	9642



- **Thử nghiệm mô hình:**

- Nhãn Tài chính có các chỉ số đánh giá thấp nhất nên sẽ được thử nghiệm với 3 trường hợp bài viết có độ dài: 250 từ, 300 từ và 350 từ.

```
predict(embed_lookup, best_model, financial_post250, seq_length)

Predicted class: 3 (label: Tai_chinh)
Predicted class probability: 0.545604
(3,
 tensor([[1.1113e-05, 9.8824e-04, 2.8423e-04, 5.4560e-01, 1.0468e-05, 2.6935e-04,
          4.5211e-01, 6.7885e-07, 2.5530e-04, 6.3095e-07, 2.4356e-06, 2.3782e-05,
          4.2852e-05, 4.2040e-05, 8.6453e-08, 9.6457e-07, 9.3977e-08, 1.3442e-05,
          2.4567e-04, 4.1732e-06, 8.2022e-05, 3.3632e-06, 7.8586e-08]],
        device='cuda:0'))
```

```
predict(embed_lookup, best_model, financial_post300, seq_length)

Predicted class: 3 (label: Tai_chinh)
Predicted class probability: 0.526299
(3,
 tensor([[6.0597e-07, 4.0199e-05, 1.7488e-05, 5.2630e-01, 5.9943e-06, 7.0812e-06,
          4.7339e-01, 2.8122e-07, 6.5137e-05, 2.1352e-08, 2.7772e-07, 3.3656e-06,
          2.0005e-05, 5.8237e-07, 7.8834e-09, 2.5392e-08, 4.2711e-08, 2.7309e-07,
          9.1715e-05, 2.8646e-07, 4.9363e-05, 6.7785e-06, 3.5478e-08]],
        device='cuda:0'))
```

```
predict(embed_lookup, best_model, financial_post350, seq_length)

Predicted class: 6 (label: Kinh_doanh_va_Cong_nghiep)
Predicted class probability: 0.545713
(6,
 tensor([[7.5403e-05, 3.5755e-03, 1.7968e-03, 4.4508e-01, 8.8976e-05, 1.5892e-03,
          5.4571e-01, 7.3767e-06, 5.5639e-04, 6.3352e-06, 2.6020e-05, 1.2970e-04,
          2.1118e-04, 1.3066e-04, 3.4931e-06, 1.4399e-05, 2.0831e-06, 9.1920e-05,
          4.4061e-04, 4.0504e-05, 3.8575e-04, 3.6403e-05, 2.6086e-06]],
        device='cuda:0'))
```

=> Có thể thấy với độ dài bài việt nhỏ hơn hoặc bằng số số từ tối đa mô hình vẫn có thể dự đoán chính xác nhưng khi độ dài bài viết lớn hơn, mô hình có sự nhầm lẫn sang nhãn "Kinh doanh và công nghệ". Xác suất của nhãn xấp xỉ 50% cho thấy mô hình đang có sự nhập nhằng.

- Với các nhãn có chỉ số cao mô hình cho kết quả dự đoán với xác xuất gần như tuyệt đối với các 3 trường hợp độ dài văn bản:

```
predict(embed_lookup, best_model, education_post250, seq_length)

Predicted class: 8 (label: Giao_duc)
Predicted class probability: 0.999994
(8,
 tensor([[2.5847e-11, 1.0716e-17, 3.7138e-15, 1.0298e-08, 2.5122e-17, 7.7237e-12,
          3.8681e-08, 2.0170e-07, 9.9999e-01, 4.9373e-10, 9.4564e-08, 4.2720e-07,
          2.1129e-06, 1.7004e-10, 1.5666e-22, 8.2863e-12, 1.8965e-09, 2.0863e-10,
          1.0615e-08, 2.7427e-06, 6.0587e-14, 7.4693e-14, 2.8765e-12]],
        device='cuda:0'))
```

```
predict(embed_lookup, best_model, food_post_300, seq_length)

Predicted class: 13 (label: Do_an_va_do_uong)
Predicted class probability: 0.997475
(13,
 tensor([[1.3362e-03, 3.0095e-07, 8.2549e-04, 2.3530e-07, 8.7773e-12, 2.2697e-06,
          2.7279e-05, 2.1372e-11, 8.9725e-07, 1.4830e-07, 2.0490e-06, 9.3365e-06,
          2.1175e-11, 9.9747e-01, 8.2872e-13, 1.0489e-05, 4.8028e-15, 3.0876e-04,
          1.2001e-08, 1.5999e-06, 8.0787e-13, 6.0135e-16, 5.1489e-12]],
        device='cuda:0'))
```

```
predict(embed_lookup, best_model, entertainment_post350, seq_length)

Predicted class: 1 (label: Nha_dat)
Predicted class probability: 0.838587
(1,
 tensor([[1.6252e-05, 8.3859e-01, 7.3611e-03, 1.5180e-02, 1.1218e-05, 1.1302e-04,
          1.3871e-01, 6.6639e-12, 3.1038e-09, 8.7799e-12, 1.3754e-11, 1.0312e-10,
          1.5818e-10, 4.4891e-07, 5.3471e-06, 5.5391e-10, 1.4483e-15, 3.0814e-10,
          1.1420e-05, 3.3821e-11, 3.1998e-09, 9.7002e-11, 4.0968e-13]],
        device='cuda:0'))
```

3. BERT:

```
test_loss, test_acc = evaluate(loaded_model, test_loader, loss_fn)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc:.4f}")
```

```
Test Loss: 0.5284, Test Accuracy: 0.8218
```

	precision	recall	f1-score	support
Chinh_tri	0.83	0.94	0.88	472
Con_nguoi_va_xa_hoi	0.69	0.88	0.77	205
Cong_nghe_moi	1.00	0.35	0.52	20
Do_an_va_do_uong	0.96	0.98	0.97	1425
Du_lich	0.85	0.90	0.87	606
Giai_tri	0.96	0.99	0.97	114
Giao_duc	0.85	0.94	0.89	460
Giao_thong	0.80	0.94	0.86	50
Khoa_hoc	0.75	0.16	0.27	92
Kinh_doanh_va_Cong_nghiep	0.57	0.83	0.67	1407
Lam_dep_va_the_hinh	0.91	0.89	0.90	181
Mang_internet_va_vien_thong	0.96	0.95	0.95	401
May_tinh_va_thiet_bi_dien_tu	0.93	0.95	0.94	119
Mua_sam	0.89	0.90	0.90	762
Nghe_thuat	0.97	0.93	0.95	398
Nha_dat	0.95	0.96	0.95	1624
Nha_va_vuon	0.76	0.71	0.73	158
Phap_luat	0.33	0.01	0.02	80
Sach	0.91	0.92	0.91	249
Suc_khoe_va_benh_tat	0.65	0.34	0.45	114
Tai_chinh	0.49	0.16	0.24	906
The_thao	0.88	0.67	0.76	52
Thoi_quen_va_so_thich	1.00	0.40	0.57	122
accuracy			0.82	10017
macro avg	0.82	0.73	0.74	10017
weighted avg	0.81	0.82	0.80	10017

- Từ kết quả ta có thể thấy một số nhãn với điểm precision khá cao nhưng recall và F1-score lại thấp, điều này là do dữ liệu về các nhãn khá ít khiến model gặp khó khăn trong việc phân loại

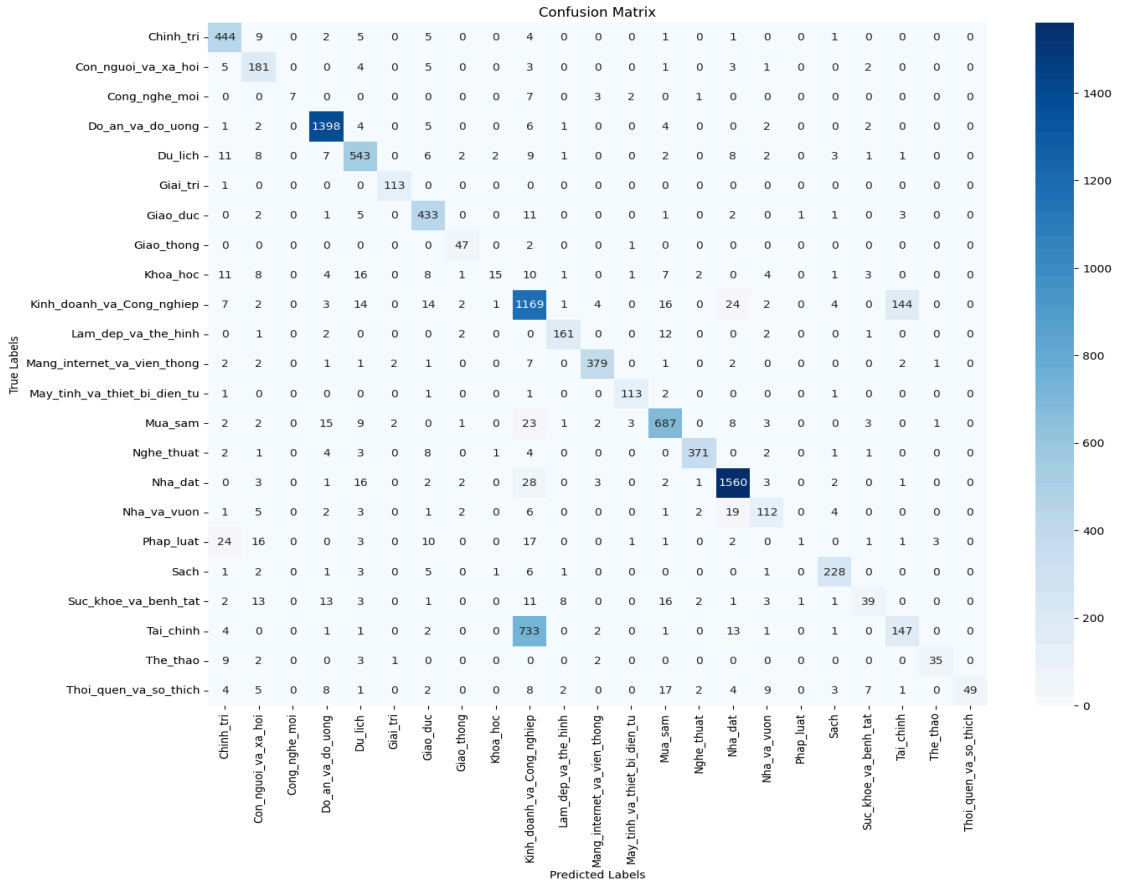
Để xử lí vấn đề này, chúng ta cần lượng dữ liệu lớn hơn để train mô hình, ở đây chúng em sử dụng pre_train của model VisoBERT từ HuggingFace. Và ở dưới là kết quả khi sử dung VisoBERT:

	precision	recall	f1-score	support
0	0.97	0.96	0.96	606
1	0.96	0.99	0.98	1624
2	0.98	0.93	0.96	762
3	0.50	0.67	0.57	763
4	0.99	0.98	0.98	401
5	0.98	0.89	0.93	158
6	0.67	0.56	0.61	1254
7	0.99	0.98	0.99	398
8	0.96	0.95	0.95	460
9	0.94	0.99	0.96	181
10	0.96	0.95	0.95	205
11	0.99	0.92	0.95	249
12	0.95	0.96	0.96	472
13	0.98	0.99	0.98	1425
14	0.98	0.96	0.97	50
15	0.98	0.91	0.94	122
16	0.99	1.00	1.00	114
17	0.80	0.79	0.80	114
18	0.90	0.78	0.83	80
19	0.97	0.75	0.85	92
20	0.98	0.97	0.98	119
21	0.95	0.90	0.92	20
22	1.00	0.96	0.98	52
accuracy			0.89	9721
macro avg	0.93	0.90	0.91	9721
weighted avg	0.89	0.89	0.89	9721

```
'Du_lich': 0,
'Nha_dat': 1,
'Mua_sam': 2,
'Tai_chinh': 3,
'Mang_internet_va_vien_thong': 4,
'Nha_va_vuon': 5,
'Kinh_doanh_va_Cong_nghiep': 6,
'Nghe_thuat': 7,
'Giao_duc': 8,
'Lam_dep_va_the_hinh': 9,
'Con_nguoi_va_xa_hoi': 10,
'Sach': 11,
'Chinh_tri': 12,
'Do_an_va_do_uong': 13,
'Giao_thong': 14,
'Thoi_quen_va_so_thich': 15,
'Giai_tri': 16,
'Suc_khoe_va_benh_tat': 17,
'Phap_luat': 18,
'Khoa_hoc': 19,
'May_tinh_va_thiet_bi_dien_tu': 20,
'Cong_nghe_moi': 21,
'The_thao': 22}
```

- Ta có thể thấy chỉ số của các nhãn ít dữ liệu đã tăng lên rất nhiều còn 2 nhãn Tai_chinh(3) và Kinh_doanh_va_cong_nghiep(6) dù đã cải thiện nhưng vẫn thấp hẳn so với các nhãn khác, nguyên nhân dẫn đến điều này như đã nói ở trên, đó là do có tới 4441 từ ngữ bị trùng lặp giữa 2 nhãn, chiếm tới hơn một nửa dữ liệu của lớp Tai_chinh và hơn ⅓ dữ liệu của lớp Kinh_doanh_va_cong_nghiep và 2 lĩnh vực này trong thực tế cũng có liên quan mật thiết với nhau

Ma trận nhầm lẫn:



4. So sánh các mô hình:

- Tổng quan: Ba mô hình được sử dụng trong bài toán gồm LSTM (baseline), BiLSTM+CNN, và BERT. Mỗi mô hình được thiết kế và triển khai với cấu trúc riêng biệt để giải quyết bài toán phân loại văn bản tiếng Việt trên các bài đăng Facebook.

So sánh hiệu suất:

- **LSTM (Baseline):**
  - Đóng vai trò làm mô hình tham chiếu ban đầu.
  - **Ưu điểm:** Có khả năng ghi nhớ mối quan hệ dài trong chuỗi văn bản, đạt hiệu quả tốt với cấu trúc đơn giản.
  - **Nhược điểm:** Thời gian huấn luyện lâu, tài nguyên yêu cầu cao.
  - **Hiệu quả:** Độ chính xác thấp nhất trong ba mô hình. Là mô hình tốt để làm baseline cho bài toán
- **BiLSTM+CNN:**
  - **Cải tiến:** Kết hợp sức mạnh của BiLSTM (học ngữ cảnh hai chiều) và Conv1D (phát hiện mẫu cục bộ) để tăng cường khả năng xử lý ngữ nghĩa và ngữ cảnh.

    - **Ưu điểm:** Hiệu suất vượt trội so với LSTM, với Test Loss giảm **2.3517** và Test Accuracy tăng **8.28%**. Mô hình tổng quát hóa tốt hơn, ổn định và xử lý hiệu quả các nhãn khó phân loại.
    - **Nhược điểm:** Cấu trúc phức tạp hơn và yêu cầu tinh chỉnh nhiều tham số để đạt hiệu suất tốt nhất. Khi văn bản dài mô hình cho xu hướng giảm hiệu suất đáng kể do đặc thù của mạng **LSTM**.
- **BERT:**
    - **Cải tiến:** Áp dụng mô hình ngôn ngữ tiên tiến BERT với khả năng xử lý ngữ cảnh sâu, phân tích ý nghĩa văn bản một cách chi tiết.
    - **Ưu điểm:** Kết quả chính xác cao, đặc biệt tốt ở các nhãn phổ biến. Sử dụng kỹ thuật học sâu hiện đại, mô hình BERT đạt độ chính xác vượt trội trên các nhãn có nhiều dữ liệu và các văn bản với ngữ cảnh dài. Khi tăng độ dài văn bản tối đa, hiệu suất vượt trội hơn hẳn so với **BiLSTM-CNN**.
    - **Nhược điểm:** Nhạy cảm với sự mất cân bằng dữ liệu, hiệu suất giảm ở các nhãn ít dữ liệu và nhãn có sự trùng lặp từ khóa (ví dụ: "Tài chính" và "Kinh doanh và công nghiệp").

Ma trận nhầm lẫn:

- **LSTM:** Nhầm lẫn cao do khả năng nắm bắt ngữ cảnh còn hạn chế.
- **BiLSTM+CNN:** Tăng khả năng phân biệt giữa các nhãn, giảm nhầm lẫn ở nhãn phức tạp nhưng vẫn có độ nhập nhằng.
- **BERT:** Hiệu suất tốt nhất với các nhãn nhiều dữ liệu nhưng giảm đáng kể độ chính xác với các nhãn có từ khóa trùng lặp.

Đánh giá cuối cùng:

- Với bài toán phân loại văn bản nhiều nhãn, cả **BiLSTM+CNN** và **BERT** đều vượt trội so với **LSTM**.
- **BiLSTM+CNN** có tính ổn định tốt hơn khi xử lý dữ liệu nhiều nhãn không cân bằng. Nhưng chỉ phù hợp với các văn bản ngắn, ngữ cảnh không quá phức tạp
- **BERT** phù hợp với các bài toán đòi hỏi phân tích ngữ nghĩa phức tạp và sâu, tuy nhiên cần tài nguyên tính toán cao hơn. Hiệu quả đối với các văn bản dài, có ngữ cảnh khó và phức tạp.

V. Tổng kết

1. Kết luận

Sau khi thực nghiệm với các mô hình **LSTM**, **BiLSTM-CNN**, và **BERT**, nhóm em nhận thấy rằng mỗi mô hình đều có ưu và nhược điểm riêng, thể hiện sự khác biệt đáng kể về hiệu suất:

- **LSTM (baseline):** Là mô hình tham chiếu ban đầu, có khả năng xử lý ngữ cảnh chuỗi dài nhưng hiệu suất tổng thể thấp nhất trong ba mô hình. LSTM phù hợp để làm baseline nhờ cấu trúc đơn giản và dễ triển khai.
- **BiLSTM+CNN:** Mang lại hiệu suất vượt trội hơn LSTM nhờ khả năng học ngữ cảnh hai chiều và phát hiện các mẫu cục bộ thông qua CNN. Mô hình này đạt độ chính xác cao trên các văn bản ngắn và ổn định hơn trên tập dữ liệu mất cân bằng. Tuy nhiên, BiLSTM-CNN gặp khó khăn với văn bản dài hoặc ngữ cảnh phức tạp.
- **BERT:** Là mô hình có hiệu suất tốt nhất ở các văn bản dài và ngữ cảnh phức tạp nhờ khả năng xử lý ngữ nghĩa sâu với kiến trúc Transformer. Tuy nhiên, hiệu suất của BERT giảm ở các nhãn ít dữ liệu và các nhãn có từ khóa trùng lặp, chẳng hạn như "Tài chính" và "Kinh doanh và công nghiệp." Điều này cho thấy hạn chế của mô hình trong việc xử lý dữ liệu không cân bằng.

Khó khăn:

1. Dữ liệu không cân bằng
2. Một số nhãn có các từ khóa trùng lặp nhau
3. Sự đa dạng của tiếng Việt làm cho việc xác định ngữ cảnh khó khăn

Giải pháp:

1. Gen, crawl thêm data ở các nhóm nhãn có số lượng quá ít.
2. Hiệu chỉnh các tham số để tìm ra những giá trị tham số tốt nhất cho mô hình

2. Cải tiến trong tương lai

Về dữ liệu:

- **Tăng cường chất lượng dữ liệu:** Tăng cường thu thập dữ liệu để làm phong phú từ vựng của các nhãn hơn, đặc biệt là nhãn đang bị nhầm lẫn là nhãn 3 và nhãn 6 để có thể tối ưu được độ chính xác của mô hình.
- **Đa dạng hóa dữ liệu:** Bổ sung nhiều các văn bản với các nội dung khác nữa ngoài bộ dữ liệu ban đầu, nếu được thì là toàn bộ các nội dung có trong văn bản tiếng Việt hiện

tại để mô hình có khả năng phân loại được các văn bản trong những tình huống thực tế.

- **Dữ liệu không gán nhãn:** Áp dụng các phương pháp học bán giám sát hoặc học không giám sát để khai thác dữ liệu không gán nhãn nhằm tăng cường mô hình và giảm phụ thuộc vào dữ liệu đã được gán nhãn.

Về thuật toán:

- **Khám phá các mô hình học sâu khác:** Thực nghiệm với một số mô hình học sâu khác như GPT,… để xem xét sự khác biệt, ưu nhược điểm từ đó rút ra mô hình nào là thích hợp nhất với bài toán này.
- **Tuning và tối ưu hóa mô hình:** Thực hiện các kỹ thuật tinh chỉnh mô hình như tìm kiếm siêu tham số, điều chỉnh các tham số huấn luyện và áp dụng các phương pháp regularization để cải thiện hiệu suất của mô hình.
- **Xử lý ngữ nghĩa sâu hơn:** Nghiên cứu và áp dụng các phương pháp học sâu hơn về ngữ nghĩa văn bản như phân tích cảm xúc, phát hiện chủ đề, hoặc nhận diện thực thể để cải thiện độ chính xác và khả năng phân loại của mô hình.

3. Hướng phát triển

- **Phát triển giao diện người dùng:** Tạo ra các công cụ hoặc ứng dụng có giao diện thân thiện để người dùng có thể dễ dàng tương tác với hệ thống phân loại văn bản. Điều này có thể bao gồm việc phát triển các ứng dụng web hoặc di động để sử dụng mô hình phân loại trong các tình huống thực tế.
- **Tích hợp vào hệ thống hiện tại:** Xem xét tích hợp mô hình phân loại vào các hệ thống hiện tại hoặc quy trình công việc của tổ chức. Ví dụ, mô hình có thể được tích hợp vào hệ thống quản lý tài liệu hoặc công cụ tìm kiếm để tự động phân loại và tổ chức tài liệu.