

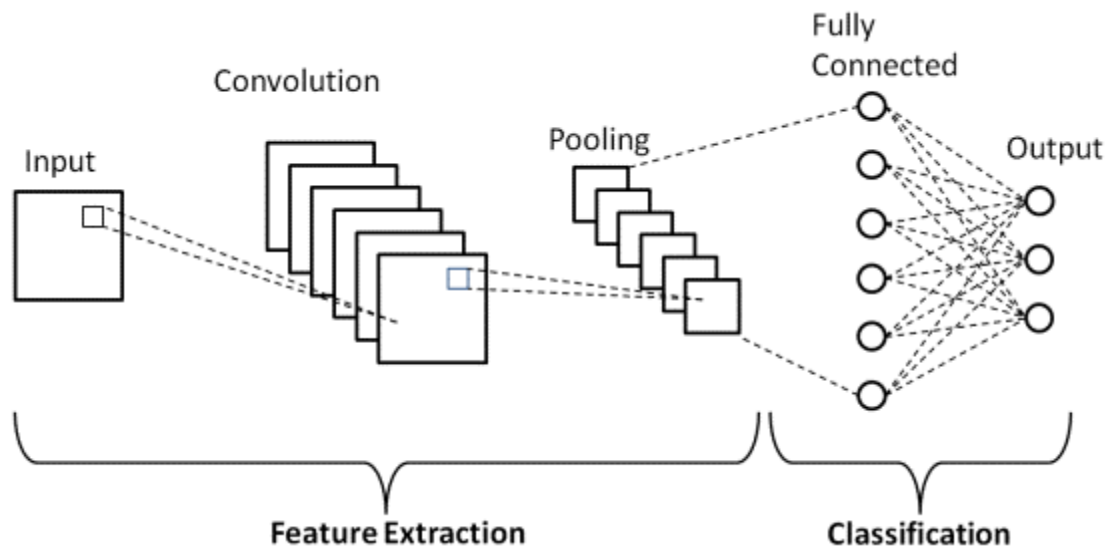
Họ và tên : Trịnh Vinh Tuấn Đạt

Mã sinh viên : B21DCCN031

1. Trình bày sự hiểu biết về CNN, RNN, LSTM

a. CNN

Bắt đầu từ những năm 1980, CNN được phát triển bởi Yann LeCun để nhận diện ký tự viết tay trong hệ thống đọc zip-code. Kiến trúc của CNN giúp xử lý tốt các dữ liệu hình ảnh, và đã được cải tiến thành nhiều loại CNN phức tạp như AlexNet (2012), VGGNet (2014) và ResNet (2015), đạt được thành công lớn trong các bài toán nhận diện hình ảnh.



Mạng CNN là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và tanh để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

Các layer trong mô hình CNN liên kết được với nhau thông qua cơ chế convolution. Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó. Mỗi một lớp được sử dụng các filter khác nhau thông thường có hàng trăm hàng nghìn filter như vậy và kết hợp kết quả của chúng lại. Ngoài ra có một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu). Trong quá trình huấn luyện mạng (training) CNN tự động học các giá trị qua các lớp filter dựa vào cách thức mà bạn thực hiện. Ví dụ trong tác vụ phân lớp ảnh, CNNs sẽ cố gắng tìm ra thông số tối ưu cho các filter tương ứng theo thứ tự raw pixel > edges > shapes > facial > high-level features. Layer cuối cùng được dùng để phân lớp ảnh.

Code mẫu :

```
[ ]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Khởi tạo mô hình
model_cnn = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)), # Lớp tích chập
    MaxPooling2D((2, 2)), # Lớp gộp (pooling)
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(), # Biến đổi thành vector phẳng
    Dense(128, activation='relu'), # Fully connected layer
    Dense(10, activation='softmax') # Lớp đầu ra với softmax
])

# Biên dịch mô hình
model_cnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_cnn.summary()
```

Các lớp trong mô hình CNN:

+ Conv2D

Mô tả: Đây là lớp tích chập (Convolutional Layer) 2D, áp dụng các filter (bộ lọc) lên các phần của ảnh đầu vào để trích xuất đặc trưng (feature). Conv2D là thành phần chính trong kiến trúc CNN, giúp mô hình học được các mẫu cục bộ trong hình ảnh.

Tham số chính: filters: Số lượng bộ lọc (số lượng đặc trưng cần trích xuất), trong đoạn code là 32 và 64. kernel_size: Kích thước của filter (ở đây là 3x3).

Activation='relu': Sử dụng hàm kích hoạt ReLU để loại bỏ các giá trị âm, chỉ giữ lại giá trị dương.

+ MaxPooling2D

Mô tả: Lớp gộp (Pooling Layer) 2D này được dùng để giảm kích thước của đặc trưng trích xuất, giúp giảm thiểu số lượng tham số và ngăn chặn hiện tượng quá khớp (overfitting).

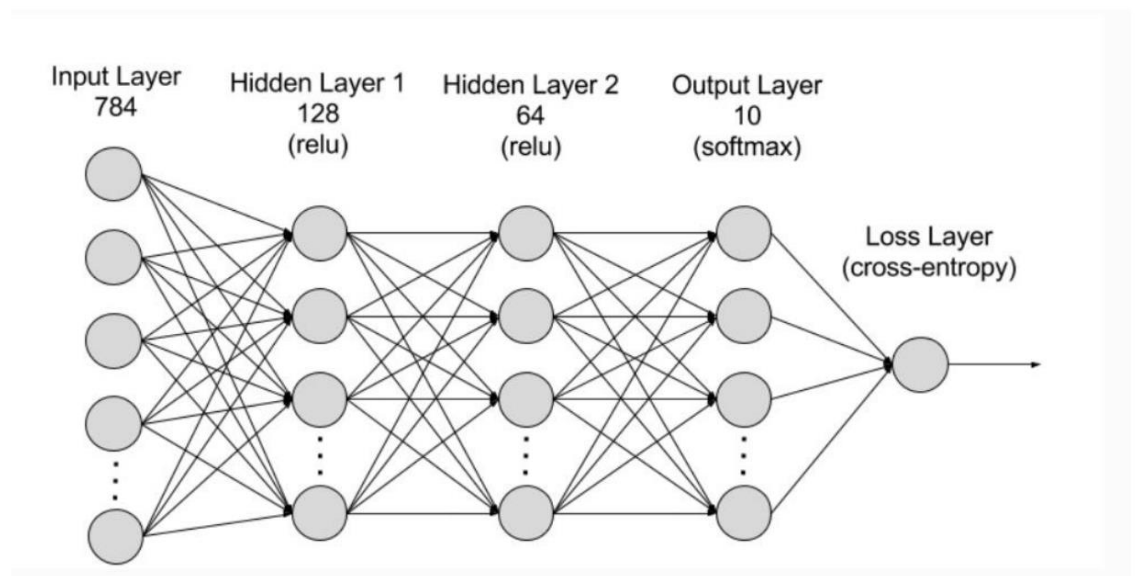
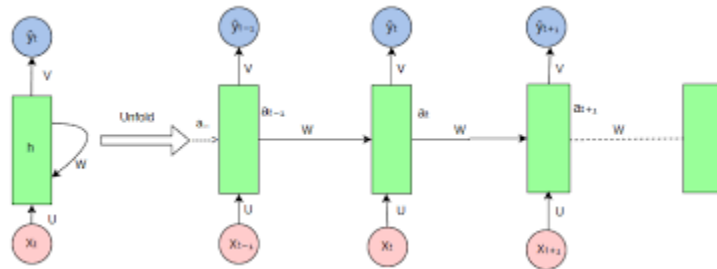
Tham số chính: pool_size=(2, 2): Kích thước của filter dùng để lấy giá trị lớn nhất trong vùng này. Kích thước (2, 2) giúp giảm dữ liệu thành một nửa chiều dài và chiều rộng.

+ Flatten: Mô tả: Lớp này làm phẳng dữ liệu từ dạng ma trận 2D (hoặc 3D) thành vector 1D. Điều này giúp chuyển đổi dữ liệu từ các lớp tích chập thành dạng đầu vào phù hợp cho các lớp fully connected (dense layers).

+ Dense: Mô tả: Đây là lớp fully connected, kết nối tất cả các neuron từ lớp trước với mỗi neuron trong lớp này. Tham số chính: units: Số lượng neuron trong lớp. Ví dụ, lớp đầu tiên có 128 neuron và lớp cuối cùng có 10 neuron cho 10 nhãn phân loại. activation: Sử dụng relu trong lớp hidden và softmax trong lớp cuối để tính xác suất cho mỗi nhãn trong bài toán phân loại.

b. RNN

RNN được phát triển từ những năm 1980 và được ứng dụng nhiều vào các năm 1990. Các ý tưởng ban đầu về RNN đã được đề xuất bởi các nhà khoa học như David Rumelhart và Ronald J. Williams. RNN được phát triển để xử lý dữ liệu tuần tự (sequence data), đặc biệt hữu ích cho các ứng dụng ngôn ngữ tự nhiên (NLP). Tuy nhiên, RNN gặp khó khăn với các chuỗi dài do hiện tượng “vanishing gradient” (độ dốc mất dần) khiến mô hình khó học tốt các phụ thuộc xa. Ứng dụng phổ biến nhất hiện nay của RNN là xử lý ngôn ngữ tự nhiên, ví dụ như dịch máy, phân loại văn bản, và phân tích cảm xúc.



RNN được tạo thành từ các nơ-ron: các nút xử lý dữ liệu kết hợp cùng nhau để thực hiện các tác vụ phức tạp. Các nơ-ron được tổ chức dưới dạng lớp đầu vào, đầu ra và ẩn. Lớp đầu vào nhận thông tin để xử lý và lớp đầu ra cung cấp kết quả. Quá trình xử lý dữ liệu, phân tích và dự đoán diễn ra trong lớp ẩn.

Lớp ẩn RNN hoạt động bằng cách lần lượt truyền dữ liệu tuần tự nhận được đến các lớp ẩn. Tuy nhiên, RNN cũng có quy trình làm việc tự lặp lại hay hồi quy: lớp ẩn có thể ghi nhớ và sử dụng các đầu vào trước đó cho các dự đoán trong tương lai trong một thành phần bộ nhớ ngắn hạn. Quy trình này sử dụng đầu vào hiện tại và bộ nhớ đã lưu trữ để dự đoán chuỗi tiếp theo.

Ví dụ: hãy xem xét chuỗi: Apple is red (Táo màu đỏ). Bạn muốn RNN dự đoán red (màu đỏ) khi nhận được chuỗi đầu vào Apple is (Táo màu). Khi xử lý từ Apple (Táo), lớp ẩn sẽ lưu trữ một bản sao trong bộ nhớ. Tiếp theo, khi thấy từ is (màu), lớp ẩn gọi lại Apple (Táo) từ bộ nhớ của mình và hiểu toàn bộ chuỗi: Apple is (Táo màu) là ngữ cảnh. Sau đó, lớp ẩn có thể dự đoán red (màu đỏ) để cải thiện độ chính xác. Do đó, RNN trở nên hữu ích trong nhận dạng giọng nói, dịch máy và các tác vụ lập mô hình ngôn ngữ khác

Code mẫu

```
[ ]: from tensorflow.keras.layers import SimpleRNN

# Khởi tạo mô hình
model_rnn = Sequential([
    SimpleRNN(50, activation='tanh', input_shape=(100, 1)), # Lớp RNN
    Dense(1) # Lớp đầu ra
])

# Biên dịch mô hình
model_rnn.compile(optimizer='adam', loss='mse')
model_rnn.summary()
```

Mô hình RNN này bao gồm ba lớp, mỗi lớp đảm nhiệm một vai trò cụ thể trong quá trình xử lý và dự đoán dữ liệu:

layers.SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], 1))

Đây là lớp mạng nơ-ron hồi quy đơn giản (Simple Recurrent Neural Network - RNN) với 50 đơn vị ẩn (neurons).

input_shape=(X_train.shape[1], 1): Cấu trúc đầu vào là (số bước thời gian, số đặc trưng đầu vào). Ở đây, đầu vào được thiết lập với một đặc trưng trên mỗi bước thời gian.

activation='relu': Chức năng kích hoạt ReLU (Rectified Linear Unit) được áp dụng tại mỗi bước thời gian. ReLU thường giúp mô hình học nhanh hơn và hạn chế vấn đề vanishing gradient. Lớp này sẽ xử lý đầu vào tuần tự, giúp mô hình nắm bắt thông tin theo trình tự thời gian và học cách giữ thông tin từ các bước trước đó.

Lớp Dense 64 đơn vị ẩn: layers.Dense(64, activation='relu') Đây là lớp kết nối đầy đặc (fully connected) với 64 neurons.

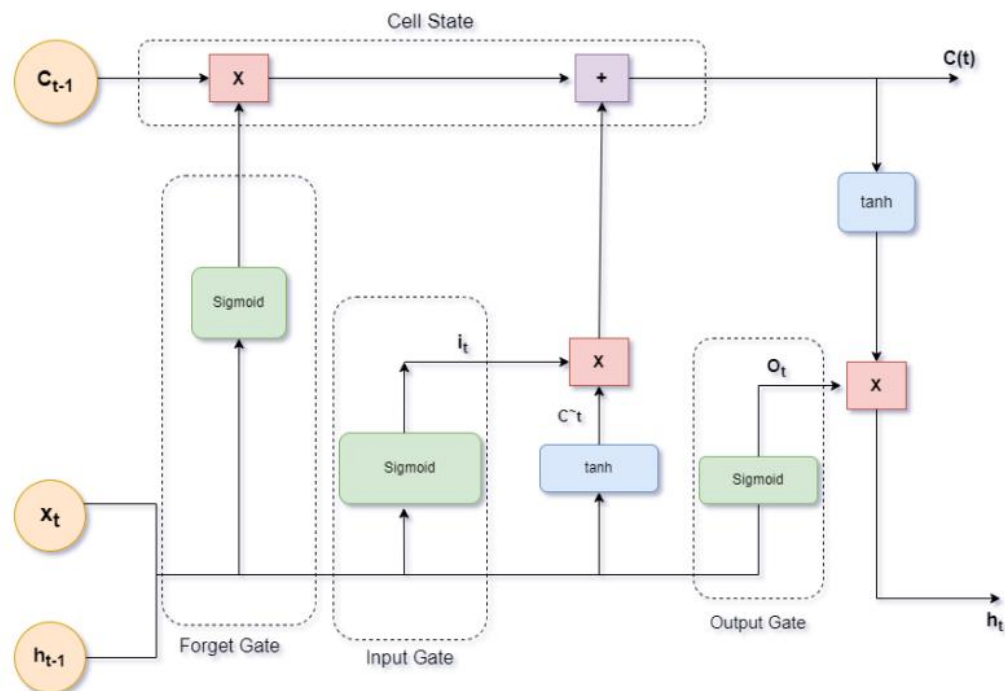
activation='relu': Kích hoạt ReLU được dùng để tạo phi tuyến cho mô hình, giúp học các mẫu phức tạp hơn trong dữ liệu. Lớp này hoạt động như một tầng ẩn thêm để tăng độ sâu của mô hình, học các đặc trưng từ đầu ra của lớp RNN phía trước.

Lớp Dense đầu ra: layers.Dense(1) Lớp cuối cùng này chứa một neuron, dùng để dự đoán một giá trị đầu ra duy nhất (giá trị hồi quy). Không sử dụng hàm kích hoạt ở lớp này vì giá trị đầu ra là một số thực liên tục, phù hợp cho các bài toán dự đoán giá trị số như hồi quy.

Mô hình này thực hiện tốt cho bài toán dự đoán chuỗi thời gian hoặc các bài toán hồi quy cần xử lý dữ liệu tuần tự và lấy thông tin từ những bước trước đó trong chuỗi.

c. LSTM

LSTM được giới thiệu vào năm 1997. Được phát triển bởi Sepp Hochreiter và Jürgen Schmidhuber, hai nhà nghiên cứu Đức nổi tiếng trong lĩnh vực học sâu. LSTM được thiết kế để khắc phục vấn đề vanishing gradient của RNN. Mô hình này sử dụng các “cổng” (input gate, forget gate, output gate) để kiểm soát cách thông tin được lưu trữ, ghi đè, và xuất ra. Nhờ vậy, LSTM có thể duy trì trạng thái ổn định và ghi nhớ thông tin quan trọng trong thời gian dài. LSTM rất phổ biến trong các bài toán yêu cầu ghi nhớ thông tin dài hạn như dịch máy, nhận diện giọng nói, và các hệ thống chatbot. Đặc biệt, LSTM đã cải thiện đáng kể độ chính xác của các mô hình xử lý ngôn ngữ tự nhiên.



Code mẫu

```
[ ]: from tensorflow.keras.layers import LSTM

# Khởi tạo mô hình
model_lstm = Sequential([
    LSTM(50, activation='tanh', input_shape=(100, 1)), # Lớp LSTM
    Dense(1) # Lớp đầu ra
])

# Biên dịch mô hình
model_lstm.compile(optimizer='adam', loss='mse')
model_lstm.summary()
```

Lớp LSTM layers.

LSTM(50, activation='relu', input_shape=(X_train.shape[1], 1)):

LSTM (Long Short-Term Memory): Đây là lớp chính trong mô hình, chuyên xử lý dữ liệu tuần tự. LSTM giúp giải quyết vấn đề vanishing gradient mà các mô hình RNN truyền thống gặp phải, cho phép mô hình học từ các mối quan hệ dài hạn trong dữ liệu.

50: Số lượng đơn vị ẩn (neurons) trong lớp LSTM. Điều này quyết định kích thước của trạng thái ẩn và số lượng thông tin mà mô hình có thể lưu trữ và xử lý.

activation='relu': Hàm kích hoạt ReLU (Rectified Linear Unit) được sử dụng tại mỗi đơn vị. Hàm ReLU giúp tăng tốc độ huấn luyện và cung cấp tính phi tuyến cho mô hình.

input_shape=(X_train.shape[1], 1):

Định nghĩa kích thước đầu vào của mô hình. X_train.shape[1] đại diện cho số bước thời gian (timesteps) trong mỗi chuỗi dữ liệu. 1 là số đặc trưng đầu vào tại mỗi bước thời gian (có thể có nhiều hơn nếu dữ liệu có nhiều đặc trưng).

Lớp Dense (kết nối dày đặc)

layers.Dense(64, activation='relu'): Đây là một lớp kết nối dày đặc với 64 neurons. Lớp này được sử dụng để tăng cường khả năng học của mô hình, cho phép nó học các đặc trưng phức tạp hơn từ đầu ra của lớp LSTM.

activation='relu': Hàm kích hoạt ReLU cũng được sử dụng ở lớp này để cung cấp tính phi tuyến và giúp mô hình học nhanh hơn.

Lớp Dense đầu ra

layers.Dense(1): Đây là lớp đầu ra của mô hình, chứa một neuron. Lớp này dùng để dự đoán giá trị đầu ra duy nhất (trong trường hợp này là giá trị hồi quy). Không sử dụng hàm kích hoạt tại lớp này vì đầu ra là một số thực liên tục, phù hợp với các bài toán hồi quy.

Kết luận

Mô hình LSTM này bao gồm một lớp LSTM để xử lý dữ liệu tuần tự và học từ các mối quan hệ dài hạn, sau đó là một lớp Dense với 64 neurons để tăng cường khả năng học, và cuối cùng là một lớp Dense đầu ra với một neuron để dự đoán giá trị hồi quy. Cấu trúc này giúp mô hình có khả năng nắm bắt các thông tin quan trọng trong dữ liệu chuỗi và tạo ra dự đoán chính xác hơn

2. Xây dựng bộ data về nhà đất Hà Nội – Hà Đông. Sử dụng kỹ thuật CNN, RNN, LSTM để dự đoán nó

B1: Xây dựng bộ dữ liệu bằng cách sử dụng tool crawl data giá nhà đất Hà Đông

Sử dụng tool Apify để crawl dữ liệu

<https://apify.com/minhlucvan/batdongsan-scraper/api>

Cần tinh chỉnh một số tham số để thực hiện việc crawl dữ liệu

Batdongsan Scraper Pay for usage ▾
minhlucvan/batdongsan-scraper 6 monthly users ☆ 1 star Crafted by Minh Lục Vân Maintained by Community
scrape listing data from batdongsan.com.vn
Do you like this Actor? Give it a star! ✕
Start Create task

Input Information Runs 3 Builds 1 Integrations 0 Monitoring Issues 0 Saved tasks 0

Manual JSON >>

Start URLs (optional) ⓘ

https://batdongsan.com.vn/nha-dat-ban-ha-dong Advanced ✕

+ Add ▾ Bulk edit Text file ▾

Max Requests per Crawl (optional) ⓘ

120 + -

Total Pages (optional) ⓘ

25 + -

> Run options BUILD latest TIMEOUT 300s MEMORY 1 GB

Start Save Restore default input ▾

Với start Urls sẽ chứa đường dẫn trực tiếp đến web để có thể thực hiện crawl dữ liệu

+ Để Maxrequest và Toltal page sau đó thực hiện quá trình crawl dữ liệu

✓ Succeeded

Finished! Total 127 requests: 125 succeeded, 2 failed.

RESULTS

117

REQUESTS

127 of 170 handled

USAGE

\$0.018

STARTED

2024-10-31 09:20

DURATION

1 m 6 s

More details

Help Minh Lục Văn improve your experience with Batdongsan Scraper

We need your permission to share all past and future runs of this Actor with the developer. This will help them resolve issues and ensure better performance. You can change your consent anytime in [Settings > Privacy](#).

Yes, I agree

No, thanks

Output 117

Log

Input

Storage

Live view

Integrations 0

View full log for older lines

View full log

2024-10-31T02:20:57.830Z INFO CheerioCrawler: Processing item {"itemUrl":"https://batdongsan.com.vn/ban-nha-biet-thu-lien-ke-duong-phuc-la-phuong-kien-hung-i

2024-10-31T02:20:58.027Z INFO CheerioCrawler: Processing item {"itemUrl":"https://batdongsan.com.vn/ban-nha-rieng-duong-da-sy-phuong-kien-hung/-5-tang-xay-m

2024-10-31T02:20:58.044Z INFO CheerioCrawler: Processing item {"itemUrl":"https://batdongsan.com.vn/ban-nha-rieng-duong-van-la-phuong-phu-la/sieu-vip-ha-don

2024-10-31T02:20:58.238Z WARN CheerioCrawler: Reclaiming failed request back to the list or queue. Request blocked - received 403 status code.

2024-10-31T02:20:58.240Z {"id":"kHfXnB6FKLVLRJ","url":"https://batdongsan.com.vn/ban-nha-rieng-duong-ba-la-phuong-phu-lam-1/7-tang-thang-may-thoang-truoc-s

2024-10-31T02:20:58.242Z WARN CheerioCrawler: Reclaiming failed request back to the list or queue. Request blocked - received 403 status code.

2024-10-31T02:20:58.244Z {"id":"somWgG4nxILR5DU","url":"https://batdongsan.com.vn/ban-nha-biet-thu-lien-ke-duong-le-trong-tan-phuong-duong-noi-prj-khu-do-th

2024-10-31T02:20:58.628Z WARN CheerioCrawler: Reclaiming failed request back to the list or queue. Request blocked - received 403 status code.

2024-10-31T02:20:58.630Z {"id":"2lylKwKot0nvaKC","url":"https://batdongsan.com.vn/ban-nha-rieng-phuong-mo-lao/ban-nh-sieu-dep-6-tang-1-gac-xep-1-tum-san-vuo

2024-10-31T02:20:58.633Z WARN CheerioCrawler: Reclaiming failed request back to the list or queue. Request blocked - received 403 status code.

2024-10-31T02:20:58.635Z {"id":"rAJTBSqAmogaV0r","url":"https://batdongsan.com.vn/ban-dat-pho-xom-phuong-phu-lam-1/ban-thua-hai-mat-tien-110m2-ha-dong-vi-tr

2024-10-31T02:20:58.729Z INFO CheerioCrawler: Processing item {"itemUrl":"https://batdongsan.com.vn/ban-nha-biet-thu-lien-ke-duong-phan-ke-toai-phuong-duong

2024-10-31T02:20:58.844Z WARN CheerioCrawler: Reclaiming failed request back to the list or queue. Request blocked - received 403 status code.

Export 117 results

Go to Actor

Integrate

Delete

Resurrect

Table

JSON

Preview in new tab

#	Area	Area M2	Balcony Direction	Bedroom	Bedroom Count	Code	Direction	Expired Date	Floor Count	Frontage	Furniture	Image Urls	Lat	Legal	Long
1	88 m ²	88	undefined	undefined	undefined	40772723	Đông - Nam	06/11/2024	4 tầng	8 m	undefined	23 items	20.9792404174805	undefined	105.74
2	240 m ²	240	undefined	4 PN	4 phòng	41311000	Đông - Bắc	31/10/2024	4 tầng	12 m	Không nội thất	9 items	20.97924041748047	Số đồ/ Số hồng	105.74
3	80 m ²	80	undefined	undefined	undefined	40863029	Tây - Bắc	06/11/2024	undefined	4 m	undefined	13 items	20.9792404174805	Số đồ/ Số hồng	105.74

Export 117 results

Go to Actor

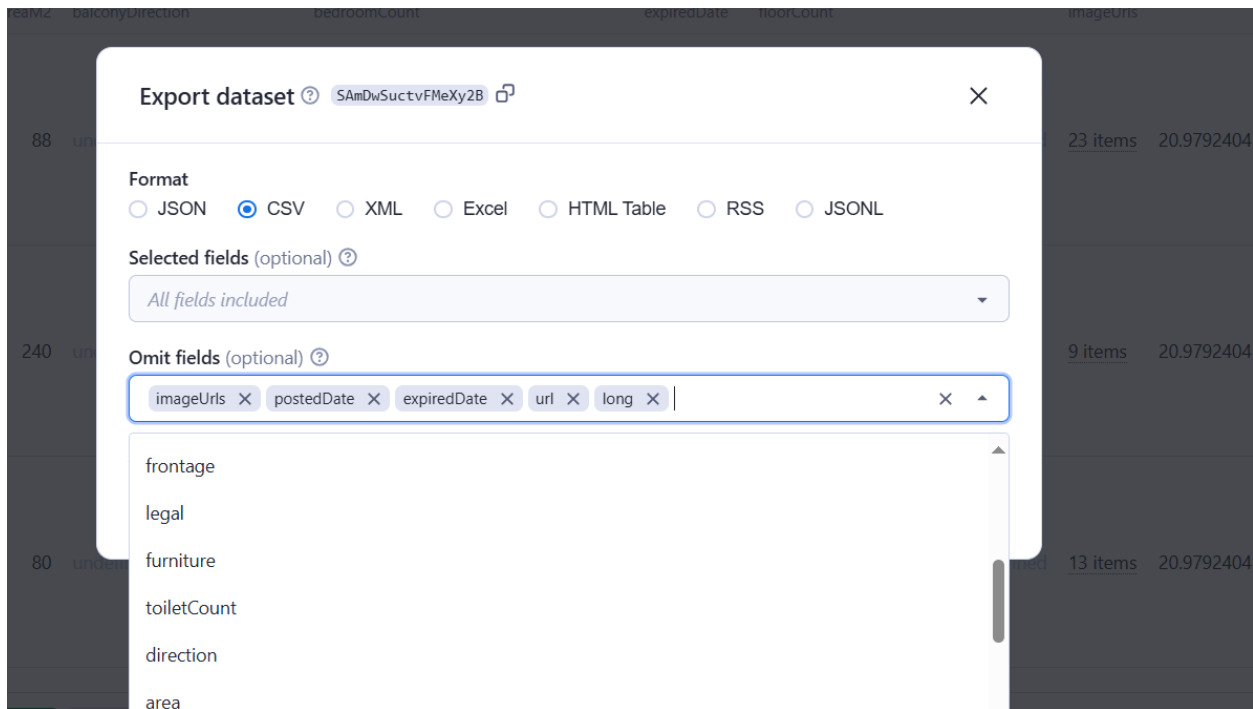
Integrate

Delete

Resurrect

Ta sẽ nhận được 117 bản ghi về dữ liệu bao gồm các trường thuộc tính như arera, AreaeM2, bedroom,

Thực hiện export để tải dữ liệu về dưới dạng file csv



Tiến hành làm sạch dữ liệu thô:

+ Hiện thị dữ liệu thô nhận được

```
[1]: import pandas as pd
df = pd.read_csv('data_house_ha_dong.csv')
df
```

[1]:	area	areaM2	balconyDirection	bedroom	bedroomCount	direction	floorCount	frontage	furniture	lat	legal	long	price	priceBil	pricePerM2
0	88 m²	88.0	NaN	NaN	NaN	Đông - Nam	4 tầng	8 m	NaN	20.979240	NaN	105.741783	14 tỷ	14.00	159.090909
1	240 m²	240.0	NaN	4 PN	4 phòng	Đông - Bắc	4 tầng	12 m	Không nội thất	20.979240	Số đồ/Số hồng	105.741783	Thỏa thuận	NaN	-0.004167
2	80 m²	80.0	NaN	NaN	NaN	Tây - Bắc	NaN	4 m	NaN	20.979240	Số đồ/Số hồng	105.741783	16,5 tỷ	16.50	206.250000
3	212,5 m²	NaN	Tây - Nam	5 PN	5 phòng	Tây - Nam	4 tầng	8,5 m	NaN	20.983617	Số đồ/Số hồng	105.772263	168 triệu/m²	NaN	NaN
4	208 m²	208.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.983466	NaN	105.772290	35,1 tỷ	35.10	168.750000
...
112	285 m²	285.0	Đông - Nam	4 PN	4 phòng	Đông - Nam	4 tầng	15 m	Đầy đủ	20.973156	Số đồ/Số hồng	105.757645	Thỏa thuận	NaN	-0.003509
113	50 m²	50.0	NaN	NaN	NaN	Đông - Nam	NaN	4 m	NaN	20.981950	Số đồ/Số hồng	105.745937	9,7 tỷ	9.70	194.000000

+ Tiến hành xác định các cột dữ liệu bị thiếu

```
[15]: def print_columns_with_na_and_dtype(dataframe):
columns_with_na = [col for col in dataframe.columns if dataframe[col].isna().any()]
if columns_with_na:
    print("Các cột còn chứa NaN và kiểu dữ liệu của chúng:")
    for col in columns_with_na:
        print(f"- Cột '{col}': kiểu dữ liệu {dataframe[col].dtype}")
else:
    print("Không có cột nào chứa NaN.")
# Gọi hàm
print_columns_with_na_and_dtype(df)
```

Các cột còn chứa NaN và kiểu dữ liệu của chúng:

- Cột 'balconyDirection': kiểu dữ liệu object
- Cột 'bedroom': kiểu dữ liệu object
- Cột 'bedroomCount': kiểu dữ liệu object
- Cột 'direction': kiểu dữ liệu object
- Cột 'floorCount': kiểu dữ liệu object
- Cột 'frontage': kiểu dữ liệu object
- Cột 'furniture': kiểu dữ liệu object
- Cột 'legal': kiểu dữ liệu object
- Cột 'priceBil': kiểu dữ liệu float64
- Cột 'pricePerM2': kiểu dữ liệu float64
- Cột 'toiletCount': kiểu dữ liệu object

+Thực hiện chuẩn hóa và điền dữ liệu còn thiếu

```
[19]: df['areaM2'] = df['areaM2'].fillna(round(df['areaM2'].mean(), 2))
df['priceBil'] = df['priceBil'].fillna(round(df['priceBil'].mean(), 2))
df['pricePerM2'] = df['pricePerM2'].fillna(round(df['pricePerM2'].mean(), 2))
df['areaM2'].isna().sum()
print_columns_with_na_and_dtype(df)
```

Các cột còn chứa NaN và kiểu dữ liệu của chúng:

- Cột 'balconyDirection': kiểu dữ liệu object
- Cột 'bedroom': kiểu dữ liệu object
- Cột 'bedroomCount': kiểu dữ liệu object
- Cột 'direction': kiểu dữ liệu object
- Cột 'floorCount': kiểu dữ liệu object
- Cột 'frontage': kiểu dữ liệu object
- Cột 'furniture': kiểu dữ liệu object
- Cột 'legal': kiểu dữ liệu object
- Cột 'toiletCount': kiểu dữ liệu object

```
[25]: for col in df.columns:
df[col] = df[col].fillna(df[col].mode()[0])
print_columns_with_na_and_dtype(df)
```

Không có cột nào chứa NaN.

```
[27]: df
```

	area	areaM2	balconyDirection	bedroom	bedroomCount	direction	floorCount	frontage	furniture	lat	legal	long	price	priceBil	pricePerM2
0	88 m²	88.00	Đông - Nam	3 PN	3 phòng	Đông - Nam	4 tầng	8 m	Đầy đủ	20.979240	Số đó/ Số hồng	105.741783	14 tỷ	14.00	159.090909
1	240 m²	240.00	Đông - Nam	4 PN	4 phòng	Đông - Bắc	4 tầng	12 m	Không nội thất	20.979240	Số đó/ Số .	105.741783	Thỏa thuận	15.79	-0.004167

+ Dữ liệu sau khi được làm sạch

[27]:

df

[27]:		area	areaM2	balconyDirection	bedroom	bedroomCount	direction	floorCount	frontage	furniture	lat	legal	long	price	priceBil	pricePerM2
	0	88 m²	88.00	Đông - Nam	3 PN	3 phòng	Đông - Nam	4 tầng	8 m	Đầy đủ	20.979240	Số đồ/Số hồng	105.741783	14 tỷ	14.00	159.090909
	1	240 m²	240.00	Đông - Nam	4 PN	4 phòng	Đông - Bắc	4 tầng	12 m	Không nội thất	20.979240	Số đồ/Số hồng	105.741783	Thỏa thuận	15.79	-0.004167
	2	80 m²	80.00	Đông - Nam	3 PN	3 phòng	Tây - Bắc	4 tầng	4 m	Đầy đủ	20.979240	Số đồ/Số hồng	105.741783	16,5 tỷ	16.50	206.250000
	3	212,5 m²	103.35	Tây - Nam	5 PN	5 phòng	Tây - Nam	4 tầng	8,5 m	Đầy đủ	20.983617	Số đồ/Số hồng	105.772263	168 triệu/m²	15.79	117.840000
	4	208 m²	208.00	Đông - Nam	3 PN	3 phòng	Đông - Nam	4 tầng	5 m	Đầy đủ	20.983466	Số đồ/Số hồng	105.772290	35,1 tỷ	35.10	168.750000

	112	285 m²	285.00	Đông - Nam	4 PN	4 phòng	Đông - Nam	4 tầng	15 m	Đầy đủ	20.973156	Số đồ/Số hồng	105.757645	Thỏa thuận	15.79	-0.003509
	113	50 m²	50.00	Đông - Nam	3 PN	3 phòng	Đông - Nam	4 tầng	4 m	Đầy đủ	20.981950	Số đồ/Số hồng	105.745937	9,7 tỷ	9.70	194.000000

+ thực hiện lọc bỏ thành phần không liên quan và chuyển sang dạng số để thực hiện huấn luyện dữ liệu

```
[152]: columns_to_drop = ['balconyDirection', 'frontage', 'lat', 'legal', 'priceBil', 'bedroomCount', 'area', 'price', 'pricePerM2', 'long', 'direction']
df = df.drop(columns=columns_to_drop)

[153]: df['floorCount'] = df['floorCount'].str.extract('(\\d+)').astype(float)
df['bedroom'] = df['bedroom'].str.extract('(\\d+)').astype(float)
df['toiletCount'] = df['toiletCount'].str.extract('(\\d+)').astype(float)
df['furniture'] = df['furniture'].map({'Đầy đủ': 1, 'Không nội thất': 0})

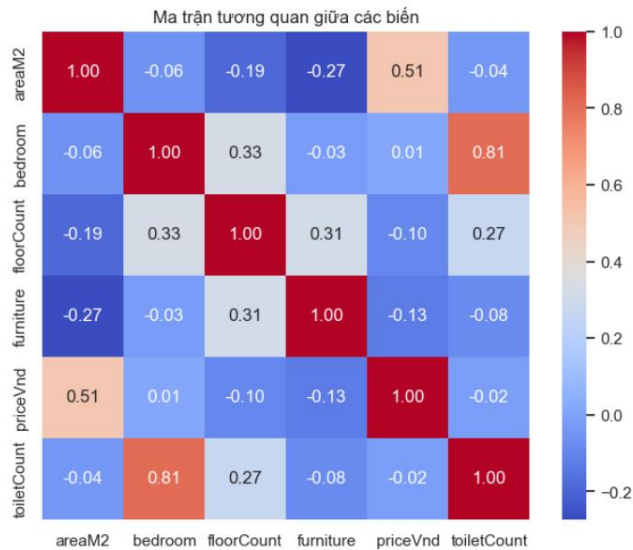
<>:1: SyntaxWarning: invalid escape sequence '\\d'
<>:2: SyntaxWarning: invalid escape sequence '\\d'
<>:3: SyntaxWarning: invalid escape sequence '\\d'
<>:1: SyntaxWarning: invalid escape sequence '\\d'
<>:2: SyntaxWarning: invalid escape sequence '\\d'
<>:3: SyntaxWarning: invalid escape sequence '\\d'
C:\Users\trinh\AppData\Local\Temp\ipykernel_16204\2695795723.py:1: SyntaxWarning: invalid escape sequence '\\d'
df['floorCount'] = df['floorCount'].str.extract('(\\d+)').astype(float)
C:\Users\trinh\AppData\Local\Temp\ipykernel_16204\2695795723.py:2: SyntaxWarning: invalid escape sequence '\\d'
df['bedroom'] = df['bedroom'].str.extract('(\\d+)').astype(float)
C:\Users\trinh\AppData\Local\Temp\ipykernel_16204\2695795723.py:3: SyntaxWarning: invalid escape sequence '\\d'
df['toiletCount'] = df['toiletCount'].str.extract('(\\d+)').astype(float)
```

Dữ liệu được sử dụng để huấn luyện

[154]:	df						
[154]:		areaM2	bedroom	floorCount	furniture	priceVnd	toiletCount
	0	88.00	3.0	4.0	1.0	1.400000e+10	2.0
	1	240.00	4.0	4.0	0.0	1.579091e+10	4.0
	2	80.00	3.0	4.0	1.0	1.650000e+10	2.0
	3	103.35	5.0	4.0	1.0	1.579091e+10	5.0
	4	208.00	3.0	4.0	1.0	3.510000e+10	2.0

	112	285.00	4.0	4.0	1.0	1.579091e+10	4.0
	113	50.00	3.0	4.0	1.0	9.700000e+09	2.0
	114	105.00	6.0	5.0	1.0	2.450000e+10	6.0
	115	90.00	6.0	5.0	1.0	2.050000e+10	4.0
	116	103.35	3.0	4.0	1.0	4.450000e+09	3.0
	117 rows x 6 columns						

+ biểu đồ thể hiện sự tương quan của dữ liệu



+ Thực hiện chia tập dữ liệu sử dụng sklearn

```
[167]: from sklearn.model_selection import train_test_split

# Chọn các cột để huấn luyện và cột mục tiêu
X = df[['areaM2', 'bedroom', 'floorCount', 'furniture', 'toiletCount']] # Các đặc trưng
y = df['priceVnd'] # Cột mục tiêu

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Kiểm tra kích thước của các tập dữ liệu
print("Kích thước của tập huấn luyện:", X_train.shape, y_train.shape)
print("Kích thước của tập kiểm tra:", X_test.shape, y_test.shape)

Kích thước của tập huấn luyện: (93, 5) (93,)
Kích thước của tập kiểm tra: (24, 5) (24,)
```

B2: xây dựng mô hình CNN

```
[38]: from tensorflow.keras import models, layers

# Define the CNN model with 5 layers
model_5_layers = models.Sequential([
    layers.Conv2D(32, (3, 1), activation='relu', input_shape=(X_train.shape[1], 1, 1), padding='same'),
    layers.Conv2D(64, (3, 1), activation='relu', padding='same'),
    layers.Conv2D(128, (2, 1), activation='relu', padding='same'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
```

B3: Xây dựng mô hình RNN

```
[41]: from tensorflow.keras import models, layers

# Định nghĩa mô hình RNN với 5 lớp
model_rnn = models.Sequential([
    layers.SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], 1)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
```

B4: Xây dựng mô hình LSTM

```

sequence as model's input using an Input(shape) object as the first layer in the model instead.
super().__init__(**kwargs)

[43]: from tensorflow.keras import models, layers

# Định nghĩa mô hình LSTM với 5 lớp
model_lstm = models.Sequential([
    layers.LSTM(50, activation='relu', input_shape=(X_train.shape[1], 1)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])

```

Huấn luyện mô hình

CNN

```

[47]: # Biên dịch mô hình CNN
model_5_layers.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Huấn Luyện mô hình CNN
history_cnn = model_5_layers.fit(X_train_cnn, y_train, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
400/400 — 8s 9ms/step - loss: 9761026.0000 - mae: 2379.9983 - val_loss: 2560057.5000 - val_mae: 1286.9131
Epoch 2/10
400/400 — 4s 9ms/step - loss: 2201532.7500 - mae: 1187.0421 - val_loss: 559031.0625 - val_mae: 595.8951
Epoch 3/10
400/400 — 3s 8ms/step - loss: 296351.6250 - mae: 398.1218 - val_loss: 73569.5625 - val_mae: 175.2994
Epoch 4/10
400/400 — 4s 9ms/step - loss: 60729.2656 - mae: 164.1838 - val_loss: 51203.1367 - val_mae: 161.8188
Epoch 5/10
400/400 — 3s 8ms/step - loss: 36792.5117 - mae: 128.7102 - val_loss: 27246.9395 - val_mae: 100.5994
Epoch 6/10
400/400 — 3s 8ms/step - loss: 27184.0566 - mae: 111.0227 - val_loss: 23986.3047 - val_mae: 108.1934
Epoch 7/10
400/400 — 3s 8ms/step - loss: 23614.5605 - mae: 104.5998 - val_loss: 19131.8105 - val_mae: 93.1597
Epoch 8/10
400/400 — 4s 9ms/step - loss: 17168.4199 - mae: 90.5161 - val_loss: 13528.9688 - val_mae: 75.5157
Epoch 9/10
400/400 — 4s 9ms/step - loss: 14360.1270 - mae: 82.0994 - val_loss: 9952.9473 - val_mae: 62.1394
Epoch 10/10
400/400 — 3s 8ms/step - loss: 13797.7822 - mae: 84.5913 - val_loss: 8625.2822 - val_mae: 59.6123

```

RNN

```

[48]: # Biên dịch mô hình RNN
model_rnn.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Huấn Luyện mô hình RNN
history_rnn = model_rnn.fit(X_train_rnn_lstm, y_train, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
400/400 — 6s 7ms/step - loss: 12266514.0000 - mae: 2782.1953 - val_loss: 3423380.7500 - val_mae: 1506.7629
Epoch 2/10
400/400 — 2s 5ms/step - loss: 3314090.5000 - mae: 1478.6577 - val_loss: 3161868.2500 - val_mae: 1444.2739
Epoch 3/10
400/400 — 2s 6ms/step - loss: 2979056.5000 - mae: 1392.4360 - val_loss: 2272800.0000 - val_mae: 1224.6818
Epoch 4/10
400/400 — 2s 5ms/step - loss: 1949621.3750 - mae: 1127.1321 - val_loss: 1621819.3750 - val_mae: 1035.0143
Epoch 5/10
400/400 — 2s 5ms/step - loss: 1466300.8750 - mae: 985.4668 - val_loss: 1236954.3750 - val_mae: 911.7713
Epoch 6/10
400/400 — 2s 5ms/step - loss: 1196364.2500 - mae: 891.6934 - val_loss: 1000153.8125 - val_mae: 813.0414
Epoch 7/10
400/400 — 2s 5ms/step - loss: 798202.5625 - mae: 731.0916 - val_loss: 456179.2812 - val_mae: 553.1849
Epoch 8/10
400/400 — 2s 5ms/step - loss: 345834.0000 - mae: 481.9360 - val_loss: 86146.9688 - val_mae: 243.7996
Epoch 9/10
400/400 — 2s 5ms/step - loss: 54027.7070 - mae: 186.5742 - val_loss: 11538.5615 - val_mae: 86.2762
Epoch 10/10
400/400 — 2s 5ms/step - loss: 10823.3740 - mae: 82.4240 - val_loss: 5648.4644 - val_mae: 59.7902

```

LSTM

```

51]: # Biên dịch mô hình LSTM
model_lstm.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Huấn luyện mô hình LSTM
history_lstm = model_lstm.fit(X_train_rnn_lstm, y_train, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
400/400 ————— 8s 9ms/step - loss: 13834750.0000 - mae: 2842.9189 - val_loss: 1103957.8750 - val_mae: 864.7368
Epoch 2/10
400/400 ————— 3s 7ms/step - loss: 1005665.8750 - mae: 834.3998 - val_loss: 878642.0625 - val_mae: 785.9733
Epoch 3/10
400/400 ————— 3s 6ms/step - loss: 906880.3750 - mae: 797.7617 - val_loss: 837829.7500 - val_mae: 766.8002
Epoch 4/10
400/400 ————— 3s 7ms/step - loss: 838735.0000 - mae: 770.7191 - val_loss: 728919.6875 - val_mae: 724.3583
Epoch 5/10
400/400 ————— 3s 7ms/step - loss: 730811.6875 - mae: 724.4357 - val_loss: 658885.8125 - val_mae: 693.6937
Epoch 6/10
400/400 ————— 3s 7ms/step - loss: 675127.0000 - mae: 694.8762 - val_loss: 621255.7500 - val_mae: 671.9487
Epoch 7/10
400/400 ————— 2s 6ms/step - loss: 635826.8125 - mae: 675.3798 - val_loss: 549103.7500 - val_mae: 633.2079
Epoch 8/10
400/400 ————— 3s 6ms/step - loss: 539206.3125 - mae: 623.5603 - val_loss: 451450.1562 - val_mae: 569.5582
Epoch 9/10
400/400 ————— 3s 6ms/step - loss: 422905.1875 - mae: 547.1137 - val_loss: 274407.7188 - val_mae: 439.0622
Epoch 10/10
400/400 ————— 3s 6ms/step - loss: 252697.2188 - mae: 411.7122 - val_loss: 84809.3672 - val_mae: 235.9654

```

Tính toán các tham số mae, rmse, mse

```

y_pred_cnn = model_5.layers.predict(X_test_cnn)
y_pred_rnn = model_rnn.predict(X_test_rnn_lstm)
y_pred_lstm = model_lstm.predict(X_test_rnn_lstm)

# Tính các chỉ số MAE, MSE, RMSE cho từng mô hình
def calculate_metrics(y_test, y_pred):
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    return mae, mse, rmse

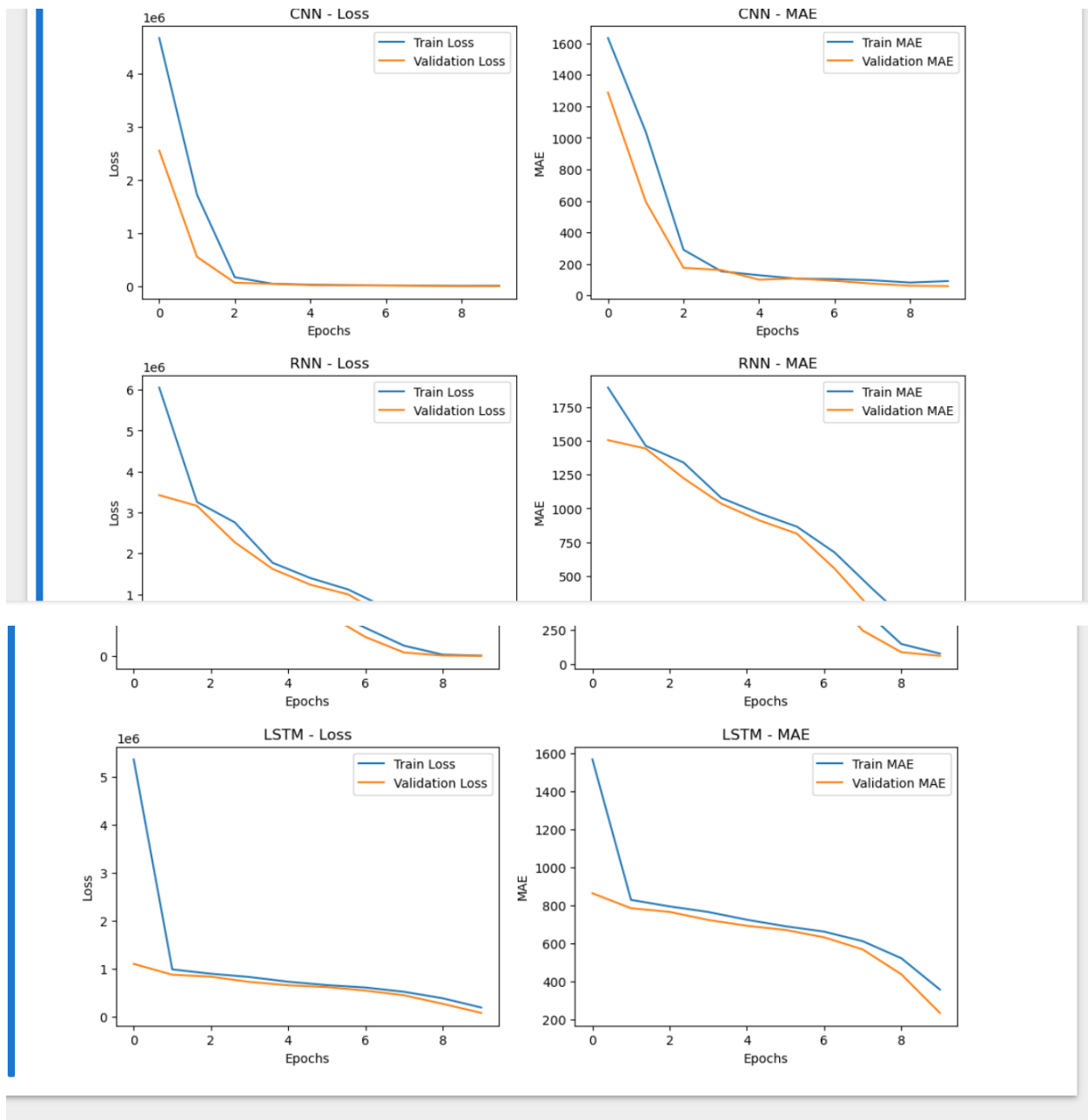
mae_cnn, mse_cnn, rmse_cnn = calculate_metrics(y_test, y_pred_cnn)
mae_rnn, mse_rnn, rmse_rnn = calculate_metrics(y_test, y_pred_rnn)
mae_lstm, mse_lstm, rmse_lstm = calculate_metrics(y_test, y_pred_lstm)

# In kết quả
print("CNN: MAE =", mae_cnn, "MSE =", mse_cnn, "RMSE =", rmse_cnn)
print("RNN: MAE =", mae_rnn, "MSE =", mse_rnn, "RMSE =", rmse_rnn)
print("LSTM: MAE =", mae_lstm, "MSE =", mse_lstm, "RMSE =", rmse_lstm)

125/125 ————— 1s 5ms/step
125/125 ————— 1s 3ms/step
125/125 ————— 1s 4ms/step
CNN: MAE = 59.04237149169922 MSE = 7934.870854821794 RMSE = 89.07789206543784
RNN: MAE = 59.52021856079101 MSE = 5698.819090803399 RMSE = 75.49052318538665
LSTM: MAE = 235.36910126708983 MSE = 85767.43792174855 RMSE = 292.86078249186687

```

Vẽ biểu đồ so sánh



3. Tương tự câu 2 cho dữ liệu comment sản phẩm trên tiki, shoppe sử dụng bert
 Hiển thị tập dữ liệu

[4]:

	text	label
0	sp tốt giao tầm tuần tại hàng qte	0
1	mn nên mua nha hình ảnh mang tính chất minh họa	0
2	h h d d s s d d h s h d d s s h s h s d d	0
3	đã nhận đx hàng đẹp giá rẻ nhưng về dép bị...	0
4	chất liệu cao su	0
...
3730	màu sắc đúng	1
3731	chất lượng sản phẩm	1
3732	đúng với mô tả khá tốt	1
3733	túi phù hợp với giá tiền	1
3734	chất lượng sản phẩm tốt	1

3735 rows x 2 columns

Xây dựng mô hình


```

✓ 15s from transformers import BertTokenizer


# Tải tokenizer của BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')


# Mã hóa dữ liệu
train_encodings = tokenizer(list(X_train), truncation=True, padding=True, max_length=128)
test_encodings = tokenizer(list(X_test), truncation=True, padding=True, max_length=128)


```


 /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning: The secret `HF_TOKEN` does not exist in your colab secrets. To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as secret in your notebook. You will be able to reuse this secret in all of your notebooks. Please note that authentication is recommended but still optional to access public models or datasets.

warnings.warn(

tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 984B/s]

vocab.txt: 100%  232k/232k [00:00<00:00, 1.10MB/s]

tokenizer.json: 100%  466k/466k [00:00<00:00, 8.23MB/s]

config.json: 100%  570/570 [00:00<00:00, 9.91kB/s]

/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set

warnings.warn(

Chia tập dữ liệu

```
import torch

class TextDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# Tạo dataset cho train và test
train_dataset = TextDataset(train_encodings, y_train.values)
test_dataset = TextDataset(test_encodings, y_test.values)
```

Huấn luyện mô hình

```
from transformers import BertForSequenceClassification, Trainer, TrainingArguments

# Tạo mô hình BERT cho phân loại văn bản
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# Định nghĩa các tham số huấn luyện
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

# Tạo Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)

# Huấn luyện mô hình
trainer.train()
```

*** models.safetensors: 100% 440M/440M [00:01<00:00, 242MB/s]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight'] You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

wandb: WARNING The 'run_name' is currently set to the same value as 'TrainingArguments.output_dir'. If this was not intended, please specify a different run name by setting the 'TrainingArguments.run_name' attribute.

wandb: Using wandb-core as the SDK backend. Please refer to <https://wandb.me/wandb-core> for more information.

wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: <https://wandb.me/wandb-server>)

wandb: You can find your API key in your browser here: <https://wandb.ai/authorize>

wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:

Đánh giá mô hình

```
trainer.evaluate()
```

[12/12 00:04]

```
{'eval_loss': 0.4673506021499634,
 'eval_runtime': 4.9393,
 'eval_samples_per_second': 151.235,
 'eval_steps_per_second': 2.429,
 'epoch': 3.0}
```

Dự đoán kết quả

```
# Kiểm tra xem mô hình có đang ở trên GPU không
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Mã hóa văn bản mới và chuyển dữ liệu sang GPU nếu cần
new_text = ["Sản phẩm rất tốt", "Giá cả hàng chệch và hàng bị lỗi"]
new_encodings = tokenizer(new_text, truncation=True, padding=True, max_length=128, return_tensors="pt")
new_encodings = {key: val.to(device) for key, val in new_encodings.items()} # Chuyển tensor sang GPU

# Dự đoán nhãn
with torch.no_grad():
    outputs = model(**new_encodings)
    predictions = torch.argmax(outputs.logits, dim=1)

print(predictions) # In ra nhãn dự đoán cho từng câu

tensor([[1, 1], device='cuda:0'])
```