CHƯƠNG I

1.1 **Tại sao lại có Machine Learning? Mục đích của Machine Learning**?

* **Tại sao lại có Machine Learning?**
* **Sự phát triển của dữ liệu lớn (Big Data):**  
  Thế giới hiện nay đang chứng kiến sự bùng nổ dữ liệu. Từ dữ liệu do con người tạo ra như văn bản, hình ảnh, video đến dữ liệu từ các thiết bị IoT, cảm biến hay mạng xã hội. Khối lượng dữ liệu này lớn đến mức con người không thể tự mình phân tích và khai thác giá trị một cách hiệu quả.  
  Ví dụ: Hệ thống thương mại điện tử như Amazon thu thập dữ liệu về hành vi mua sắm của hàng triệu khách hàng mỗi ngày. Để phân tích thói quen mua sắm và đưa ra khuyến nghị sản phẩm phù hợp, cần có một hệ thống tự động và thông minh.
* **Hạn chế của lập trình truyền thống:**  
  Các chương trình máy tính truyền thống hoạt động dựa trên quy tắc được lập trình sẵn. Nhưng nhiều vấn đề không thể mô tả dễ dàng qua các quy tắc này, ví dụ như nhận diện hình ảnh, giọng nói, hay phân tích ngữ nghĩa trong văn bản.  
  Machine Learning xuất hiện để giải quyết vấn đề này, khi máy tính có thể **học** từ dữ liệu mà không cần được lập trình tường minh.
* **Sự tiến bộ của công nghệ tính toán:**  
  Với sự phát triển của phần cứng (GPU, TPU) và khả năng lưu trữ dữ liệu, Machine Learning đã trở nên khả thi và hiệu quả hơn. Việc xử lý hàng triệu hoặc thậm chí hàng tỷ dữ liệu trong thời gian ngắn không còn là trở ngại.
* **Cần tối ưu hóa và tự động hóa:**  
  Các doanh nghiệp và tổ chức cần giải pháp để tối ưu hóa chi phí, tự động hóa quy trình và cải thiện hiệu suất. Machine Learning trở thành công cụ mạnh mẽ để giải quyết các bài toán này, từ tự động hóa dịch vụ khách hàng bằng chatbot đến dự đoán nhu cầu thị trường.
* **Mục đích của Machine Learning**
* **Tự động hóa quyết định và dự đoán:**  
  Machine Learning giúp đưa ra dự đoán hoặc quyết định chính xác hơn dựa trên dữ liệu lịch sử.
  + **Ứng dụng thực tế:**
    - Ngân hàng sử dụng ML để phát hiện giao dịch gian lận.
    - Hệ thống dự báo thời tiết hoặc dự đoán giá cổ phiếu.
* **Tăng cường trí tuệ nhân tạo (AI):**  
  Machine Learning là một phần cốt lõi trong trí tuệ nhân tạo. Nhờ đó, các hệ thống AI có khả năng học hỏi và cải thiện hiệu suất qua thời gian.
  + Ví dụ: Các hệ thống trợ lý ảo như Siri, Google Assistant sử dụng ML để hiểu ngôn ngữ tự nhiên và phản hồi chính xác hơn.
* **Tối ưu hóa và cá nhân hóa trải nghiệm người dùng:**
  + **Cá nhân hóa:** ML giúp các hệ thống như Netflix hoặc Spotify đề xuất nội dung dựa trên sở thích của người dùng.
  + **Tối ưu hóa:** Các công ty logistics như FedEx sử dụng ML để tối ưu hóa lộ trình vận chuyển.
* **Phân tích dữ liệu phức tạp:**  
  Machine Learning có thể phát hiện các mẫu và mối quan hệ phức tạp trong dữ liệu mà con người không thể dễ dàng nhận ra.
  + Ví dụ: Trong y học, ML được dùng để phân tích ảnh chụp X-quang nhằm phát hiện bệnh sớm.
* **Giải quyết các vấn đề phi tuyến tính:**  
  Nhiều bài toán thực tế có tính phi tuyến tính cao, ví dụ như dự đoán thị trường tài chính hay phân loại hình ảnh. Machine Learning, đặc biệt là các phương pháp như Deep Learning, đã chứng minh khả năng vượt trội trong các bài toán này.
* **Thay đổi cách làm việc truyền thống:**  
  ML không chỉ tối ưu hóa các hệ thống hiện có mà còn tạo ra những phương pháp làm việc hoàn toàn mới.
  + Ví dụ: Xe tự lái sử dụng ML để nhận diện đường đi, biển báo, và người đi bộ.
* **Hỗ trợ nghiên cứu khoa học:**  
  Trong các lĩnh vực như vật lý, sinh học, và hóa học, ML giúp các nhà khoa học phân tích dữ liệu lớn và phát hiện ra những kiến thức mới.
  + Ví dụ: ML được dùng để tìm ra các loại thuốc mới hoặc mô hình hóa các phản ứng hóa học.

1.2 Trình bày các kỹ thuật machine learning cơ bản

Các kỹ thuật cơ bản trong Machine Learning thường được chia thành ba nhóm chính: Học có giám sát (Supervised Learning), Học không giám sát (Unsupervised Learning), và Học tăng cường (Reinforcement Learning).

**1. Học có giám sát (Supervised Learning)**

Học có giám sát liên quan đến việc sử dụng một tập dữ liệu đã được gắn nhãn để huấn luyện mô hình. Mô hình học cách ánh xạ từ đầu vào (input) sang đầu ra (output) dựa trên dữ liệu mẫu.

**Các phương pháp phổ biến:**

* **Hồi quy (Regression):**  
  Dùng để dự đoán giá trị liên tục.
  + **Ví dụ:** Dự đoán giá nhà, dự đoán doanh số bán hàng.
  + Thuật toán: Hồi quy tuyến tính (Linear Regression), Hồi quy Ridge, Lasso.
* **Phân loại (Classification):**  
  Dùng để phân loại dữ liệu vào các nhóm cụ thể.
  + **Ví dụ:** Phân loại email là spam hoặc không spam, dự đoán bệnh lý từ triệu chứng.
  + Thuật toán:
    - Hồi quy Logistic (Logistic Regression)
    - K-Nearest Neighbors (KNN)
    - Support Vector Machines (SVM)
    - Random Forest, Gradient Boosting Machines (GBM), XGBoost.

**2. Học không giám sát (Unsupervised Learning)**

Học không giám sát làm việc với dữ liệu chưa gắn nhãn. Mục tiêu là khám phá cấu trúc tiềm ẩn hoặc các mẫu trong dữ liệu.

**Các phương pháp phổ biến:**

* **Phân cụm (Clustering):**  
  Phân nhóm các đối tượng có đặc điểm tương tự.
  + **Ví dụ:** Phân nhóm khách hàng dựa trên hành vi mua sắm.
  + Thuật toán:
    - K-Means
    - Hierarchical Clustering
    - DBSCAN
* **Giảm số chiều (Dimensionality Reduction):**  
  Dùng để giảm bớt số lượng biến (features) trong dữ liệu mà vẫn giữ được thông tin quan trọng.
  + **Ví dụ:** Trực quan hóa dữ liệu, tăng tốc độ xử lý.
  + Thuật toán:
    - Principal Component Analysis (PCA)
    - t-Distributed Stochastic Neighbor Embedding (t-SNE).
* **Phát hiện bất thường (Anomaly Detection):**  
  Tìm ra các điểm dữ liệu không bình thường trong tập hợp.
  + **Ví dụ:** Phát hiện giao dịch gian lận, lỗi trong hệ thống IoT.
  + Thuật toán: Isolation Forest, One-Class SVM.

**3. Học tăng cường (Reinforcement Learning - RL)**

Học tăng cường dựa trên cơ chế học qua thử-sai (trial-and-error). Mô hình (hay tác nhân, agent) học cách đưa ra hành động trong môi trường để đạt được phần thưởng tối đa.

**Các phương pháp phổ biến:**

* **Q-Learning:**  
  Học bảng giá trị Q để tối ưu hóa chính sách hành động.
* **Deep Q-Learning (DQL):**  
  Kết hợp học tăng cường với deep learning, dùng mạng nơ-ron để xấp xỉ giá trị Q.
* **Policy Gradient:**  
  Học trực tiếp chiến lược (policy) thay vì giá trị Q.
* **Ứng dụng:**
  + Chơi game tự động (AlphaGo, Dota 2).
  + Điều khiển robot.

**4. Kỹ thuật bổ trợ và thuật toán nâng cao**

Dưới đây là một số kỹ thuật và thuật toán thường được áp dụng để cải thiện các mô hình Machine Learning:

**Kỹ thuật xử lý trước dữ liệu:**

* Chuẩn hóa dữ liệu (Normalization), tiêu chuẩn hóa (Standardization).
* Xử lý dữ liệu bị thiếu (Missing Data Handling).
* Encoding (One-Hot Encoding, Label Encoding).

**Kỹ thuật điều chỉnh mô hình:**

* **Regularization:** Giảm thiểu overfitting. Ví dụ: L1, L2 Regularization.
* **Cross-Validation:** Đánh giá mô hình bằng cách chia dữ liệu thành nhiều phần nhỏ.

**Học sâu (Deep Learning):**

* **Mạng nơ-ron nhân tạo (Artificial Neural Networks - ANN):** Mô hình cơ bản trong học sâu.
* **Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN):** Dùng trong xử lý ảnh.
* **Mạng nơ-ron hồi tiếp (Recurrent Neural Networks - RNN):** Dùng trong xử lý chuỗi, như dịch máy.

**1.3 Các kỹ thuật của Deep learning**

**Các kỹ thuật của Deep Learning** tập trung vào việc xây dựng và huấn luyện các mô hình mạng nơ-ron sâu (Deep Neural Networks) để giải quyết các bài toán phức tạp liên quan đến dữ liệu lớn, phi cấu trúc như hình ảnh, video, âm thanh và văn bản.

* **Mạng nơ-ron nhân tạo (Artificial Neural Networks - ANN)**
* **Đặc điểm:**  
  ANN là mô hình cơ bản nhất của Deep Learning, bao gồm các lớp đầu vào, ẩn và đầu ra. Mỗi lớp chứa các nút (neuron) liên kết với nhau thông qua trọng số và hàm kích hoạt.
* **Ứng dụng:**
  + Dự đoán giá trị (regression).
  + Phân loại cơ bản (classification).
* **Hạn chế:**  
  Không hiệu quả với dữ liệu hình ảnh, âm thanh hoặc chuỗi thời gian.
* **Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN)**
* **Đặc điểm:**  
  CNN được thiết kế đặc biệt để xử lý dữ liệu hình ảnh. Nó sử dụng các lớp tích chập (convolutional layers) để trích xuất đặc trưng từ ảnh và các lớp gộp (pooling layers) để giảm chiều dữ liệu.
* **Ứng dụng:**
  + Nhận diện khuôn mặt (facial recognition).
  + Phân loại hình ảnh (image classification).
  + Phát hiện vật thể (object detection).
  + Phân đoạn hình ảnh (image segmentation).
* **Các kiến trúc phổ biến:**
  + AlexNet, VGG, ResNet, EfficientNet.
* **Mạng nơ-ron hồi tiếp (Recurrent Neural Networks - RNN)**
* **Đặc điểm:**  
  RNN được thiết kế để xử lý dữ liệu tuần tự hoặc dữ liệu thời gian, chẳng hạn như văn bản, âm thanh, hoặc chuỗi thời gian. Nó sử dụng trạng thái ẩn (hidden state) để ghi nhớ thông tin từ các bước trước.
* **Ứng dụng:**
  + Dịch máy (machine translation).
  + Dự đoán chuỗi thời gian (time series forecasting).
  + Phân tích cảm xúc (sentiment analysis).
* **Hạn chế:**
  + Vanishing gradient (độ dốc mất dần) khiến việc học trên chuỗi dài trở nên khó khăn.
* **LSTM và GRU (Long Short-Term Memory & Gated Recurrent Unit)**
* **Đặc điểm:**  
  LSTM và GRU là cải tiến của RNN, giúp khắc phục vấn đề vanishing gradient. Chúng sử dụng các cổng (gates) để kiểm soát dòng thông tin.
* **Ứng dụng:**
  + Nhận dạng giọng nói (speech recognition).
  + Dự đoán chuỗi dữ liệu dài (long-term dependencies).
* **Mạng nơ-ron tự mã hóa (Autoencoders)**
* **Đặc điểm:**  
  Autoencoders học cách nén dữ liệu vào một không gian nhỏ gọn (latent space) và sau đó tái tạo lại dữ liệu ban đầu.
* **Ứng dụng:**
  + Giảm số chiều dữ liệu.
  + Loại bỏ nhiễu (denoising).
  + Sinh dữ liệu mới (data generation).
* **Biến thể:**
  + Denoising Autoencoders.
  + Variational Autoencoders (VAE).
* **Mạng sinh đối kháng (Generative Adversarial Networks - GANs)**
* **Đặc điểm:**  
  GANs bao gồm hai mạng: một mạng sinh (generator) và một mạng phân biệt (discriminator) hoạt động đối kháng. Generator cố gắng sinh dữ liệu giống thật, trong khi discriminator cố gắng phân biệt dữ liệu thật và giả.
* **Ứng dụng:**
  + Tạo ảnh chân dung giả (deepfake).
  + Tăng cường dữ liệu (data augmentation).
  + Phục hồi ảnh (image restoration).
* **Các kiến trúc nổi bật:**
  + DCGAN, CycleGAN, StyleGAN.
* **Mạng Transformer**
* **Đặc điểm:**  
  Transformer là kiến trúc sử dụng cơ chế attention để tập trung vào các phần quan trọng của dữ liệu. Nó đặc biệt hiệu quả trong xử lý dữ liệu tuần tự như văn bản.
* **Ứng dụng:**
  + Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP).
  + Mô hình GPT, BERT, và các biến thể.
* **Học chuyển giao (Transfer Learning)**
* **Đặc điểm:**  
  Học chuyển giao tận dụng các mô hình đã được huấn luyện trên tập dữ liệu lớn và tái sử dụng chúng cho các bài toán khác với ít dữ liệu hơn.
* **Ứng dụng:**
  + Phân loại ảnh với mô hình ResNet hoặc EfficientNet đã được huấn luyện trước.
  + Xử lý văn bản với BERT hoặc GPT
* **Học sâu tăng cường (Deep Reinforcement Learning)**
* **Đặc điểm:**  
  Kết hợp giữa học tăng cường và mạng nơ-ron sâu, cho phép tác nhân học cách thực hiện hành động trong môi trường phức tạp.
* **Ứng dụng:**
  + Xe tự lái.
  + Chơi game (AlphaGo, Dota 2 AI).
  + Điều khiển robot.
* **Dropout và Batch Normalization**
* **Dropout:** Kỹ thuật để giảm overfitting bằng cách "tắt" ngẫu nhiên một số nút (neuron) trong quá trình huấn luyện.
* **Batch Normalization:** Chuẩn hóa đầu ra của mỗi lớp trong mạng, giúp cải thiện tốc độ huấn luyện và độ ổn định.

**1.**4 Clean và chuẩn hoá Data

1.4.1 Các bước clean data và chuẩn hoá data

**Clean và chuẩn hóa dữ liệu (Data Cleaning and Data Normalization)** là hai bước quan trọng trong quy trình xử lý dữ liệu, đảm bảo dữ liệu có chất lượng tốt và sẵn sàng để sử dụng trong các thuật toán Machine Learning

* **Data Cleaning (Làm sạch dữ liệu)**

**Làm sạch dữ liệu** là bước đầu tiên và rất quan trọng trong quy trình tiền xử lý dữ liệu. Dữ liệu thô thường chứa lỗi, thiếu giá trị, hoặc không đồng nhất. Nếu không làm sạch, các mô hình Machine Learning có thể cho ra kết quả không chính xác hoặc kém hiệu quả.

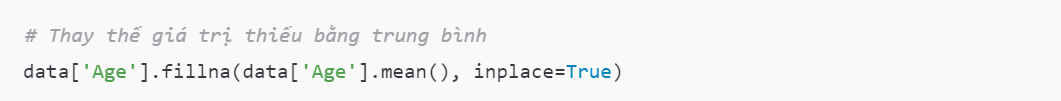
* **Xử lý dữ liệu bị thiếu (Missing Data)**

Dữ liệu bị thiếu là một vấn đề phổ biến, thường xảy ra trong các tập dữ liệu thực tế. Các nguyên nhân có thể bao gồm lỗi thu thập dữ liệu, không trả lời khảo sát, hoặc các vấn đề kỹ thuật.

**Các cách xử lý:**

* **Loại bỏ dữ liệu thiếu:**
  + Áp dụng nếu số lượng dữ liệu bị thiếu không đáng kể.
  + Ví dụ: Nếu chỉ có 1-2% dữ liệu thiếu, bạn có thể xóa các dòng hoặc cột chứa giá trị này.
* **Thay thế dữ liệu:**
  + **Bằng giá trị trung bình:** Tốt với dữ liệu liên tục.
  + **Bằng giá trị trung vị:** Phù hợp khi dữ liệu chứa ngoại lai.
  + **Dự đoán dữ liệu thiếu:** Sử dụng các mô hình Machine Learning như KNN hoặc regression để dự đoán.

**Ví dụ:**



**Những điều cần lưu ý:**

* Không nên xóa dữ liệu thiếu nếu chúng chiếm tỷ lệ lớn, vì điều này có thể làm mất thông tin quan trọng.

Cần kiểm tra sự phân phối của dữ liệu trước và sau khi thay thế.

* **Loại bỏ dữ liệu trùng lặp (Duplicate Data)**

Dữ liệu trùng lặp có thể xảy ra khi kết hợp nhiều nguồn dữ liệu hoặc do lỗi nhập liệu.

**Cách phát hiện và xử lý:**

1. Sử dụng phương pháp thống kê để đếm các bản ghi trùng lặp.
2. Loại bỏ chúng để giảm tải tính toán và tăng hiệu quả mô hình.

**Ví dụ**



**Ứng dụng:**

Trong hệ thống e-commerce, loại bỏ các đơn hàng bị nhập nhiều lần giúp cải thiện chất lượng phân tích.

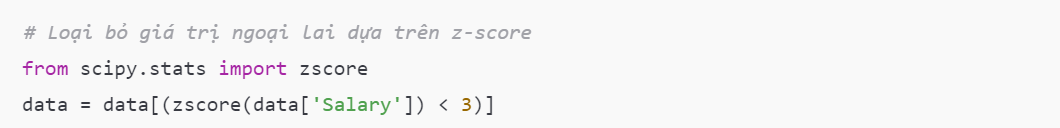
**+ Xử lý giá trị ngoại lai (Outliers)**

**Giá trị ngoại lai** là các điểm dữ liệu không bình thường, thường có giá trị quá lớn hoặc quá nhỏ so với phần còn lại.

**Phát hiện giá trị ngoại lai:**

1. **Boxplot:** Minh họa phân phối dữ liệu và phát hiện các giá trị nằm ngoài IQR (interquartile range).
2. **Z-score:** Xác định các giá trị cách xa trung bình hơn 3 lần độ lệch chuẩn.

**Ví dụ**



**Cách xử lý:**

* Loại bỏ giá trị ngoại lai hoặc biến đổi chúng (log-transform).

**Những lưu ý:**

* Đừng loại bỏ giá trị ngoại lai quá mức nếu chúng có ý nghĩa thực tế (ví dụ: doanh thu rất cao từ một khách hàng đặc biệt).
* **Data Normalization (Chuẩn hóa dữ liệu)**

Chuẩn hóa dữ liệu là bước quan trọng để các đặc trưng có cùng phạm vi hoặc phân phối, đặc biệt khi sử dụng các thuật toán nhạy cảm với tỷ lệ như k-NN hoặc SVM.

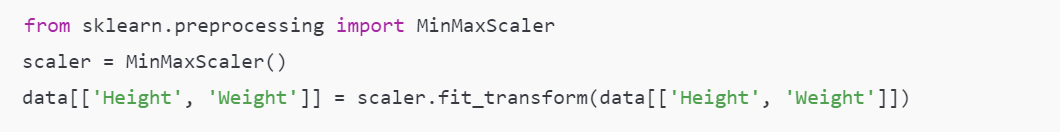
**+ Min-Max Scaling**

* Công thức:

x =

* **Ứng dụng:**
  + Dữ liệu cần đưa về [0, 1].
  + Phù hợp với các bài toán dự đoán giá nhà hoặc phân loại cơ bản.

Ví dụ:



**Ưu điểm:**

* Đơn giản, dễ hiểu.
* Hiệu quả khi không có ngoại lai.

**Nhược điểm:**

* Nhạy cảm với giá trị ngoại lai.

**+ Standardization (Z-Score Scaling)**

* **Công thức:**

x =

**Ứng dụng:**

* Các bài toán cần dữ liệu có phân phối chuẩn.
* Sử dụng trong thuật toán Linear Regression, Logistic Regression.

**+ Log Transformation**

* **Ứng dụng:**
  + Làm giảm tác động của giá trị lớn.
  + Phù hợp với dữ liệu có phân phối lệch phải.

Ví dụ:



**Kết luận**

Data Cleaning và Data Normalization là hai bước không thể thiếu để đảm bảo chất lượng dữ liệu và tối ưu hóa hiệu suất của các mô hình Machine Learning. Khi thực hiện đúng, dữ liệu sẽ trở nên đáng tin cậy, giúp các mô hình học tập hiệu quả hơn, từ đó cải thiện kết quả đầu ra.

1.4.2 Ví dụ

Ví dụ 1: Clean và chuẩn hoá dữ liệu điểm thi

Link code: [D:\HTTM\Tiểu luận\code\_chuan\_hoa\chuan\_hoa\_diem\_thi.ipynb](file:///D:\HTTM\Tiểu%20luận\code_chuan_hoa\chuan_hoa_diem_thi.ipynb)

Bước 1: Clean data

import pandas as pd

*# Load the three uploaded files*

file\_1 = 'data\_pttk\_1\_new.xlsx'

file\_2 = 'data\_pttk\_2\_new.xlsx'

file\_3 = 'data\_pttk\_3\_new.csv'

*# Load Excel files with possible multiple sheets*

data\_1 = pd.read\_excel(file\_1)

data\_2 = pd.read\_excel(file\_2)

*# Load the CSV file*

data\_3 = pd.read\_csv(file\_3)

*# Displaying the first few rows of each dataset to inspect*

data\_1\_head = data\_1.head()

data\_2\_head = data\_2.head()

data\_3\_head = data\_3.head()

data\_1\_head

*# Clean the first file (data\_1)*

*# Renaming columns for better understanding*

data\_1\_cleaned = data\_1.rename(columns={

'Unnamed: 3': 'extra\_col\_1',

'Unnamed: 5': 'extra\_col\_2',

'Unnamed: 7': 'extra\_col\_3',

'mã sv': 'student\_id',

0.1: 'assignment\_1',

'0.1.1': 'assignment\_2',

0.2: 'project',

'điểm thi': 'final\_score'

})

*# Mapping Vietnamese textual scores to numbers*

score\_map = {

"sáu điểm": 6.00,

"bảy hai lăm": 7.25,

"bảy rưỡi": 7.50

}

data\_1\_cleaned['final\_score'] = data\_1\_cleaned['final\_score'].replace(score\_map)

*# Dropping unnecessary columns*

data\_1\_cleaned = data\_1\_cleaned.drop(columns=['extra\_col\_1', 'extra\_col\_2', 'extra\_col\_3'])

*# Clean the second file (data\_2) in a similar way*

data\_2\_cleaned = data\_2.rename(columns={

'Unnamed: 3': 'extra\_col\_1',

'Unnamed: 5': 'extra\_col\_2',

'Unnamed: 7': 'extra\_col\_3',

'mã sv': 'student\_id',

'0.1': 'assignment\_1',

'0.1.1': 'assignment\_2',

'0.2': 'project',

'điểm thi': 'final\_score'

})

*# Dropping unnecessary columns*

data\_2\_cleaned = data\_2\_cleaned.drop(columns=['extra\_col\_1', 'extra\_col\_2', 'extra\_col\_3'])

*# Clean the third file (data\_3)*

data\_3\_cleaned = data\_3.rename(columns={

'Họ Tên': 'student\_id',

'10%': 'assignment\_1',

'20%': 'assignment\_2',

'20%.1': 'project',

'Thi': 'final\_score'

})

*# Check for missing values and clean data*

print(data\_1\_cleaned)

data\_2\_cleaned = data\_2.rename(columns={

'Unnamed: 3': 'extra\_col\_1',

'Unnamed: 5': 'extra\_col\_2',

'Unnamed: 7': 'extra\_col\_3',

'mã sv': 'student\_id',

0.1: 'assignment\_1',

'0.1.1': 'assignment\_2',

0.2: 'project',

'điểm thi': 'final\_score'

})

*# Dropping unnecessary columns*

data\_2\_cleaned = data\_2\_cleaned.drop(columns=['extra\_col\_1', 'extra\_col\_2', 'extra\_col\_3'])

*# Display the cleaned data for inspection*

print(data\_2\_cleaned)

data\_3\_cleaned = data\_3.rename(columns={

'Họ Tên': 'student\_id',

'10%': 'assignment\_1',

'20%': 'assignment\_2',

'20%.1': 'project',

'Thi': 'final\_score'

})

*# Displaying the cleaned data for inspection*

data\_3\_cleaned.head()

Bước 2: Chuẩn hoá data dùng công thức min-max

*# Chuẩn hoá data -> [0,1]*

cols\_to\_normalize = ["assignment\_1", "assignment\_2", "project", "final\_score"]

df\_normalized = finnal\_data.copy()

df\_normalized[cols\_to\_normalize] = (finnal\_data[cols\_to\_normalize] - finnal\_data[cols\_to\_normalize].min()) / (

finnal\_data[cols\_to\_normalize].max() - finnal\_data[cols\_to\_normalize].min()

)

*# Kết quả*

print(df\_normalized)

Kết quả thu được sau khi chuẩn hoá:

student\_id assignment\_1 assignment\_2 project final\_score

0 b1 0.900 0.777778 0.631579 0.600

1 b2 0.900 0.777778 0.631579 0.600

2 b3 1.000 1.000000 0.736842 0.725

3 b4 0.900 0.777778 0.631579 0.600

4 b5 1.000 0.888889 0.736842 0.750

.. ... ... ... ... ...

305 31\_151 0.825 0.861111 0.868421 0.800

306 31\_152 0.350 0.361111 0.631579 0.650

307 31\_153 0.725 0.722222 0.631579 0.650

308 31\_154 0.775 0.388889 0.789474 0.750

309 31\_155 0.000 0.000000 0.000000 0.000

[245 rows x 5 columns]

Ví dụ 2: Clean tập data giá nhà đất:

Link code: [D:\HTTM\Tiểu luận\code\_chuan\_hoa\chuan\_hoa\_gia\_nha\_dat.ipynb](file:///D:\HTTM\Tiểu%20luận\code_chuan_hoa\chuan_hoa_gia_nha_dat.ipynb)

Bước 1: Clean data

import pandas as pd

file\_path = "nhadathadong.csv"

*# Đọc file với dấu phân cách và hỗ trợ dấu ngoặc kép*

df = pd.read\_csv(file\_path, delimiter=",", quotechar='"')

print(df.head())

*# Bước 1: Làm sạch cột 'area'*

df['areaM2'] = pd.to\_numeric(df['areaM2'], errors='coerce') *# Chuyển đổi về số*

df['area'] = df['areaM2'].fillna(df['area']).apply(**lambda** x: str(x).replace(' m²', '').strip() if isinstance(x, str) else x)

df['area'] = pd.to\_numeric(df['area'], errors='coerce') *# Chuyển đổi về số*

*# Bước 2: Xử lý giá trị 'price' (tách và làm sạch các đơn vị)*

df['priceBil'] = pd.to\_numeric(df['priceBil'], errors='coerce') *# Giá tính bằng tỷ*

df['priceMil'] = pd.to\_numeric(df['priceMil'], errors='coerce') *# Giá tính bằng triệu*

df['pricePerM2'] = pd.to\_numeric(df['pricePerM2'], errors='coerce') *# Giá mỗi m2*

*# Bước 3: Xử lý các cột khác như 'direction', 'floorCount', 'road'*

df['floorCount'] = pd.to\_numeric(df['floorCount'], errors='coerce') *# Số tầng*

df['road'] = df['road'].apply(**lambda** x: str(x).strip() if isinstance(x, str) else x)

*# Bước 4: Điền giá trị mặc định hoặc loại bỏ dòng dữ liệu không hợp lệ*

df = df.dropna(subset=['area', 'priceBil']) *# Bỏ các dòng thiếu thông tin quan trọng*

df = df.fillna({"balconyDirection": "Không xác định", "legal": "Không rõ"}) *# Điền giá trị mặc định*

*# Bước 5: Chuẩn hóa tên cột*

df.rename(columns=**lambda** x: x.strip().lower().replace(" ", "\_"), inplace=True)

*# Ghi dữ liệu đã làm sạch ra file CSV*

clean\_file\_path = "nhadathadong\_cleaned.csv"

df.to\_csv(clean\_file\_path, index=False)

print(**f**"Dữ liệu đã làm sạch và lưu tại {clean\_file\_path}")

Bước 2: Chuẩn hoá dữ liệu:

import numpy as np

from sklearn.preprocessing import MinMaxScaler

*# Xử lý từng cột*

*# Chuyển cột 'price' sang kiểu chuỗi trước khi xử lý*

df["price"] = df["price"].astype(str)

*# Sau đó tiếp tục xử lý như bình thường*

df["price"] = df["price"].str.replace(" tỷ", "", regex=False).str.replace(",", ".", regex=False)

df["price"] = pd.to\_numeric(df["price"], errors="coerce") *# Chuyển thành số*

*## 2. Chuẩn hóa cột 'priceext' (loại bỏ "triệu/m²" và các ký tự không cần thiết)*

df["priceext"] = df["priceext"].astype(str)

df["priceext"] = df["priceext"].str.replace("[^0-9.,]", "", regex=True).str.replace(",", ".", regex=False)

df["priceext"] = pd.to\_numeric(df["priceext"], errors="coerce")

*## 3. Chuẩn hóa cột 'frontage' (loại bỏ "m")*

df["frontage"] = df["frontage"].astype(str)

df["frontage"] = df["frontage"].str.replace(" m", "", regex=False)

df["frontage"] = pd.to\_numeric(df["frontage"], errors="coerce")

*## 4. Chuẩn hóa cột 'bedroom' và 'toiletcount'*

df["bedroom"] = df["bedroom"].astype(str)

df["bedroom"] = df["bedroom"].str.extract("(\d+)").astype(float) *# Lấy số từ chuỗi*

df["toiletcount"] = df["toiletcount"].astype(str)

df["toiletcount"] = df["toiletcount"].str.extract("(\d+)").astype(float)

*## 5. Chuẩn hóa cột 'balconydirection' và 'direction' (loại bỏ giá trị không xác định)*

df["balconydirection"] = df["balconydirection"].replace("Không xác định", np.nan)

df["direction"] = df["direction"].replace("Không xác định", np.nan)

*## 6. Xử lý giá trị thiếu*

df.fillna({"balconydirection": "Không rõ", "direction": "Không rõ"}, inplace=True)

columns\_to\_scale = ['area', 'aream2', 'price', 'pricebil', 'priceext', 'pricemil', 'priceperm2', 'pricevnd']

*# Khởi tạo MinMaxScaler*

scaler = MinMaxScaler()

*# Chuẩn hóa dữ liệu*

df[columns\_to\_scale] = scaler.fit\_transform(df[columns\_to\_scale])

df = df.fillna(0)

*# Hiển thị kết quả*

print(df)

*# In dữ liệu đã chuẩn hóa*

print(df.head())

Kết quả:

area aream2 balconydirection bedroom bedroomcount direction \

0 0.290476 0.290476 Không rõ 3.0 3 phòng Không rõ

1 0.328571 0.328571 Không rõ 0.0 0 Không rõ

2 0.233333 0.233333 Không rõ 0.0 0 Tây - Bắc

6 0.709524 0.709524 Không rõ 0.0 0 Không rõ

7 0.314286 0.314286 Không rõ 3.0 3 phòng Không rõ

9 0.290476 0.290476 Đông - Nam 3.0 3 phòng Đông - Nam

10 0.538095 0.538095 Không rõ 0.0 0 Không rõ

11 0.423810 0.423810 Không rõ 0.0 0 Không rõ

13 0.985714 0.985714 Không rõ 0.0 0 Không rõ

16 1.000000 1.000000 Không rõ 0.0 0 Đông - Nam

17 0.000000 0.000000 Không rõ 3.0 3 phòng Tây - Bắc

19 0.123810 0.123810 Không rõ 3.0 3 phòng Không rõ

21 0.447619 0.447619 Không rõ 6.0 6 phòng Đông - Nam

Ví dụ 3: Clean tập dữ liệu ảnh cảm xúc khuôn mặt

1.5 Kết luận

**Kết luận về một số vấn đề cơ bản của Machine Learning (ML)**

**Tại sao lại có Machine Learning (ML)?**

* **Giải quyết các bài toán phức tạp**: Trước khi có ML, nhiều bài toán yêu cầu phải lập trình thủ công các quy tắc và logic phức tạp. ML xuất hiện như một giải pháp giúp máy tính tự động học từ dữ liệu mà không cần phải lập trình chi tiết mọi quy tắc. Các bài toán như nhận dạng hình ảnh, phân tích văn bản, hay dự đoán hành vi khách hàng có thể được giải quyết hiệu quả hơn bằng cách học từ dữ liệu thực tế.
* **Khả năng xử lý dữ liệu lớn (Big Data)**: ML có thể xử lý khối lượng dữ liệu lớn mà không cần sự can thiệp của con người, giúp phát hiện các mẫu và xu hướng trong dữ liệu mà con người khó có thể nhận ra.
* **Cải thiện hiệu suất theo thời gian**: Mô hình ML có thể tự cải thiện qua từng lần học, giúp tối ưu hóa các quy trình và dự đoán chính xác hơn theo thời gian.

**Mục đích của ML**

* **Tự động hóa các quyết định**: ML giúp tự động hóa các quy trình ra quyết định, đặc biệt là trong các lĩnh vực như tài chính, y tế, và marketing, nơi có rất nhiều dữ liệu và quy trình phức tạp.
* **Dự đoán và phân tích xu hướng**: Các mô hình ML được sử dụng để dự đoán các kết quả trong tương lai (ví dụ: dự báo nhu cầu khách hàng, dự đoán bệnh tật) và phân tích xu hướng từ dữ liệu quá khứ.
* **Cải thiện các dịch vụ và sản phẩm**: ML giúp các doanh nghiệp cải thiện chất lượng dịch vụ và sản phẩm thông qua việc phân tích hành vi người dùng, từ đó cá nhân hóa trải nghiệm và tối ưu hóa quy trình.

**Kỹ thuật cơ bản trong ML**

* **Học giám sát (Supervised Learning)**: Là kỹ thuật học mà trong đó mô hình học từ một bộ dữ liệu có nhãn (ví dụ: phân loại email là spam hay không spam, dự đoán giá nhà dựa trên các đặc trưng).
* **Học không giám sát (Unsupervised Learning)**: Trong học không giám sát, dữ liệu không có nhãn. Mục tiêu là phát hiện các mẫu, cấu trúc hoặc mối quan hệ trong dữ liệu (ví dụ: phân cụm khách hàng, giảm chiều dữ liệu).
* **Học bán giám sát (Semi-supervised Learning)**: Sử dụng cả dữ liệu có nhãn và không có nhãn để cải thiện hiệu quả mô hình, hữu ích khi việc thu thập nhãn là tốn kém.
* **Học tăng cường (Reinforcement Learning)**: Mô hình học từ các phản hồi của môi trường, tối ưu hóa hành động dựa trên thưởng và phạt (ví dụ: tối ưu hóa chiến lược chơi game, robot tự động).

**Deep Learning (Học sâu)**

* **Khái niệm**: Deep Learning là một nhánh con của Machine Learning, sử dụng các mạng nơ-ron nhân tạo với nhiều lớp (deep networks) để học các đặc trưng phức tạp từ dữ liệu. Các mạng nơ-ron này có khả năng học và trích xuất đặc trưng mà không cần sự can thiệp của con người.
* **Ưu điểm**:
  + **Khả năng tự học đặc trưng**: Deep Learning có thể tự động trích xuất đặc trưng mà không cần phải lập trình thủ công.
  + **Xử lý dữ liệu phức tạp**: Đặc biệt mạnh mẽ với các bài toán phức tạp như nhận diện hình ảnh, phân tích văn bản và giọng nói.
* **Nhược điểm**:
  + **Cần nhiều dữ liệu và tài nguyên tính toán**: Deep Learning yêu cầu lượng dữ liệu rất lớn và khả năng tính toán mạnh mẽ như GPU.
  + **Khó giải thích**: Các mô hình Deep Learning thường được coi là "hộp đen", khó để giải thích các quyết định mà chúng đưa ra.

**Làm sạch dữ liệu (Data Cleaning)**

* **Xử lý thiếu dữ liệu**: Dữ liệu thiếu (NaN) cần được xử lý để không ảnh hưởng đến độ chính xác của mô hình. Phương pháp xử lý bao gồm thay thế giá trị thiếu bằng giá trị trung bình, trung vị hoặc mô hình dự đoán.
* **Phát hiện và xử lý giá trị cực trị (Outliers)**: Nhiễu và giá trị bất thường có thể làm giảm hiệu quả của mô hình. Phương pháp xử lý bao gồm loại bỏ hoặc giảm ảnh hưởng của các giá trị cực trị.
* **Chuẩn hoá và mã hóa dữ liệu**: Dữ liệu phân loại cần được mã hóa lại (One-Hot Encoding, Label Encoding). Dữ liệu văn bản cần được vector hóa (TF-IDF, Word2Vec). Việc chuẩn hoá giúp giảm thiểu sự chênh lệch giữa các đặc trưng có đơn vị khác nhau.

**Chuẩn hoá dữ liệu (Data Normalization)**

* **Lý do chuẩn hoá**: Chuẩn hoá dữ liệu giúp các mô hình học máy hoạt động hiệu quả hơn, tránh trường hợp các đặc trưng có phạm vi giá trị lớn chiếm ưu thế trong quá trình huấn luyện.
* **Các phương pháp chuẩn hoá**:
  + **Min-Max Scaling**: Biến đổi giá trị các đặc trưng về một khoảng cố định, thường là [0, 1].
  + **Z-score Standardization**: Biến đổi giá trị các đặc trưng sao cho có giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1.
  + **Robust Scaling**: Phương pháp này sử dụng các thông số trung vị và khoảng IQR để chuẩn hoá, giúp giảm ảnh hưởng của các giá trị cực trị.

CHƯƠNG 2: DEEP LEARNING

2.1 Lịch sử phát triển của Deep learning

**Khởi đầu và nền tảng**

Trong những năm 1950, các nghiên cứu về trí tuệ nhân tạo (AI) bắt đầu hình thành, với Alan Turing đưa ra ý tưởng về trí thông minh máy tính qua bài viết nổi tiếng "Computing Machinery and Intelligence". Đồng thời, **Perceptron**, một mô hình mạng nơ-ron đơn giản do Frank Rosenblatt phát triển vào năm 1958, là bước đầu tiên hướng đến việc áp dụng mạng nơ-ron trong nhận diện mẫu, mặc dù chỉ có thể giải quyết các vấn đề rất cơ bản.

**Những thử thách và sự phát triển trong học sâu**

Bước sang những năm 1970 và 1980, nghiên cứu về mạng nơ-ron tiếp tục nhưng gặp phải nhiều hạn chế về tính toán và dữ liệu. Tuy nhiên, vào năm 1986, một đột phá quan trọng xảy ra khi **Geoffrey Hinton** cùng các cộng sự phát triển thuật toán **Lan truyền ngược (Backpropagation)**. Thuật toán này cho phép tối ưu hóa các trọng số trong mạng nơ-ron nhiều lớp, mở ra cơ hội mới cho việc xây dựng các mô hình học máy phức tạp hơn.

**Khởi đầu của Deep Learning**

Năm 2006 đánh dấu một mốc quan trọng trong lịch sử Deep Learning khi **Geoffrey Hinton** và các cộng sự giới thiệu **Deep Belief Networks (DBNs)**, khôi phục lại sự quan tâm vào mạng nơ-ron sâu. Các mô hình này có thể học được các đặc trưng phức tạp từ dữ liệu mà không cần nhãn, đặt nền tảng cho sự phát triển của Deep Learning trong những năm sau.

**Bùng nổ Deep Learning**

Năm 2012, một bước đột phá lớn diễn ra với **AlexNet** — mô hình mạng nơ-ron sâu (CNN) giành chiến thắng trong cuộc thi **ImageNet**. Sự thành công của AlexNet không chỉ làm nổi bật khả năng của Deep Learning trong nhận diện hình ảnh, mà còn đánh dấu sự khởi đầu của thời kỳ bùng nổ công nghệ Deep Learning. Các kỹ thuật như **Generative Adversarial Networks (GANs)** cũng được giới thiệu trong cùng thời gian, mở ra những khả năng mới trong việc tạo dữ liệu giả.

**Sự phát triển vượt bậc trong NLP và các ứng dụng AI**

Vào năm 2014, **GANs** (Generative Adversarial Networks) được **Ian Goodfellow** giới thiệu và nhanh chóng trở thành một trong những công cụ mạnh mẽ trong việc tạo ra dữ liệu giả như hình ảnh, video. Tiếp theo, sự xuất hiện của **ResNet** vào năm 2015 với các mạng nơ-ron sâu hơn đã giải quyết vấn đề "vanishing gradient", giúp các mô hình học sâu ngày càng phức tạp và hiệu quả hơn.

Từ năm 2017, **Transformers** đã thay đổi cách tiếp cận trong xử lý ngôn ngữ tự nhiên (NLP). Kiến trúc này, với cơ chế **chú ý (Attention)**, đã thúc đẩy các mô hình như **BERT** và **GPT**, giúp nâng cao khả năng của AI trong các nhiệm vụ như dịch máy, tạo văn bản và phân tích cảm xúc.

**Hiện tại và tương lai của Deep Learning**

Ngày nay, những mô hình như **GPT-3** với hàng trăm tỷ tham số đã chứng minh khả năng vượt trội trong các ứng dụng NLP. Đồng thời, các nghiên cứu về **AI tổng quát (AGI)** và **đạo đức AI** đang dần trở nên quan trọng. Deep Learning không chỉ tác động mạnh mẽ đến các lĩnh vực như y tế, tài chính, mà còn hứa hẹn mở ra những khả năng chưa từng có trong sáng tạo nội dung, tự động hóa và hiểu biết sâu sắc về thế giới.

2.2 Kiến thức mạng Neuron

**Mạng neuron là gì?**

* **Mô phỏng não người**: Mạng neuron được thiết kế để mô phỏng cách hoạt động của các neuron trong não bộ con người.
* **Tập hợp các nút (nodes)**: Mỗi nút (neuron) được kết nối với nhau qua các liên kết (connections), mỗi liên kết có trọng số (weight) để điều chỉnh mức độ ảnh hưởng giữa các nút.

**Thành phần chính của mạng neuron**

**Neuron**: Là đơn vị cơ bản xử lý thông tin, gồm:

* + **Input**: Dữ liệu đầu vào (x).
  + **Weights**: Trọng số www, điều chỉnh tầm quan trọng của từng đầu vào.
  + **Bias**: Giá trị cố định bbb, thêm linh hoạt cho quá trình học.
  + **Activation Function**: Hàm kích hoạt, quyết định đầu ra của neuron (ví dụ: sigmoid, ReLU).

Công thức:



**Layers (Các tầng)**:

* **Input Layer**: Nhận dữ liệu từ bên ngoài.
* **Hidden Layers**: Tầng ẩn thực hiện các phép tính phức tạp.
* **Output Layer**: Tầng đầu ra, trả về kết quả.

**Weights và Biases**:

* Trọng số www: Điều chỉnh tầm quan trọng của từng kết nối.
* Bias bbb: Dịch chuyển đầu ra của hàm kích hoạt, tránh trường hợp giá trị luôn bằng 0.

**Hàm kích hoạt (Activation Function)**:

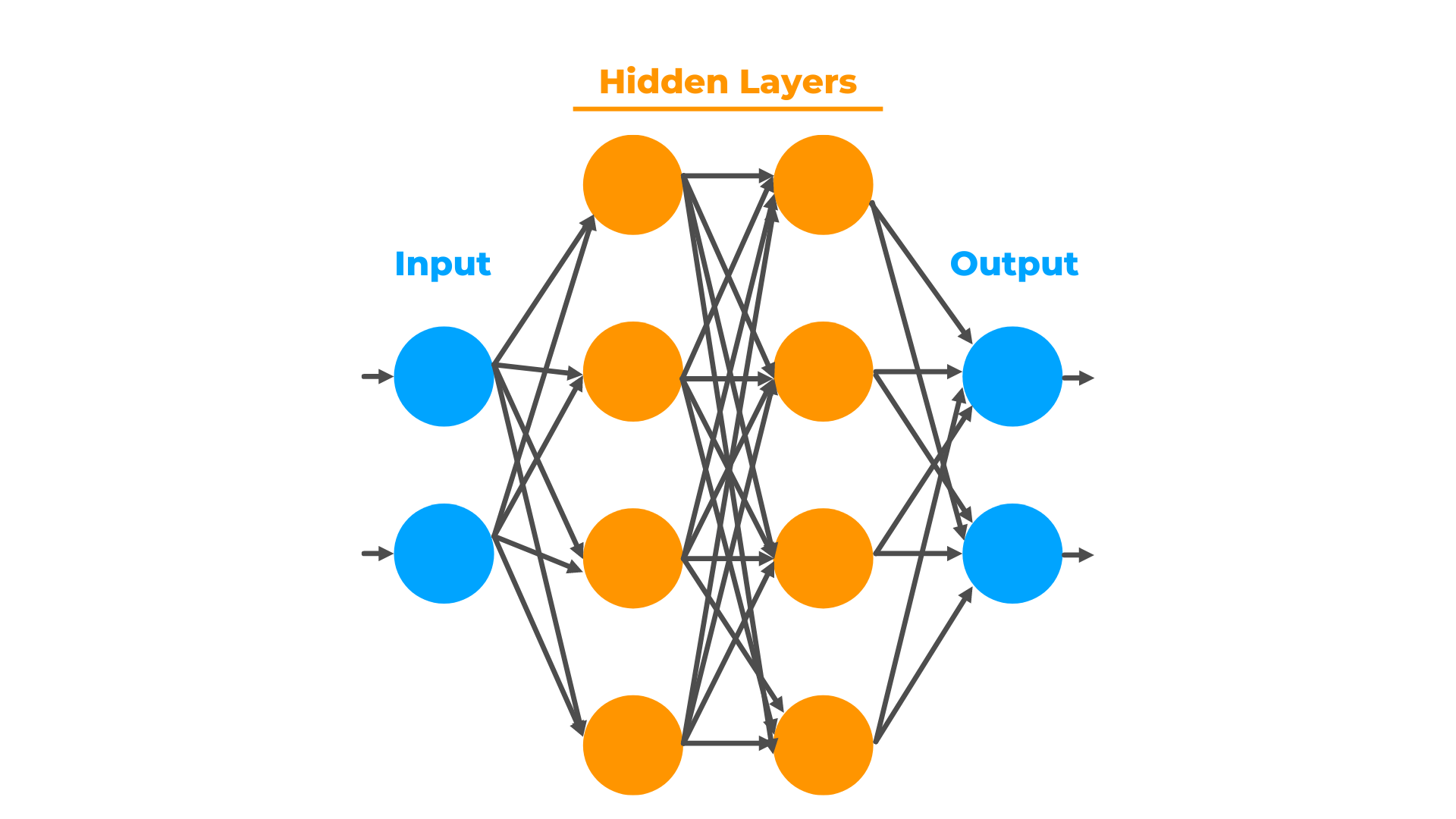
* **ReLU** (Rectified Linear Unit): f(x)=max⁡(0,x)f(x) = \max(0, x)f(x)=max(0,x).
* **Sigmoid**: Chuyển đầu ra về khoảng (0, 1).
* **Tanh**: Đưa đầu ra về khoảng (-1, 1).
* **Softmax**: Thường dùng cho bài toán phân loại đa lớp.

**Cách mạng neuron học**

* **Forward Propagation (Lan truyền tiến)**:
  + Tính toán kết quả đầu ra từ input qua các tầng.
* **Backward Propagation (Lan truyền ngược)**:
  + Dùng để điều chỉnh trọng số dựa trên lỗi (error) giữa đầu ra thực tế và dự đoán.
  + Sử dụng thuật toán **gradient descent**.

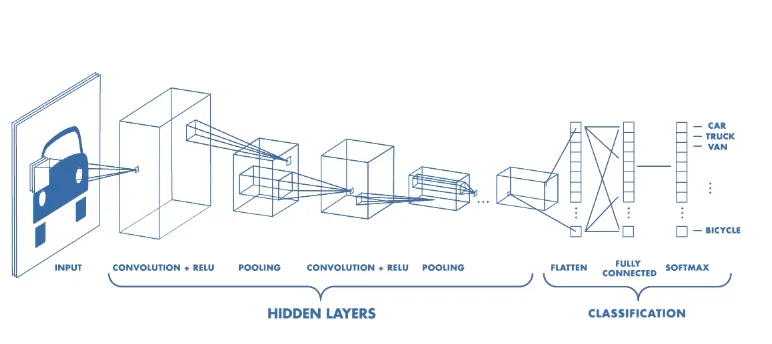
**Các loại mạng neuron phổ biến**

1. **Feedforward Neural Network**:
   * Luồng dữ liệu đi từ đầu vào đến đầu ra.
   * Thường dùng trong các bài toán hồi quy hoặc phân loại cơ bản.



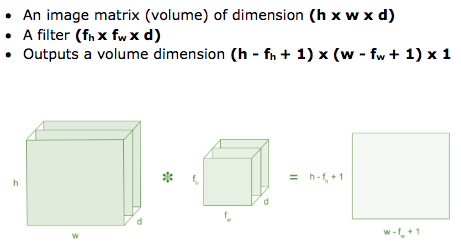
* **Tầng đầu vào (Input Layer):** Đây là tầng đầu tiên trong mạng nơ-ron truyền thẳng, nơi dữ liệu ban đầu được đưa vào hệ thống. Mỗi nơ-ron trong tầng này đại diện cho một đặc trưng duy nhất của tập dữ liệu đầu vào. Ví dụ, nếu chúng ta đang làm việc với dữ liệu hình ảnh, mỗi nơ-ron có thể tương ứng với một giá trị pixel. Tầng đầu vào không thực hiện bất kỳ phép tính nào mà chỉ đóng vai trò là điểm khởi đầu, chuyển thông tin sang các tầng tiếp theo để xử lý.
* **Tầng ẩn (Hidden Layer):** Đây là các tầng trung gian trong mạng, nơi thực hiện các phép tính và xử lý dữ liệu thực sự. Các tầng này rất quan trọng để mạng có thể học được các mẫu phức tạp và biểu diễn các đặc trưng của dữ liệu đầu vào. Một tầng ẩn bao gồm các nơ-ron áp dụng trọng số (weights) lên dữ liệu nhận được và đưa dữ liệu qua một hàm kích hoạt (activation function) để thêm tính phi tuyến. Số lượng tầng ẩn và số lượng nơ-ron trong mỗi tầng có thể ảnh hưởng lớn đến khả năng của mạng trong việc mô hình hóa các hàm phức tạp.
* **Tầng đầu ra (Output Layer):** Đây là tầng cuối cùng chịu trách nhiệm tạo ra dự đoán của mạng hoặc giải pháp cho vấn đề mà dữ liệu đầu vào đặt ra. Số lượng nơ-ron trong tầng này tương ứng với số lượng kết quả hoặc lớp (classes) có thể có trong một bài toán phân loại. Đầu ra được tạo ra thường ở dạng dễ hiểu với người dùng hoặc các hệ thống khác, thường thông qua các loại hàm kích hoạt khác nhau như **softmax** cho bài toán phân loại hoặc các hàm tuyến tính cho bài toán hồi quy.

1. **Convolutional Neural Network (CNN)**:
   * Dùng trong xử lý ảnh và video.
   * Sử dụng các bộ lọc (filters) để trích xuất đặc trưng từ dữ liệu.

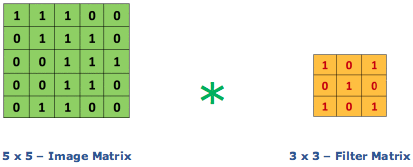


**Lớp tích chập - Convolution Layer**

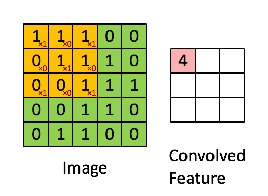
* Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vương nhỏ của dữ liệu đầu vào. Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân.



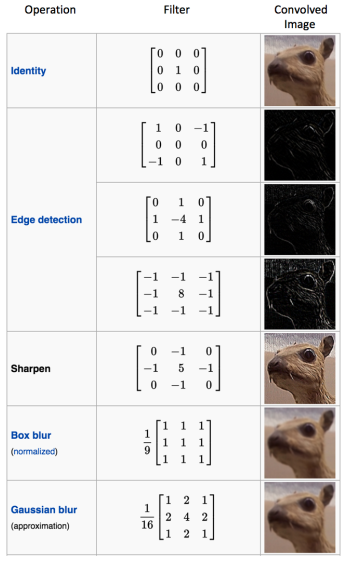
Xem xét 1 ma trận 5 x 5 có giá trị pixel là 0 và 1. Ma trận bộ lọc 3 x 3 như hình bên dưới.



Sau đó, lớp tích chập của ma trận hình ảnh 5 x 5 nhân với ma trận bộ lọc 3 x 3 gọi là 'Feature Map' như hình bên dưới.

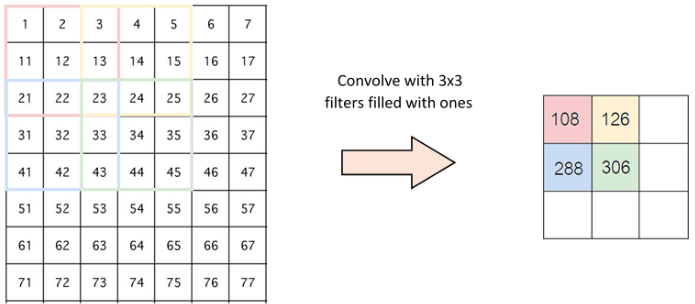


Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.



**Bước nhảy - Stride**

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.

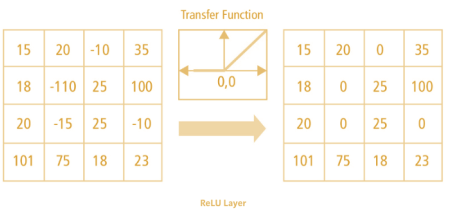


**Đường viền - Padding**

* Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:
* Chèn thêm các số 0 vào 4 đường biên của hình ảnh (padding).
* Cắt bớt hình ảnh tại những điểm không phù hợp với kernel.

**Hàm phi tuyến - ReLU**

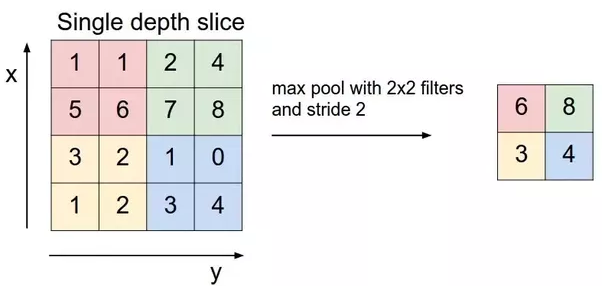
* ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là: ƒ (x) = max (0, x).
* Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.



* Có 1 số hàm phi tuyến khác như tanh, sigmoid cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

**Lớp gộp - Pooling Layer**

* Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:
* Max Pooling
* Average Pooling
* Sum Pooling
* Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling



Ví dụ 1: Áp dụng CNN vào nhận diện cảm xúc gương mặt

Link code model: [D:\HTTM\Tiểu luận\btl\_xu\_ly\_anh\code\_model\trainmodel.ipynb](file:///D:\HTTM\Tiểu%20luận\btl_xu_ly_anh\code_model\trainmodel.ipynb)

model = Sequential()

*# convolutional layers*

model.add(Conv2D(128, kernel\_size=(3,3), activation='relu', input\_shape=(48,48,1)))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.4))

model.add(Conv2D(256, kernel\_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.4))

model.add(Conv2D(512, kernel\_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.4))

model.add(Conv2D(512, kernel\_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.4))

model.add(Flatten())

*# fully connected layers*

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.4))

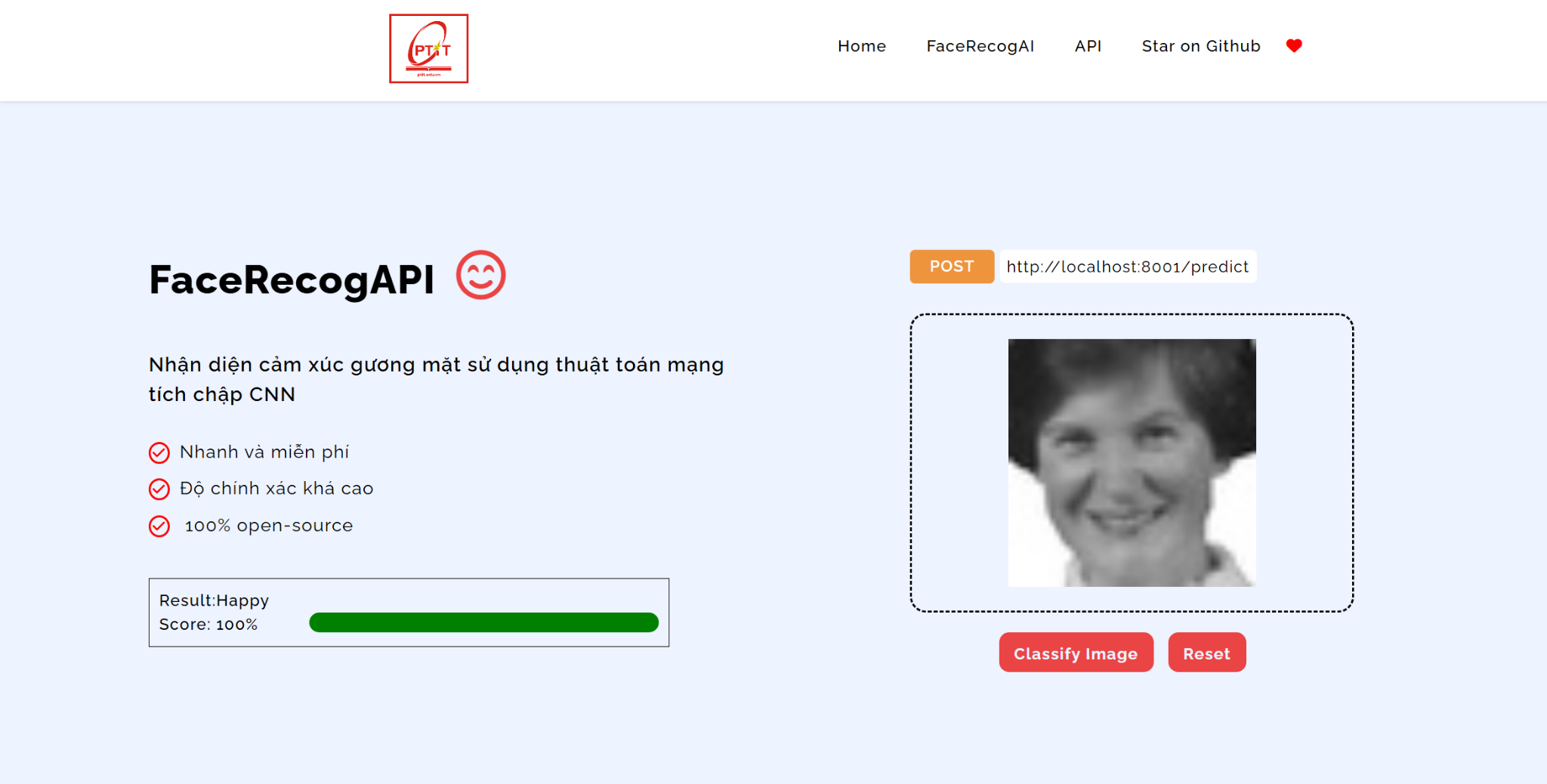
model.add(Dense(256, activation='relu'))

model.add(Dropout(0.3))

*# output layer*

model.add(Dense(7, activation='softmax'))

Giao diện



Ví dụ 2: Nhận diện ảnh chữ số viết tay

Link Code: [D:\HTTM\Tiểu luận\code\_demo\_cnn\mnist\_model.ipynb](file:///D:\HTTM\Tiểu%20luận\code_demo_cnn\mnist_model.ipynb)

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import load\_model

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageOps

*# Bước 1: Tải và xử lý dữ liệu MNIST*

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

*# Chuẩn hóa dữ liệu (Rescale các giá trị pixel từ 0-255 về 0-1)*

x\_train = x\_train.astype('float32') / 255

x\_test = x\_test.astype('float32') / 255

*# Thay đổi hình dạng của dữ liệu từ (28, 28) thành (28, 28, 1)*

x\_train = np.expand\_dims(x\_train, -1)

x\_test = np.expand\_dims(x\_test, -1)

*# Chuyển đổi nhãn thành dạng one-hot encoding*

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

*# Bước 2: Xây dựng mô hình Keras với 5 layers*

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

Kết quả:

Epoch 1/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **27s** 31ms/step - accuracy: 0.9019 - loss: 0.3272 - val\_accuracy: 0.9828 - val\_loss: 0.0640

Epoch 2/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **20s** 27ms/step - accuracy: 0.9855 - loss: 0.0464 - val\_accuracy: 0.9839 - val\_loss: 0.0511

Epoch 3/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **20s** 26ms/step - accuracy: 0.9903 - loss: 0.0299 - val\_accuracy: 0.9862 - val\_loss: 0.0436

Epoch 4/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **19s** 25ms/step - accuracy: 0.9929 - loss: 0.0223 - val\_accuracy: 0.9872 - val\_loss: 0.0442

Epoch 5/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **18s** 24ms/step - accuracy: 0.9952 - loss: 0.0144 - val\_accuracy: 0.9887 - val\_loss: 0.0390

Epoch 6/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **21s** 28ms/step - accuracy: 0.9952 - loss: 0.0132 - val\_accuracy: 0.9854 - val\_loss: 0.0521

Epoch 7/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **20s** 27ms/step - accuracy: 0.9969 - loss: 0.0092 - val\_accuracy: 0.9881 - val\_loss: 0.0477

Epoch 8/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **21s** 28ms/step - accuracy: 0.9974 - loss: 0.0069 - val\_accuracy: 0.9857 - val\_loss: 0.0612

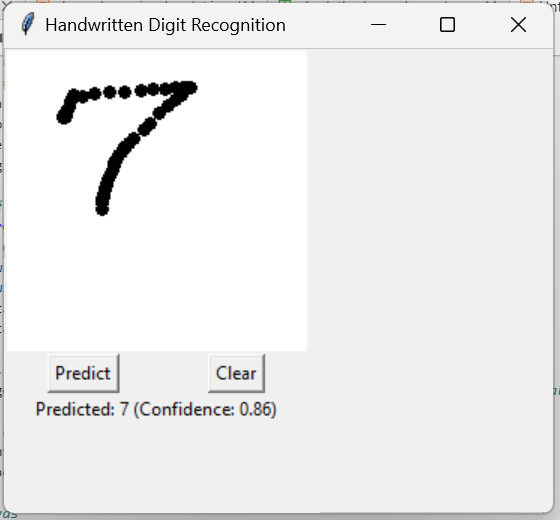
Epoch 9/10

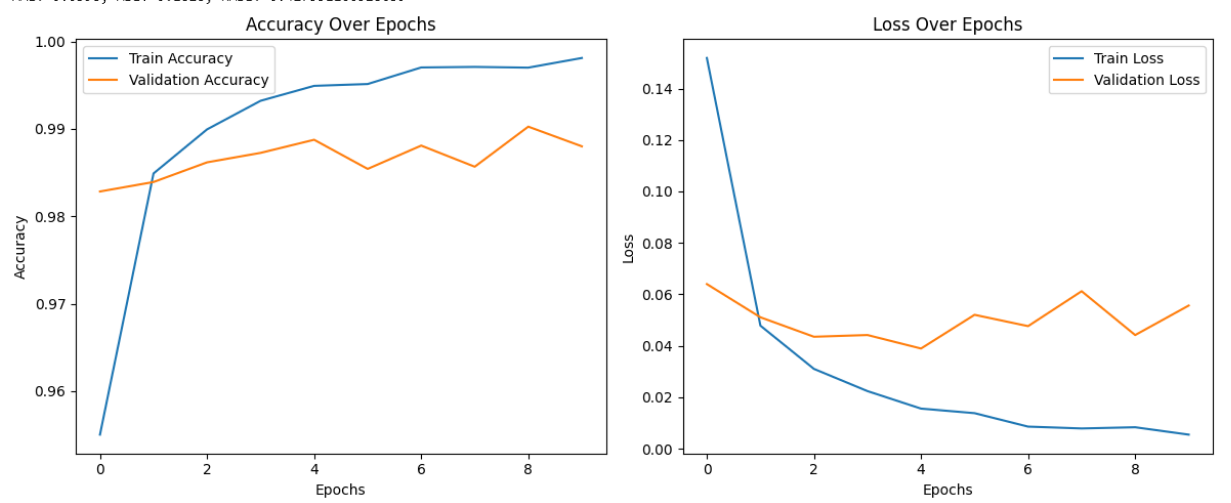
**750/750** ━━━━━━━━━━━━━━━━━━━━ **22s** 29ms/step - accuracy: 0.9964 - loss: 0.0091 - val\_accuracy: 0.9902 - val\_loss: 0.0442

Epoch 10/10

**750/750** ━━━━━━━━━━━━━━━━━━━━ **21s** 28ms/step - accuracy: 0.9986 - loss: 0.0044 - val\_accuracy: 0.9880 - val\_loss: 0.0557

Giao diện:





Nhận xét:

* Biểu đồ Accuracy và Loss trên hai tập validate và train khá sát nhau cho thấy model học khá tốt

Ví dụ 3: Ví dụ nhận diện ảnh chó mèo dựa trên tập data kaggle:

Link code: [D:\HTTM\Tiểu luận\code\_demo\_cat\_dog\code\_train\_model.ipynb](file:///D:\HTTM\Tiểu%20luận\code_demo_cat_dog\code_train_model.ipynb)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

*# Huấn luyện mô hình*

history = model.fit(

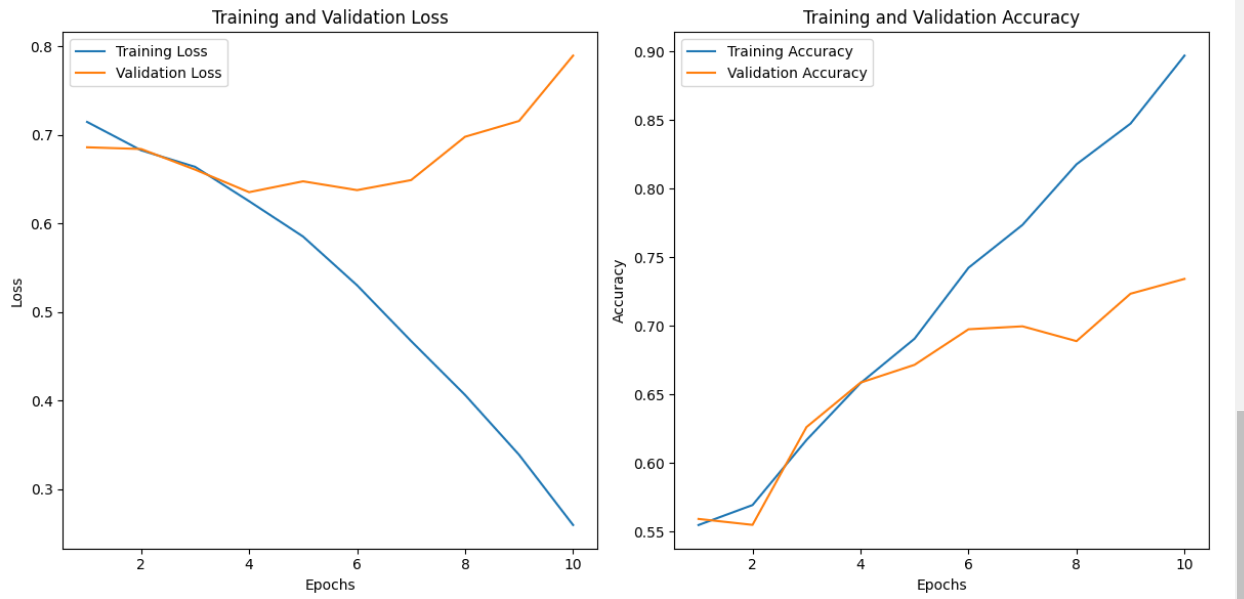
train\_generator,

epochs=10,

validation\_data=val\_generator

)

Kết quả:



Found 1856 images belonging to 2 classes.

Found 463 images belonging to 2 classes.

Epoch 1/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **28s** 420ms/step - accuracy: 0.5457 - loss: 0.7708 - val\_accuracy: 0.5594 - val\_loss: 0.6860

Epoch 2/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **26s** 448ms/step - accuracy: 0.5631 - loss: 0.6846 - val\_accuracy: 0.5551 - val\_loss: 0.6841

Epoch 3/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **23s** 402ms/step - accuracy: 0.5943 - loss: 0.6711 - val\_accuracy: 0.6263 - val\_loss: 0.6608

Epoch 4/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **27s** 469ms/step - accuracy: 0.6420 - loss: 0.6290 - val\_accuracy: 0.6587 - val\_loss: 0.6353

Epoch 5/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **29s** 492ms/step - accuracy: 0.7054 - loss: 0.5747 - val\_accuracy: 0.6717 - val\_loss: 0.6476

Epoch 6/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **27s** 469ms/step - accuracy: 0.7572 - loss: 0.5175 - val\_accuracy: 0.6976 - val\_loss: 0.6376

Epoch 7/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **26s** 450ms/step - accuracy: 0.7903 - loss: 0.4446 - val\_accuracy: 0.6998 - val\_loss: 0.6489

Epoch 8/10

**58/58** ━━━━━━━━━━━━━━━━━━━━ **26s** 442ms/step - accuracy: 0.8199 - loss: 0.3997 - val\_accuracy: 0.6890 - val\_loss: 0.6979

Epoch 9/10

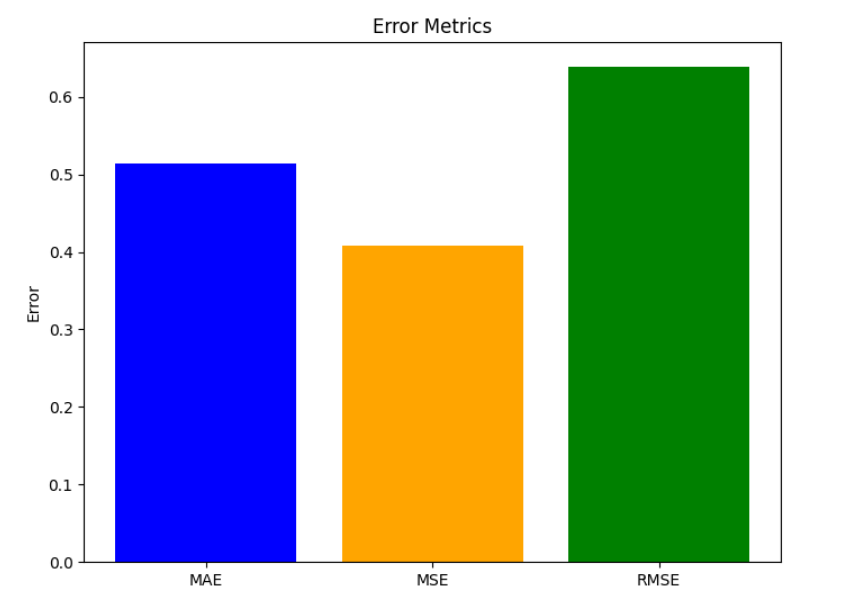
**58/58** ━━━━━━━━━━━━━━━━━━━━ **28s** 480ms/step - accuracy: 0.8462 - loss: 0.3467 - val\_accuracy: 0.7235 - val\_loss: 0.7157

Epoch 10/10

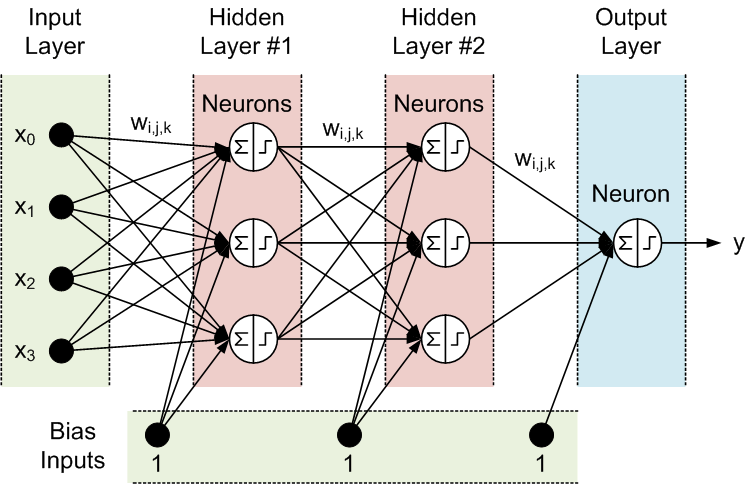
**58/58** ━━━━━━━━━━━━━━━━━━━━ **26s** 452ms/step - accuracy: 0.9030 - loss: 0.2544 - val\_accuracy: 0.7343 - val\_loss: 0.7896

* Biểu đồ Accuracy và Loss trên hai tập validate và train khá sát nhau cho thấy model học khá tốt

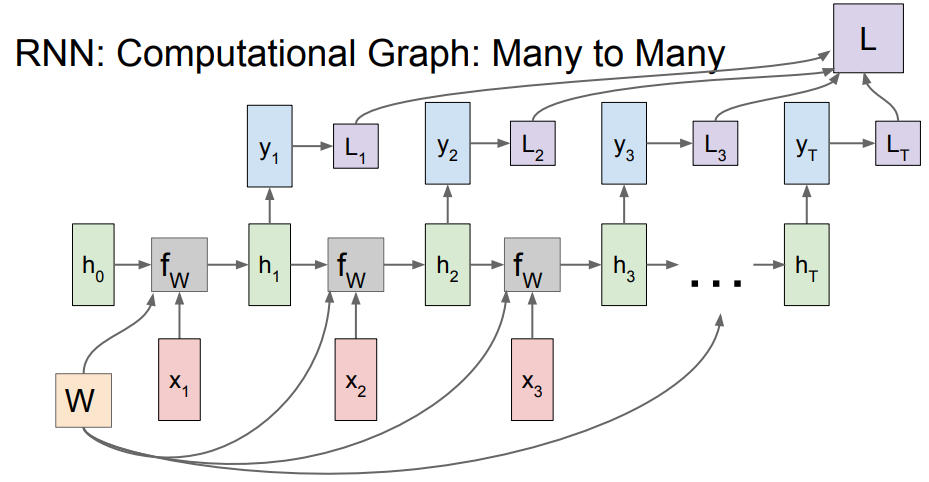
Biểu đồ so sánh giá trị MAE, MSE, RMSE:



1. **Recurrent Neural Network (RNN)**:
   * Dùng cho dữ liệu tuần tự, như xử lý ngôn ngữ tự nhiên.
   * Có kết nối ngược để lưu trạng thái trước đó.



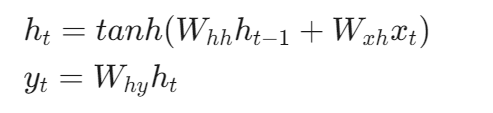
Như đã biết thì Neural Network bao gồm 3 phần chính là Input layer, Hidden layer và Output layer, có thể thấy là đầu vào và đầu ra của mạng neuron này là độc lập với nhau. Như vậy mô hình này không phù hợp với những bài toán dạng chuỗi như mô tả, hoàn thành câu, ... vì những dự đoán tiếp theo như từ tiếp theo phụ thuộc vào vị trí của nó trong câu và những từ đằng trước nó.  
Và như vậy RNN ra đời với ý tưởng chính là sử dụng một bộ nhớ để lưu lại thông tin từ từ những bước tính toán xử lý trước để dựa vào nó có thể đưa ra dự đoán chính xác nhất cho bước dự đoán hiện tại. Nếu các bạn vẫn chưa hiểu gì thì hãy cùng xem mô hình mạng RNN sau và cùng phân tích để hiểu rõ hơn:



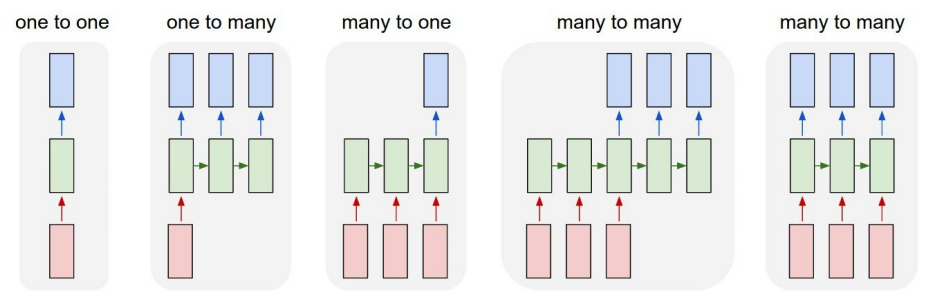
Nếu như mạng Neural Network chỉ là input layer x*x* đi qua hidden layer h*h* và cho ra output layer y*y* với ***full connected*** giữa các layer thì trong RNN, các input xt*xt*​ sẽ được kết hợp với hidden layer ht−1*ht*−1​ bằng hàm fW*fW*​ để tính toán ra hidden layer ht*ht*​ hiện tại và output yt*yt*​ sẽ được tính ra từ ht*ht*​, W*W* là tập các trọng số và nó được ở tất cả các cụm, các L1,L2,...,Lt*L*1​,*L*2​,...,*Lt*​ là các hàm mất mát sẽ được giải thích sau. Như vậy kết quả từ các quá trình tính toán trước đã được "nhớ" bằng cách kết hợp thêm ht−1*ht*−1​ tính ra ht*ht*​ để tăng độ chính xác cho những dự đoán ở hiện tại. Cụ thể quá trình tính toán được viết dưới dạng toán như sau:



Hàm fw​ chúng ta sẽ xử dụng hàm **tanh**, công thức trên sẽ trở thành :

  
  
  
Đến đây có 3 thứ mới xuất hiện: Wxh,Whh,Why*Wxh*​,*Whh*​,*Why*​. Đối với mạng NN chỉ sử dụng một ma trận trọng số W duy nhất thì với RNN nó sử dụng 3 ma trận trọng số cho 2 quá trình tính toán: Whh*Whh*​ kết hợp với "bộ nhớ trước" ht−1*ht*−1​ và Wxh *Wxh*​ kết hợp với xt*xt*​ để tính ra "bộ nhớ của bước hiện tại" ht*ht*​ từ đó kết hợp với Why*Wh*​*y* để tính ra yt*yt*​.

Ngoài mô hình Many to Many như ở trên thì RNN còn rất nhiều dạng khác như sau:



Ví dụ 1:

Ví dụ 1: Xây dựng model dự đoán điểm thi

Code model:

*# Chuẩn hóa dữ liệu*

scaler = MinMaxScaler()

X\_mon1 = scaler.fit\_transform(mon1\_df[["TP1", "TP2", "TP3"]])

*# Không cần chuẩn hóa y nếu giá trị CK không cần thiết*

y\_mon1 = mon1\_df[["CK"]]

*# Chia dữ liệu thành tập huấn luyện và kiểm tra*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_mon1, y\_mon1, test\_size=0.2, random\_state=42)

*# Định dạng lại dữ liệu cho RNN (thêm trục thời gian)*

X\_train = np.expand\_dims(X\_train, axis=1) *# (samples, timesteps, features)*

X\_test = np.expand\_dims(X\_test, axis=1)

*# Xây dựng mô hình RNN với 7 lớp*

model = Sequential([

SimpleRNN(128, activation='relu', input\_shape=(1, 3), return\_sequences=True),

Dropout(0.2), *# Dropout để giảm overfitting*

SimpleRNN(128, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(128, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(128, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(128, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(128, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(128, activation='relu'), *# Lớp cuối cùng không trả về chuỗi*

Dense(1) *# Đầu ra là một giá trị*

])

Kết quả chạy code:

Epoch 1/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **15s** 40ms/step - loss: 18.1617 - mae: 3.6225 - val\_loss: 3.0583 - val\_mae: 1.4745

Epoch 2/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 11ms/step - loss: 3.4923 - mae: 1.5418 - val\_loss: 3.5047 - val\_mae: 1.5388

Epoch 3/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 7ms/step - loss: 3.1111 - mae: 1.4541 - val\_loss: 3.2856 - val\_mae: 1.4959

Epoch 4/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 3.1066 - mae: 1.4551 - val\_loss: 2.9174 - val\_mae: 1.4352

Epoch 5/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 11ms/step - loss: 2.8250 - mae: 1.4160 - val\_loss: 2.5447 - val\_mae: 1.3808

Epoch 6/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 2.9747 - mae: 1.4547 - val\_loss: 3.2306 - val\_mae: 1.4861

Epoch 7/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 2.7273 - mae: 1.3847 - val\_loss: 2.7132 - val\_mae: 1.4004

Epoch 8/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 2.7797 - mae: 1.4014 - val\_loss: 2.5789 - val\_mae: 1.3855

Epoch 9/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 2.7702 - mae: 1.4000 - val\_loss: 2.7369 - val\_mae: 1.4069

Epoch 10/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 2.8048 - mae: 1.4023 - val\_loss: 2.7824 - val\_mae: 1.4099

Epoch 11/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 10ms/step - loss: 2.7107 - mae: 1.3859 - val\_loss: 2.6139 - val\_mae: 1.3881

Epoch 12/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 2.9089 - mae: 1.4511 - val\_loss: 2.8124 - val\_mae: 1.4155

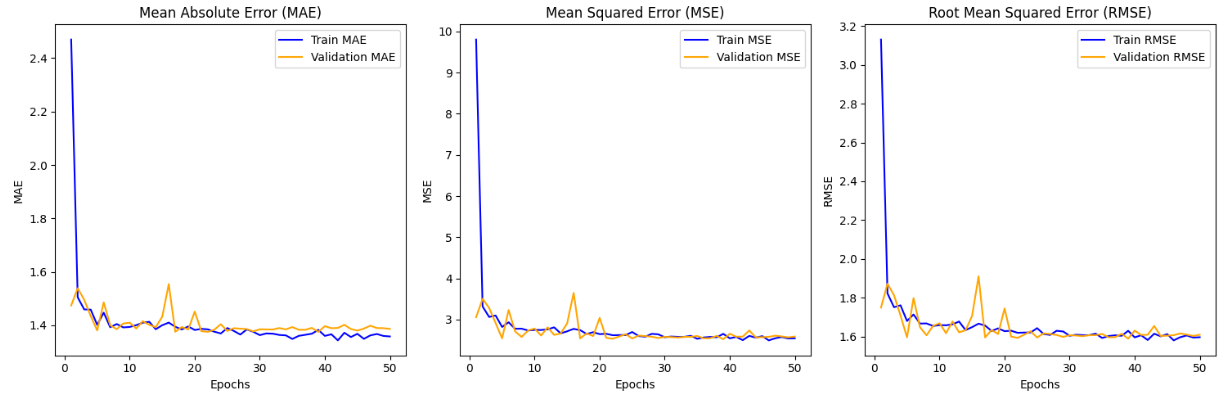
Epoch 13/50

...

**13/13** ━━━━━━━━━━━━━━━━━━━━ **0s** 3ms/step - loss: 2.6820 - mae: 1.4008

Test Loss (MSE): 2.5885772705078125, Test MAE: 1.3866147994995117

**13/13** ━━━━━━━━━━━━━━━━━━━━ **3s** 106ms/step



Giao diện:



Ví dụ 2: Xây dựng model dự đoán giá nhà

Code:

*# === Chuẩn hóa và chia dữ liệu ===*

X = data[["Area", "Rooms", "Location", "Age"]].values

y = data["Price"].values.reshape(-1, 1)

*# Chuẩn hóa dữ liệu*

scaler\_X = MinMaxScaler()

scaler\_y = MinMaxScaler()

X = scaler\_X.fit\_transform(X)

y = scaler\_y.fit\_transform(y)

*# Chia dữ liệu thành tập train/test*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Định dạng lại cho RNN*

X\_train = np.expand\_dims(X\_train, axis=1) *# (samples, timesteps, features)*

X\_test = np.expand\_dims(X\_test, axis=1)

*# === Xây dựng mô hình RNN với 7 tầng ===*

model = Sequential([

SimpleRNN(64, activation='relu', input\_shape=(1, X.shape[1]), return\_sequences=True),

Dropout(0.2),

SimpleRNN(64, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(32, activation='relu'),

Dense(32, activation='relu'),

Dense(16, activation='relu'),

Dense(8, activation='relu'),

Dense(1) *# Output là giá nhà*

])

*# Biên dịch mô hình*

model.compile(optimizer='adam', loss='mse', metrics=['mae', 'mse'])

*# Huấn luyện mô hình*

history = model.fit(X\_train, y\_train,

validation\_data=(X\_test, y\_test),

epochs=50,

batch\_size=32,

verbose=1)

Kết quả:

Epoch 1/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **16s** 88ms/step - loss: 0.1630 - mae: 0.3268 - mse: 0.1630 - val\_loss: 0.0299 - val\_mae: 0.1459 - val\_mse: 0.0299

Epoch 2/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0318 - mae: 0.1495 - mse: 0.0318 - val\_loss: 0.0105 - val\_mae: 0.0838 - val\_mse: 0.0105

Epoch 3/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0162 - mae: 0.1020 - mse: 0.0162 - val\_loss: 0.0043 - val\_mae: 0.0542 - val\_mse: 0.0043

Epoch 4/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 12ms/step - loss: 0.0103 - mae: 0.0801 - mse: 0.0103 - val\_loss: 0.0029 - val\_mae: 0.0452 - val\_mse: 0.0029

Epoch 5/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0098 - mae: 0.0769 - mse: 0.0098 - val\_loss: 0.0020 - val\_mae: 0.0364 - val\_mse: 0.0020

Epoch 6/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0089 - mae: 0.0762 - mse: 0.0089 - val\_loss: 0.0049 - val\_mae: 0.0604 - val\_mse: 0.0049

Epoch 7/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0074 - mae: 0.0671 - mse: 0.0074 - val\_loss: 0.0042 - val\_mae: 0.0565 - val\_mse: 0.0042

Epoch 8/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0056 - mae: 0.0581 - mse: 0.0056 - val\_loss: 0.0045 - val\_mae: 0.0602 - val\_mse: 0.0045

Epoch 9/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0059 - mae: 0.0594 - mse: 0.0059 - val\_loss: 0.0035 - val\_mae: 0.0516 - val\_mse: 0.0035

Epoch 10/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0048 - mae: 0.0544 - mse: 0.0048 - val\_loss: 0.0048 - val\_mae: 0.0614 - val\_mse: 0.0048

Epoch 11/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0052 - mae: 0.0556 - mse: 0.0052 - val\_loss: 0.0051 - val\_mae: 0.0634 - val\_mse: 0.0051

Epoch 12/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0047 - mae: 0.0530 - mse: 0.0047 - val\_loss: 0.0033 - val\_mae: 0.0503 - val\_mse: 0.0033

Epoch 13/50

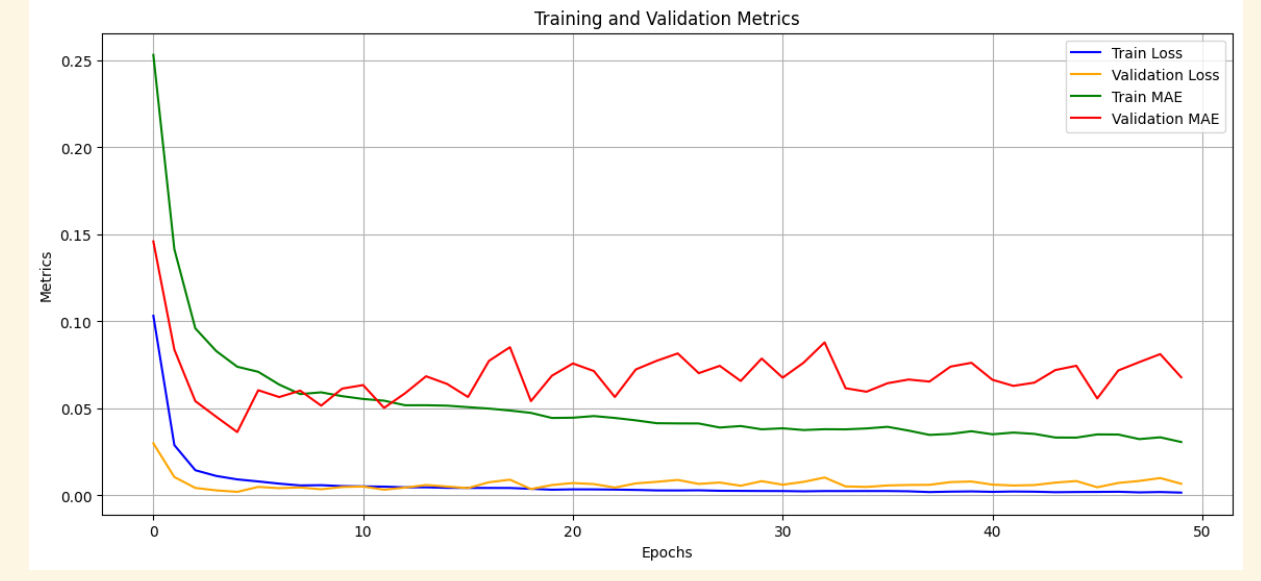
...

Epoch 49/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 12ms/step - loss: 0.0020 - mae: 0.0340 - mse: 0.0020 - val\_loss: 0.0100 - val\_mae: 0.0812 - val\_mse: 0.0100

Epoch 50/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0015 - mae: 0.0299 - mse: 0.0015 - val\_loss: 0.0067 - val\_mae: 0.0679 - val\_mse: 0.0067



Ví dụ 3: Xây dựng model dự đoán chỉ số BMI

*# Chia dữ liệu thành X và y*

X = df\_bmi[['Height', 'Weight', 'BMI']].values

y = df\_bmi['Category'].values

*# Chia dữ liệu thành tập huấn luyện và kiểm tra*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Định dạng lại dữ liệu cho RNN (tạo thêm trục thời gian)*

X\_train = np.expand\_dims(X\_train, axis=1) *# (samples, timesteps, features)*

X\_test = np.expand\_dims(X\_test, axis=1)

*# Chuyển đổi nhãn thành dạng one-hot encoding*

y\_train = to\_categorical(y\_train, num\_classes=4)

y\_test = to\_categorical(y\_test, num\_classes=4)

*# Xây dựng mô hình RNN với 7 lớp*

model = Sequential([

SimpleRNN(128, activation='relu', input\_shape=(1, 3), return\_sequences=True),

SimpleRNN(128, activation='relu', return\_sequences=True),

SimpleRNN(128, activation='relu', return\_sequences=True),

SimpleRNN(128, activation='relu', return\_sequences=True),

SimpleRNN(128, activation='relu', return\_sequences=True),

SimpleRNN(128, activation='relu', return\_sequences=True),

SimpleRNN(128, activation='relu') *# Lớp cuối cùng không cần return\_sequences*

])

*# Lớp Dense cuối cùng để phân loại*

model.add(Dense(4, activation='softmax')) *# 4 lớp phân loại (Underweight, Normal, Overweight, Obesity)*

*# Biên dịch mô hình, thêm 'mae' vào chỉ số*

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy', 'mae'])

*# Huấn luyện mô hình*

history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), batch\_size=32)

Kết quả:

Epoch 1/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **14s** 78ms/step - accuracy: 0.3612 - loss: 1.2595 - mae: 0.3472 - val\_accuracy: 0.6550 - val\_loss: 0.7006 - val\_mae: 0.2110

Epoch 2/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.7907 - loss: 0.5291 - mae: 0.1713 - val\_accuracy: 0.9200 - val\_loss: 0.1803 - val\_mae: 0.0708

Epoch 3/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - accuracy: 0.8926 - loss: 0.2321 - mae: 0.0774 - val\_accuracy: 0.9450 - val\_loss: 0.1507 - val\_mae: 0.0544

Epoch 4/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.9299 - loss: 0.1780 - mae: 0.0566 - val\_accuracy: 0.9250 - val\_loss: 0.1489 - val\_mae: 0.0516

Epoch 5/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 0.8991 - loss: 0.2702 - mae: 0.0648 - val\_accuracy: 0.9400 - val\_loss: 0.1704 - val\_mae: 0.0561

Epoch 6/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.9443 - loss: 0.1634 - mae: 0.0556 - val\_accuracy: 0.9550 - val\_loss: 0.1026 - val\_mae: 0.0378

Epoch 7/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.9474 - loss: 0.1428 - mae: 0.0445 - val\_accuracy: 0.9650 - val\_loss: 0.1025 - val\_mae: 0.0369

Epoch 8/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 0.9609 - loss: 0.1156 - mae: 0.0369 - val\_accuracy: 0.9900 - val\_loss: 0.0625 - val\_mae: 0.0265

Epoch 9/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 0.9647 - loss: 0.0955 - mae: 0.0306 - val\_accuracy: 0.9500 - val\_loss: 0.0948 - val\_mae: 0.0316

Epoch 10/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 0.9481 - loss: 0.1343 - mae: 0.0378 - val\_accuracy: 0.9800 - val\_loss: 0.0666 - val\_mae: 0.0260

Epoch 11/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 0.9692 - loss: 0.0944 - mae: 0.0315 - val\_accuracy: 0.9600 - val\_loss: 0.0794 - val\_mae: 0.0273

Epoch 12/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.9478 - loss: 0.1789 - mae: 0.0391 - val\_accuracy: 0.9800 - val\_loss: 0.0679 - val\_mae: 0.0259

Epoch 13/50

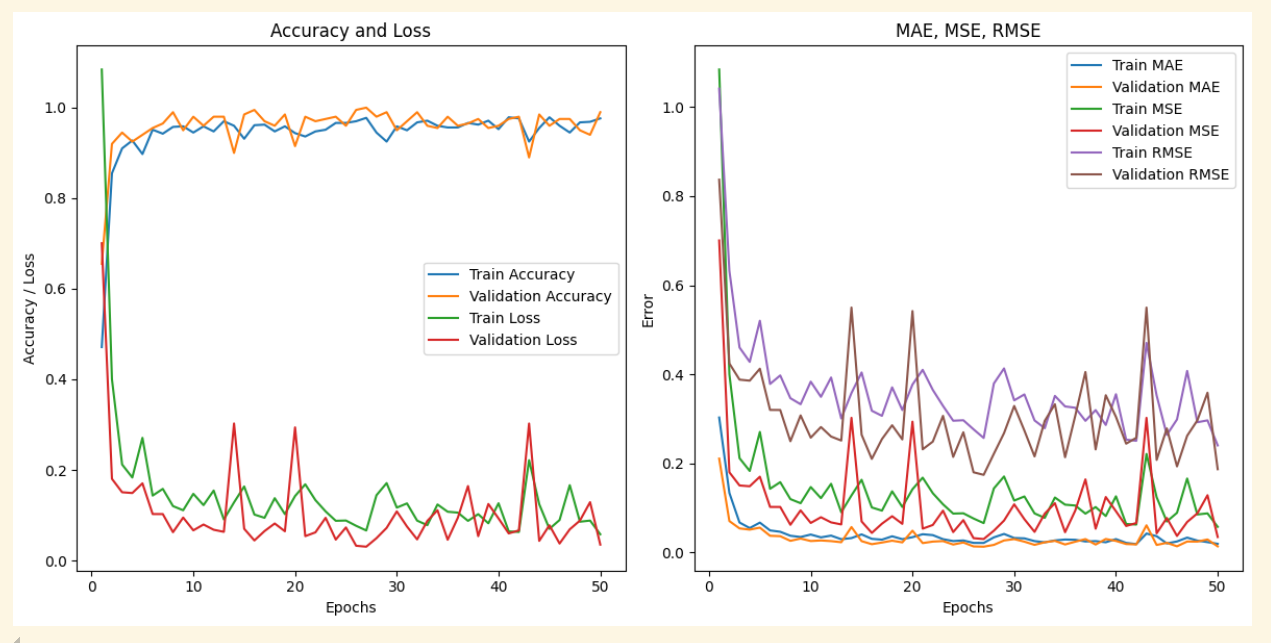
...

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - accuracy: 0.9704 - loss: 0.0858 - mae: 0.0234 - val\_accuracy: 0.9400 - val\_loss: 0.1288 - val\_mae: 0.0295

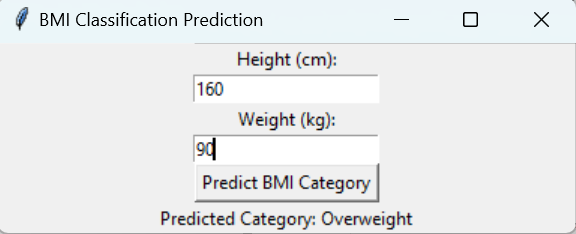
Epoch 50/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - accuracy: 0.9700 - loss: 0.0594 - mae: 0.0194 - val\_accuracy: 0.9900 - val\_loss: 0.0350 - val\_mae: 0.0140

**7/7** ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - accuracy: 0.9842 - loss: 0.0418 - mae: 0.0163



Giao diện demo:



1. **LSTM (Long Short-Term Memory)**:
   * Biến thể của RNN, giúp ghi nhớ thông tin trong thời gian dài.

Ví dụ 1: Xây dựng model dự đoán cho tập điểm thi:

*# Xây dựng mô hình LSTM 7 layers*

model = Sequential([

LSTM(100, activation='relu', return\_sequences=True, input\_shape=(1, 3)),

Dropout(0.2), *# Dropout để tránh overfitting*

LSTM(100, activation='relu', return\_sequences=True),

Dropout(0.2),

LSTM(100, activation='relu', return\_sequences=True),

Dropout(0.2),

LSTM(100, activation='relu'),

Dense(50, activation='relu'), *# Dense layer trung gian*

Dense(1) *# Lớp đầu ra dự đoán 1 giá trị*

])

*# Biên dịch mô hình*

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

*# Huấn luyện mô hình*

model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), batch\_size=32)

Kết quả chạy code:

Epoch 1/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **13s** 42ms/step - loss: 0.2108 - mae: 0.4027 - val\_loss: 0.0448 - val\_mae: 0.1816

Epoch 2/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 8ms/step - loss: 0.0437 - mae: 0.1775 - val\_loss: 0.0423 - val\_mae: 0.1769

Epoch 3/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 0.0414 - mae: 0.1732 - val\_loss: 0.0402 - val\_mae: 0.1731

Epoch 4/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0400 - mae: 0.1707 - val\_loss: 0.0394 - val\_mae: 0.1719

Epoch 5/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 0.0395 - mae: 0.1711 - val\_loss: 0.0406 - val\_mae: 0.1740

Epoch 6/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 0.0395 - mae: 0.1704 - val\_loss: 0.0391 - val\_mae: 0.1716

Epoch 7/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - loss: 0.0386 - mae: 0.1681 - val\_loss: 0.0391 - val\_mae: 0.1716

Epoch 8/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 12ms/step - loss: 0.0396 - mae: 0.1709 - val\_loss: 0.0395 - val\_mae: 0.1717

Epoch 9/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step - loss: 0.0388 - mae: 0.1673 - val\_loss: 0.0391 - val\_mae: 0.1717

Epoch 10/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0380 - mae: 0.1667 - val\_loss: 0.0392 - val\_mae: 0.1721

Epoch 11/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0404 - mae: 0.1740 - val\_loss: 0.0392 - val\_mae: 0.1720

Epoch 12/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0386 - mae: 0.1683 - val\_loss: 0.0392 - val\_mae: 0.1720

Epoch 13/50

...

Epoch 49/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 11ms/step - loss: 0.0393 - mae: 0.1703 - val\_loss: 0.0402 - val\_mae: 0.1739

Epoch 50/50

**50/50** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0393 - mae: 0.1685 - val\_loss: 0.0396 - val\_mae: 0.1721

Ví dụ 2: Xây dựng model dự đoán chỉ số BMI

Code:

*# Chuẩn hóa dữ liệu*

scaler = MinMaxScaler()

df\_bmi[['Height', 'Weight', 'BMI']] = scaler.fit\_transform(df\_bmi[['Height', 'Weight', 'BMI']])

*# Chuyển đổi phân loại thành dạng nhãn số*

category\_map = {'Underweight': 0, 'Normal weight': 1, 'Overweight': 2, 'Obesity': 3}

df\_bmi['Category'] = df\_bmi['Category'].map(category\_map)

*# Chia dữ liệu thành X và y*

X = df\_bmi[['Height', 'Weight', 'BMI']].values

y = df\_bmi['Category'].values

*# Chia dữ liệu thành tập huấn luyện và kiểm tra*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Định dạng lại dữ liệu cho LSTM (thêm trục thời gian)*

X\_train = np.expand\_dims(X\_train, axis=1) *# (samples, timesteps, features)*

X\_test = np.expand\_dims(X\_test, axis=1)

*# Chuyển đổi nhãn thành dạng one-hot encoding*

y\_train = to\_categorical(y\_train, num\_classes=4)

y\_test = to\_categorical(y\_test, num\_classes=4)

*# Xây dựng mô hình LSTM với 7 lớp*

lstm\_model = Sequential([

LSTM(128, activation='relu', input\_shape=(1, 3), return\_sequences=True),

LSTM(128, activation='relu', return\_sequences=True),

LSTM(128, activation='relu', return\_sequences=True),

LSTM(128, activation='relu', return\_sequences=True),

LSTM(128, activation='relu', return\_sequences=True),

LSTM(128, activation='relu', return\_sequences=True),

LSTM(128, activation='relu') *# Lớp cuối cùng không cần return\_sequences*

])

*# Lớp Dense cuối cùng để phân loại*

lstm\_model.add(Dense(4, activation='softmax')) *# 4 lớp phân loại (Underweight, Normal, Overweight, Obesity)*

*# Biên dịch mô hình*

lstm\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy', 'mae'])

*# Huấn luyện mô hình*

lstm\_history = lstm\_model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), batch\_size=32)

Kết quả chạy code:

Epoch 1/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **22s** 129ms/step - accuracy: 0.3318 - loss: 1.3834 - mae: 0.3746 - val\_accuracy: 0.4100 - val\_loss: 1.3734 - val\_mae: 0.3733

Epoch 2/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 22ms/step - accuracy: 0.4011 - loss: 1.3700 - mae: 0.3728 - val\_accuracy: 0.4100 - val\_loss: 1.3500 - val\_mae: 0.3695

Epoch 3/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 24ms/step - accuracy: 0.4509 - loss: 1.3307 - mae: 0.3657 - val\_accuracy: 0.4100 - val\_loss: 1.3085 - val\_mae: 0.3542

Epoch 4/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 29ms/step - accuracy: 0.3922 - loss: 1.3076 - mae: 0.3497 - val\_accuracy: 0.4100 - val\_loss: 1.1048 - val\_mae: 0.2607

Epoch 5/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.4428 - loss: 0.9797 - mae: 0.2493 - val\_accuracy: 0.4100 - val\_loss: 0.9139 - val\_mae: 0.2487

Epoch 6/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.4427 - loss: 0.8542 - mae: 0.2314 - val\_accuracy: 0.5600 - val\_loss: 0.7884 - val\_mae: 0.2238

Epoch 7/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 27ms/step - accuracy: 0.6282 - loss: 0.7052 - mae: 0.1993 - val\_accuracy: 0.7450 - val\_loss: 0.6620 - val\_mae: 0.1876

Epoch 8/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 26ms/step - accuracy: 0.7785 - loss: 0.6412 - mae: 0.1780 - val\_accuracy: 0.7900 - val\_loss: 0.6548 - val\_mae: 0.1895

Epoch 9/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 24ms/step - accuracy: 0.7559 - loss: 0.6759 - mae: 0.1913 - val\_accuracy: 0.8150 - val\_loss: 0.4959 - val\_mae: 0.1602

Epoch 10/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 23ms/step - accuracy: 0.7986 - loss: 0.4378 - mae: 0.1440 - val\_accuracy: 0.8600 - val\_loss: 0.2878 - val\_mae: 0.0869

Epoch 11/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 25ms/step - accuracy: 0.8969 - loss: 0.2755 - mae: 0.0771 - val\_accuracy: 0.9200 - val\_loss: 0.2390 - val\_mae: 0.0748

Epoch 12/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 28ms/step - accuracy: 0.9233 - loss: 0.1994 - mae: 0.0638 - val\_accuracy: 0.9700 - val\_loss: 0.0934 - val\_mae: 0.0355

Epoch 13/50

...

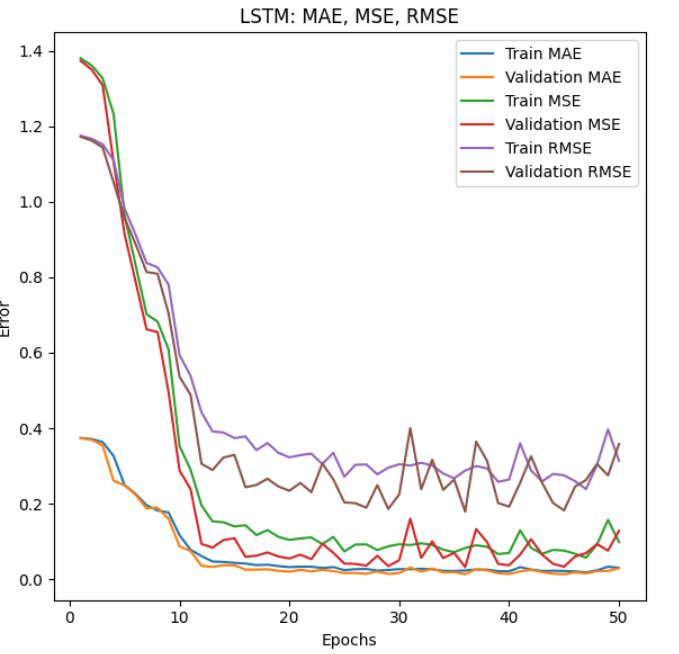
**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 21ms/step - accuracy: 0.9506 - loss: 0.1276 - mae: 0.0309 - val\_accuracy: 0.9650 - val\_loss: 0.0756 - val\_mae: 0.0220

Epoch 50/50

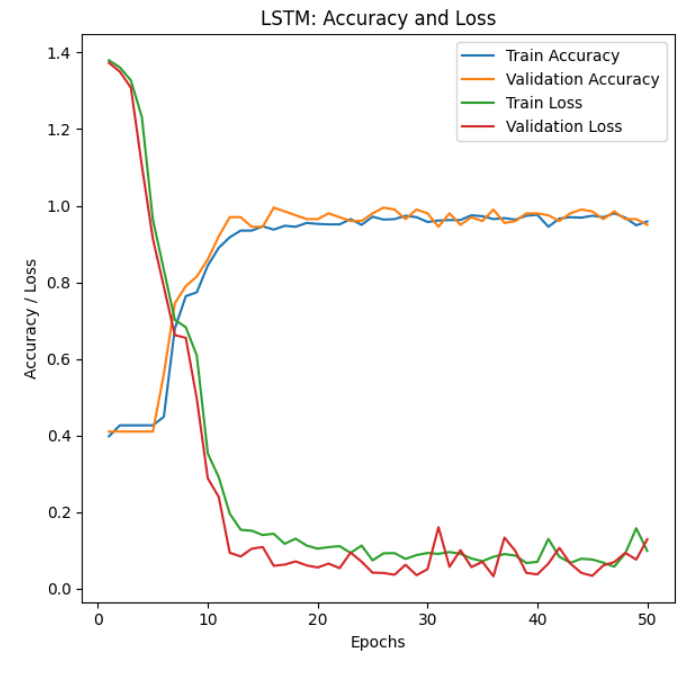
**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 20ms/step - accuracy: 0.9467 - loss: 0.1109 - mae: 0.0325 - val\_accuracy: 0.9500 - val\_loss: 0.1285 - val\_mae: 0.0292

**7/7** ━━━━━━━━━━━━━━━━━━━━ **0s** 8ms/step - accuracy: 0.9368 - loss: 0.1678 - mae: 0.0348

Biểu đồ độ đo MAE, MSE, RMSE:



Biểu đồ Accuracy và Loss:



Ví dụ 3: Xây dựng model dự đoán giá nhà:

Code:

*# Chia dữ liệu thành tập train/test*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Định dạng lại cho LSTM*

X\_train = np.expand\_dims(X\_train, axis=1) *# (samples, timesteps, features)*

X\_test = np.expand\_dims(X\_test, axis=1)

*# === Xây dựng mô hình LSTM với 7 tầng ===*

model = Sequential([

LSTM(64, activation='relu', input\_shape=(1, X.shape[1]), return\_sequences=True),

Dropout(0.2),

LSTM(64, activation='relu', return\_sequences=True),

Dropout(0.2),

LSTM(32, activation='relu', return\_sequences=True),

Dropout(0.2),

LSTM(32, activation='relu'),

Dense(32, activation='relu'),

Dense(16, activation='relu'),

Dense(8, activation='relu'),

Dense(1) *# Output là giá nhà*

])

*# Biên dịch mô hình*

model.compile(optimizer='adam', loss='mse', metrics=['mae', 'mse'])

*# Huấn luyện mô hình*

history = model.fit(X\_train, y\_train,

validation\_data=(X\_test, y\_test),

epochs=50,

batch\_size=32,

verbose=1)

*# Lưu mô hình*

model.save("./house\_price\_lstm\_model.keras")

Kết quả chạy code:  
Epoch 1/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **17s** 86ms/step - loss: 0.2790 - mae: 0.4670 - mse: 0.2790 - val\_loss: 0.1745 - val\_mae: 0.3496 - val\_mse: 0.1745

Epoch 2/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.1748 - mae: 0.3432 - mse: 0.1748 - val\_loss: 0.0758 - val\_mae: 0.2247 - val\_mse: 0.0758

Epoch 3/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - loss: 0.0724 - mae: 0.2245 - mse: 0.0724 - val\_loss: 0.0589 - val\_mae: 0.2046 - val\_mse: 0.0589

Epoch 4/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - loss: 0.0550 - mae: 0.2011 - mse: 0.0550 - val\_loss: 0.0441 - val\_mae: 0.1766 - val\_mse: 0.0441

Epoch 5/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0369 - mae: 0.1591 - mse: 0.0369 - val\_loss: 0.0152 - val\_mae: 0.0978 - val\_mse: 0.0152

Epoch 6/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - loss: 0.0187 - mae: 0.1111 - mse: 0.0187 - val\_loss: 0.0103 - val\_mae: 0.0823 - val\_mse: 0.0103

Epoch 7/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0135 - mae: 0.0936 - mse: 0.0135 - val\_loss: 0.0077 - val\_mae: 0.0691 - val\_mse: 0.0077

Epoch 8/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 9ms/step - loss: 0.0096 - mae: 0.0799 - mse: 0.0096 - val\_loss: 0.0049 - val\_mae: 0.0458 - val\_mse: 0.0049

Epoch 9/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0089 - mae: 0.0746 - mse: 0.0089 - val\_loss: 0.0065 - val\_mae: 0.0671 - val\_mse: 0.0065

Epoch 10/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0074 - mae: 0.0676 - mse: 0.0074 - val\_loss: 0.0031 - val\_mae: 0.0350 - val\_mse: 0.0031

Epoch 11/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0051 - mae: 0.0562 - mse: 0.0051 - val\_loss: 0.0024 - val\_mae: 0.0329 - val\_mse: 0.0024

Epoch 12/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0049 - mae: 0.0551 - mse: 0.0049 - val\_loss: 0.0019 - val\_mae: 0.0286 - val\_mse: 0.0019

Epoch 13/50

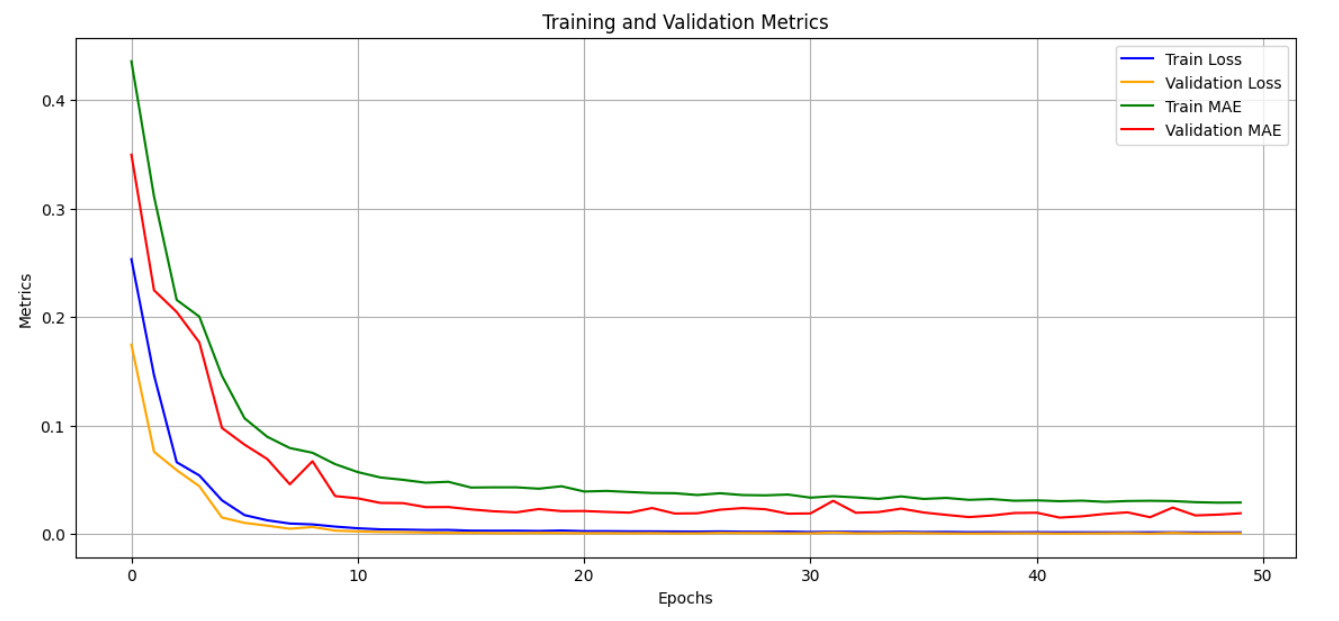
...

Epoch 49/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0014 - mae: 0.0294 - mse: 0.0014 - val\_loss: 6.5513e-04 - val\_mae: 0.0178 - val\_mse: 6.5513e-04

Epoch 50/50

**25/25** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0014 - mae: 0.0281 - mse: 0.0014 - val\_loss: 7.4316e-04 - val\_mae: 0.0191 - val\_mse: 7.4316e-04



2.8 Kết luận:  
Hình vẽ về độ đo MAE, MSE, RMSE của cả 3 CNN, LSTM, RNN 7 layer đối với model dự đoán điểm thi

+ Nhận xét:

**CNN**

* **Điểm mạnh**:
  + Xử lý dữ liệu không gian rất hiệu quả (đặc biệt là hình ảnh).
  + Khả năng tự động học đặc trưng qua các tầng tích chập (*convolution*).
  + Tính toán nhanh và dễ mở rộng mô hình.
* **Điểm yếu**:
  + Không phù hợp cho dữ liệu tuần tự, chuỗi thời gian.
  + Yêu cầu định dạng dữ liệu đầu vào đặc biệt (hình ảnh hoặc tín hiệu).
* **Khi nào sử dụng**:
  + Phân loại, nhận diện đối tượng từ hình ảnh hoặc tín hiệu 2D/3D.

**RNN**

* **Điểm mạnh**:
  + Phù hợp với các bài toán tuần tự ngắn.
  + Ít tốn tài nguyên tính toán hơn so với LSTM.
* **Điểm yếu**:
  + Hiệu suất giảm sút với chuỗi dữ liệu dài do hiện tượng *vanishing gradient*.
  + Không thể học hiệu quả các mối quan hệ dài hạn trong dữ liệu.
* **Khi nào sử dụng**:
  + Bài toán đơn giản với chuỗi dữ liệu ngắn (ví dụ: phân tích cảm xúc từ đoạn văn bản ngắn).

**LSTM**

* **Điểm mạnh**:
  + Hiệu quả trong việc học và lưu trữ thông tin dài hạn.
  + Giải quyết được vấn đề vanishing gradient nhờ kiến trúc với các cổng nhớ (*forget*, *input*, *output*).
* **Điểm yếu**:
  + Đòi hỏi tài nguyên tính toán nhiều hơn so với RNN.
  + Huấn luyện lâu hơn.
* **Khi nào sử dụng**:
  + Bài toán cần lưu giữ mối quan hệ dài hạn trong dữ liệu (dự báo chuỗi thời gian dài, dịch máy, nhận dạng giọng nói).

CHƯƠNG 3: DỮ LIỆU TRONG DỰ ĐOÁN VÀ XỬ LÝ ẢNH

3.1 Dữ liệu ảnh, dữ liệu điểm

3.1.1 Dữ liệu ảnh

**Biểu diễn dữ liệu ảnh**

Dữ liệu ảnh thường được biểu diễn dưới dạng các tensor (mảng nhiều chiều) tùy thuộc vào định dạng và màu sắc của ảnh.

**a. Dạng ma trận (Grayscale Image)**

* Ảnh đen trắng được biểu diễn dưới dạng ma trận 2D.
* Kích thước: (height, width)
* Mỗi phần tử trong ma trận là một giá trị pixel, thường từ 0 (đen) đến 255 (trắng).

**b. Dạng tensor (RGB Image)**

* Ảnh màu được biểu diễn dưới dạng tensor 3D.
* Kích thước: (height, width, channels)
  + channels = 3 cho các kênh màu đỏ (Red), xanh lá (Green), và xanh dương (Blue).

**c. Các định dạng khác**

* **Ảnh RGBA**: Bao gồm thêm kênh alpha (độ trong suốt), kích thước là (height, width, 4).
* **Batch ảnh**: Khi xử lý nhiều ảnh cùng lúc, dữ liệu được biểu diễn dưới dạng tensor 4D, kích thước (batch\_size, height, width, channels).

**Chuẩn hóa dữ liệu ảnh**

Chuẩn hóa giúp giảm thiểu sai lệch trong dữ liệu và cải thiện hiệu quả của mô hình.

**a. Chia giá trị pixel cho 255**

* Thông thường, giá trị pixel nằm trong khoảng [0, 255].
* Chuẩn hóa bằng cách chia cho 255 để giá trị nằm trong [0, 1]

**b.Trừ trung bình và chia độ lệch chuẩn**

* Phương pháp này đặt giá trị pixel về dạng chuẩn hóa với trung bình bằng 0 và độ lệch chuẩn bằng 1.
  + Trung bình mean và độ lệch chuẩn std được tính từ tập dữ liệu.

**c. Chuẩn hóa theo miền giá trị [-1, 1]**

* Một số kiến trúc mạng (như ResNet, MobileNet) yêu cầu giá trị đầu vào trong khoảng [-1, 1].
* Chuyển đổi: normalized\_image = (image / 127.5) – 1

**Tăng cường dữ liệu (Data Augmentation)**

Trong nhiều trường hợp, trước khi chuẩn hóa, dữ liệu ảnh thường được tăng cường để cải thiện hiệu suất và khả năng tổng quát hóa của mô hình:

* **Các phương pháp tăng cường**:
  + Xoay (rotation), lật (flipping), dịch chuyển (translation).
  + Thay đổi độ sáng, độ tương phản.
  + Cắt (cropping) và thay đổi kích thước (resizing).
* Thư viện phổ biến:
  + TensorFlow/Keras (ImageDataGenerator)
  + PyTorch (torchvision.transforms)
  + Albumentations

**Chú ý khi chuẩn hóa**

* **Consistency**: Phải đảm bảo cách chuẩn hóa giống nhau giữa dữ liệu huấn luyện, kiểm tra và dự đoán.
* **Reverse Normalization**: Nếu cần hiển thị hoặc lưu ảnh, cần khôi phục giá trị pixel về dạng ban đầu.

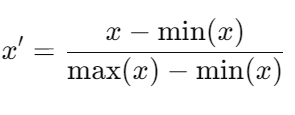
3.1.2 Dữ liệu điểm

**Chuẩn hóa dữ liệu điểm**

Việc chuẩn hóa rất quan trọng để đảm bảo rằng tất cả các cột dữ liệu có giá trị trong cùng một phạm vi, tránh cột có giá trị lớn lấn át các cột khác.

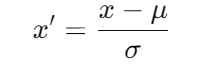
**a. Min-Max Scaling (Phổ biến nhất)**

* Đưa dữ liệu về khoảng [0, 1] hoặc một phạm vi tùy ý như [0, 10].
* Công thức



**b. Standardization (Z-Score Scaling)**

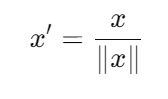
* Đưa dữ liệu về trung bình 0 và độ lệch chuẩn 1



* + μ: trung bình
  + σ: độ lệch chuẩn

**c. Normalization (L2 Norm)**

* Dùng để chuẩn hóa độ lớn của dữ liệu, hữu ích trong một số bài toán cụ thể như tính toán độ tương đồng:



3.2 Áp dụng cho tập dữ liệu ảnh:

Chuẩn hoá đưa giá trị các pixel về trong khoảng từ [0,1]:

3.3 Áp dụng cho tập dữ liệu điểm:

Link code: [D:\HTTM\Tiểu luận\code\_chuan\_hoa\chuan\_hoa\_diem\_thi.ipynb](file:///D:\HTTM\Tiểu%20luận\code_chuan_hoa\chuan_hoa_diem_thi.ipynb)

Chuẩn hoá data dùng công thức min-max

*# Chuẩn hoá data -> [0,1]*

cols\_to\_normalize = ["assignment\_1", "assignment\_2", "project", "final\_score"]

df\_normalized = finnal\_data.copy()

df\_normalized[cols\_to\_normalize] = (finnal\_data[cols\_to\_normalize] - finnal\_data[cols\_to\_normalize].min()) / (

finnal\_data[cols\_to\_normalize].max() - finnal\_data[cols\_to\_normalize].min()

)

*# Kết quả*

print(df\_normalized)

Kết quả thu được sau khi chuẩn hoá:

student\_id assignment\_1 assignment\_2 project final\_score

0 b1 0.900 0.777778 0.631579 0.600

1 b2 0.900 0.777778 0.631579 0.600

2 b3 1.000 1.000000 0.736842 0.725

3 b4 0.900 0.777778 0.631579 0.600

4 b5 1.000 0.888889 0.736842 0.750

.. ... ... ... ... ...

305 31\_151 0.825 0.861111 0.868421 0.800

306 31\_152 0.350 0.361111 0.631579 0.650

307 31\_153 0.725 0.722222 0.631579 0.650

308 31\_154 0.775 0.388889 0.789474 0.750

309 31\_155 0.000 0.000000 0.000000 0.000

3.4 Kết luận

**Dữ liệu ảnh**:

* **Biểu diễn**:  
  Dữ liệu ảnh được biểu diễn dưới dạng tensor, với các chiều thể hiện chiều cao, chiều rộng, và số kênh màu (grayscale: 2D; RGB/RGBA: 3D). Khi làm việc với nhiều ảnh cùng lúc, dữ liệu được biểu diễn thành tensor 4D với batch size.
* **Chuẩn hóa**:
  + Giá trị pixel thường nằm trong khoảng [0, 255], cần được chuẩn hóa về [0, 1] bằng cách chia cho 255 hoặc về [-1, 1] khi cần.
  + Chuẩn hóa giúp giảm thiểu sai lệch và tăng tốc độ hội tụ của mô hình, đặc biệt trong học sâu.
* **Tăng cường dữ liệu**:  
  Các kỹ thuật như xoay, lật, cắt ảnh, thay đổi độ sáng, và dịch chuyển giúp cải thiện khả năng tổng quát hóa và giảm overfitting.

**Dữ liệu điểm**:

* **Biểu diễn**:  
  Dữ liệu điểm thường là dữ liệu số, được biểu diễn dưới dạng mảng (array), ma trận, hoặc bảng (dataframe). Các cột dữ liệu đại diện cho các đặc trưng (ví dụ: bài tập, bài kiểm tra, thi giữa kỳ) và các dòng đại diện cho các mẫu dữ liệu.
* **Chuẩn hóa**:
  + **Min-Max Scaling**: Chuẩn hóa dữ liệu về khoảng [0, 1] để đảm bảo tính đồng nhất giữa các đặc trưng.
  + **Standardization (Z-score)**: Đưa dữ liệu về trung bình 0 và độ lệch chuẩn 1, phù hợp với các mô hình nhạy cảm với tỷ lệ.
  + **Trọng số**: Với dữ liệu điểm có trọng số, cần tính tổng điểm dựa trên tỷ lệ phần trăm đóng góp của từng cột vào tổng điểm.

**So sánh chung**:

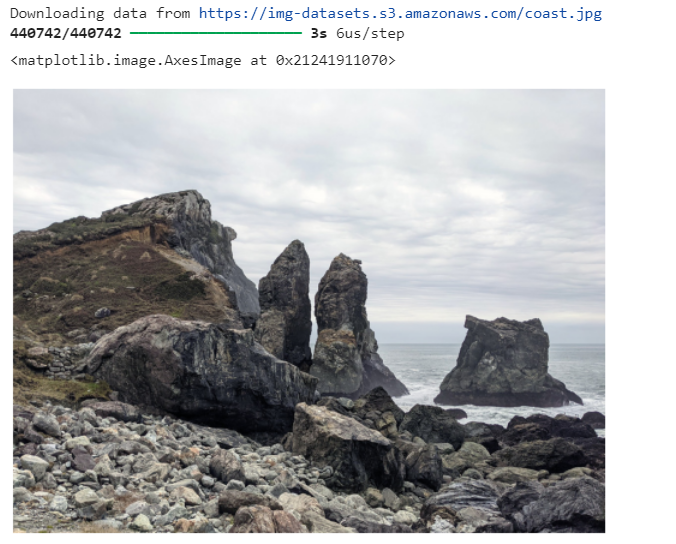
* **Mục tiêu của chuẩn hóa**:  
  Cả dữ liệu ảnh và dữ liệu điểm đều cần được chuẩn hóa để cải thiện hiệu quả xử lý của mô hình, giúp các đặc trưng có phạm vi giá trị nhất quán, giảm tác động của các đặc trưng có giá trị lớn hơn.
* **Độ phức tạp**:  
  Dữ liệu ảnh phức tạp hơn do kích thước lớn (đặc biệt với ảnh độ phân giải cao) và có nhiều kênh thông tin. Trong khi đó, dữ liệu điểm thường nhỏ hơn và dễ xử lý.
* **Ứng dụng**:
  + Dữ liệu ảnh thường được sử dụng trong các bài toán như phân loại ảnh, nhận dạng đối tượng, hoặc phát hiện bất thường.
  + Dữ liệu điểm phù hợp với các bài toán dự đoán, phân loại (ví dụ: dự đoán điểm số, xếp loại).

**CHƯƠNG 4: DEEP LEARNING TRONG XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

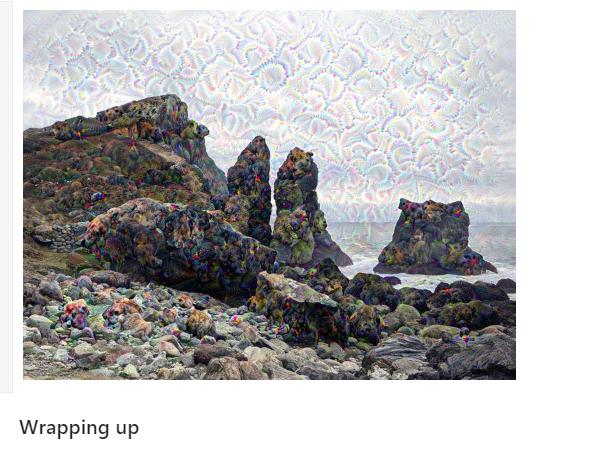
**4.1 Ví dụ**

4.1.1 Text generation

4.1.2 DeepDream



Ảnh sau khi train



4.1.3 Neural style transfer

Ngoài DeepDream, một phát triển quan trọng khác trong việc chỉnh sửa hình ảnh dựa trên deep learning là **neural style transfer** (chuyển phong cách thần kinh), được giới thiệu bởi Leon Gatys và các cộng sự vào mùa hè năm 2015.

Thuật toán **neural style transfer** đã trải qua nhiều cải tiến và xuất hiện nhiều biến thể kể từ khi được giới thiệu lần đầu, đồng thời đã được tích hợp vào nhiều ứng dụng chỉnh sửa ảnh trên smartphone.

Code:

from tensorflow import keras

base\_image\_path = keras.utils.get\_file(

"sf.jpg", origin="https://img-datasets.s3.amazonaws.com/sf.jpg")

style\_reference\_image\_path = keras.utils.get\_file(

"starry\_night.jpg", origin="https://img-datasets.s3.amazonaws.com/starry\_night.jpg")

original\_width, original\_height = keras.utils.load\_img(base\_image\_path).size

img\_height = 400

img\_width = round(original\_width \* img\_height / original\_height)

- Tải hai hình ảnh (ảnh gốc và ảnh phong cách).

- Xác định kích thước mới cho ảnh, với chiều cao cố định là 400 pixels, đồng thời đảm bảo rằng tỷ lệ khung hình của ảnh vẫn giữ nguyên. Điều này giúp chuẩn bị dữ liệu hình ảnh để xử lý trong Neural Style Transfer.

import numpy as np

**def** preprocess\_image(image\_path):

img = keras.utils.load\_img(

image\_path, target\_size=(img\_height, img\_width))

img = keras.utils.img\_to\_array(img)

img = np.expand\_dims(img, axis=0)

img = keras.applications.vgg19.preprocess\_input(img)

return img

**def** deprocess\_image(img):

img = img.reshape((img\_height, img\_width, 3))

img[:, :, 0] += 103.939

img[:, :, 1] += 116.779

img[:, :, 2] += 123.68

img = img[:, :, ::-1]

img = np.clip(img, 0, 255).astype("uint8")

return img

- preprocess\_image: Chuẩn bị hình ảnh đầu vào để đưa vào mô hình VGG19, bao gồm tải, thay đổi kích thước, và chuẩn hóa.

- deprocess\_image: Chuyển dữ liệu từ dạng tensor đã chuẩn hóa (VGG19) về ảnh thông thường để có thể hiển thị.

style\_layer\_names = [

"block1\_conv1",

"block2\_conv1",

"block3\_conv1",

"block4\_conv1",

"block5\_conv1",

]

content\_layer\_name = "block5\_conv2"

total\_variation\_weight = 1e-6

style\_weight = 1e-6

content\_weight = 2.5e-8

**def** compute\_loss(combination\_image, base\_image, style\_reference\_image):

input\_tensor = tf.concat(

[base\_image, style\_reference\_image, combination\_image], axis=0

)

features = feature\_extractor(input\_tensor)

loss = tf.zeros(shape=())

layer\_features = features[content\_layer\_name]

base\_image\_features = layer\_features[0, :, :, :]

combination\_features = layer\_features[2, :, :, :]

loss = loss + content\_weight \* content\_loss(

base\_image\_features, combination\_features

)

for layer\_name in style\_layer\_names:

layer\_features = features[layer\_name]

style\_reference\_features = layer\_features[1, :, :, :]

combination\_features = layer\_features[2, :, :, :]

style\_loss\_value = style\_loss(

style\_reference\_features, combination\_features)

loss += (style\_weight / len(style\_layer\_names)) \* style\_loss\_value

loss += total\_variation\_weight \* total\_variation\_loss(combination\_image)

return loss

- compute\_loss là hàm tính toán tổng các loại mất mát trong Neural Style Transfer.

* + Mất mát nội dung: Đảm bảo ảnh kết hợp giống ảnh gốc về nội dung.
  + Mất mát phong cách: Đảm bảo ảnh kết hợp mang phong cách của ảnh tham khảo.
  + Mất mát độ biến thiên toàn cục: Giảm nhiễu và mượt mà ảnh kết hợp.

Hàm này sẽ được sử dụng trong quá trình tối ưu hóa để cập nhật giá trị của ảnh kết hợp sao cho nó có được cả nội dung từ ảnh gốc và phong cách từ ảnh tham khảo.

import tensorflow as tf

@tf.function

**def** compute\_loss\_and\_grads(combination\_image, base\_image, style\_reference\_image):

with tf.GradientTape() as tape:

loss = compute\_loss(combination\_image, base\_image, style\_reference\_image)

grads = tape.gradient(loss, combination\_image)

return loss, grads

optimizer = keras.optimizers.SGD(

keras.optimizers.schedules.ExponentialDecay(

initial\_learning\_rate=100.0, decay\_steps=100, decay\_rate=0.96

)

)

base\_image = preprocess\_image(base\_image\_path)

style\_reference\_image = preprocess\_image(style\_reference\_image\_path)

combination\_image = tf.Variable(preprocess\_image(base\_image\_path))

iterations = 400

for i in range(1, iterations + 1):

loss, grads = compute\_loss\_and\_grads(

combination\_image, base\_image, style\_reference\_image

)

optimizer.apply\_gradients([(grads, combination\_image)])

if i % 100 == 0:

print(**f**"Iteration {i}: loss={loss**:.2f**}")

img = deprocess\_image(combination\_image.numpy())

fname = **f**"combination\_image\_at\_iteration\_{i}.png"

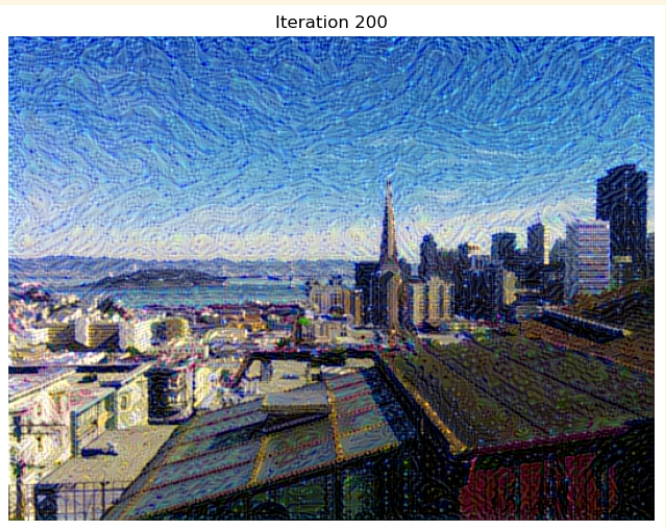
keras.utils.save\_img(fname, img)

* Đoạn mã thực hiện tối ưu hóa ảnh kết hợp để ảnh này có nội dung từ ảnh gốc và phong cách từ ảnh tham khảo.
* Tối ưu hóa bằng cách tính toán mất mát và gradient, sau đó cập nhật ảnh kết hợp qua mỗi bước với sự trợ giúp của trình tối ưu hóa SGD.
* Sau mỗi 100 vòng lặp, ảnh kết hợp sẽ được lưu lại và in ra giá trị mất mát hiện tại.

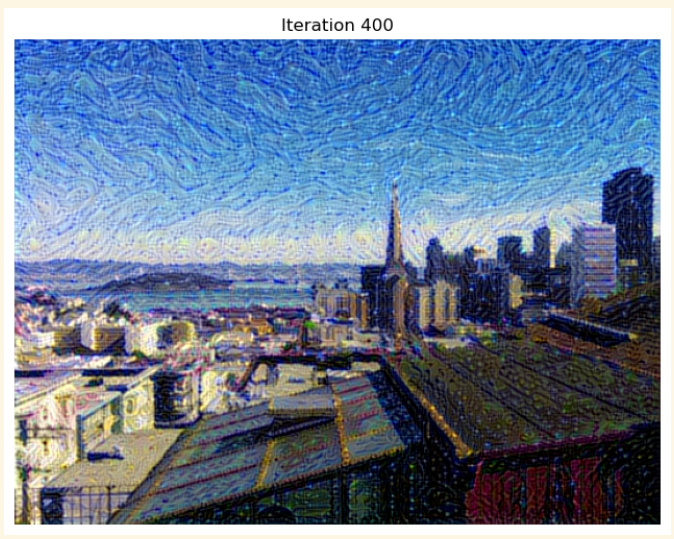
Kết quả cuối cùng là ảnh kết hợp mang đặc điểm nội dung của ảnh gốc và phong cách của ảnh tham khảo, sau 400 vòng lặp tối ưu hóa.

Kết quả:









4.1.4 Generating images with variational autoencoders

Code:

from tensorflow import keras

from tensorflow.keras import layers

latent\_dim = 2

encoder\_inputs = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder\_inputs)

x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)

x = layers.Flatten()(x)

x = layers.Dense(16, activation="relu")(x)

z\_mean = layers.Dense(latent\_dim, name="z\_mean")(x)

z\_log\_var = layers.Dense(latent\_dim, name="z\_log\_var")(x)

encoder = keras.Model(encoder\_inputs, [z\_mean, z\_log\_var], name="encoder")

import tensorflow as tf

**class** Sampler(layers.Layer):

**def** call(self, z\_mean, z\_log\_var):

batch\_size = tf.shape(z\_mean)[0]

z\_size = tf.shape(z\_mean)[1]

epsilon = tf.random.normal(shape=(batch\_size, z\_size))

return z\_mean + tf.exp(0.5 \* z\_log\_var) \* epsilon

**class** VAE(keras.Model):

**def** \_\_init\_\_(self, encoder, decoder, \*\*kwargs):

super().\_\_init\_\_(\*\*kwargs)

self.encoder = encoder

self.decoder = decoder

self.sampler = Sampler()

self.total\_loss\_tracker = keras.metrics.Mean(name="total\_loss")

self.reconstruction\_loss\_tracker = keras.metrics.Mean(

name="reconstruction\_loss")

self.kl\_loss\_tracker = keras.metrics.Mean(name="kl\_loss")

@property

**def** metrics(self):

return [self.total\_loss\_tracker,

self.reconstruction\_loss\_tracker,

self.kl\_loss\_tracker]

**def** train\_step(self, data):

with tf.GradientTape() as tape:

z\_mean, z\_log\_var = self.encoder(data)

z = self.sampler(z\_mean, z\_log\_var)

reconstruction = decoder(z)

reconstruction\_loss = tf.reduce\_mean(

tf.reduce\_sum(

keras.losses.binary\_crossentropy(data, reconstruction),

axis=(1, 2)

)

)

kl\_loss = -0.5 \* (1 + z\_log\_var - tf.square(z\_mean) - tf.exp(z\_log\_var))

total\_loss = reconstruction\_loss + tf.reduce\_mean(kl\_loss)

grads = tape.gradient(total\_loss, self.trainable\_weights)

self.optimizer.apply\_gradients(zip(grads, self.trainable\_weights))

self.total\_loss\_tracker.update\_state(total\_loss)

self.reconstruction\_loss\_tracker.update\_state(reconstruction\_loss)

self.kl\_loss\_tracker.update\_state(kl\_loss)

return {

"total\_loss": self.total\_loss\_tracker.result(),

"reconstruction\_loss": self.reconstruction\_loss\_tracker.result(),

"kl\_loss": self.kl\_loss\_tracker.result(),

}

### **a.** \_\_init\_\_(self, encoder, decoder, \*\*kwargs)

Phương thức khởi tạo của lớp VAE.

* encoder: Mô hình encoder, chuyển đầu vào (ví dụ: ảnh) thành các giá trị tiềm ẩn (latent variables), bao gồm z\_mean và z\_log\_var (trung bình và độ lệch chuẩn trong phân phối Gaussian).
* decoder: Mô hình decoder, tái tạo lại dữ liệu từ không gian ẩn.
* sampler: Lớp Sampler này thực hiện việc lấy mẫu từ phân phối tiềm ẩn để tạo ra các giá trị z từ z\_mean và z\_log\_var.
* total\_loss\_tracker, reconstruction\_loss\_tracker, kl\_loss\_tracker: Các bộ đếm được sử dụng để theo dõi và ghi nhận các giá trị mất mát trong quá trình huấn luyện.

### b. metrics

Phương thức này trả về danh sách các chỉ số cần theo dõi trong quá trình huấn luyện:

* total\_loss\_tracker: Mất mát tổng hợp (bao gồm cả mất mát tái tạo và mất mát KL).
* reconstruction\_loss\_tracker: Mất mát tái tạo, đo lường sự khác biệt giữa dữ liệu gốc và dữ liệu tái tạo.
* kl\_loss\_tracker: Mất mát KL (Kullback-Leibler divergence), đo lường sự khác biệt giữa phân phối tiềm ẩn học được và phân phối chuẩn.

### **c**. train\_step(self, data)

Phương thức này định nghĩa cách mà mỗi bước huấn luyện được thực hiện:

* tf.GradientTape: Được sử dụng để tính gradient của mất mát và áp dụng tối ưu hóa.
* z\_mean, z\_log\_var = self.encoder(data): Encoder chuyển dữ liệu đầu vào thành các tham số phân phối tiềm ẩn (z\_mean và z\_log\_var).
* z = self.sampler(z\_mean, z\_log\_var): Sampler lấy mẫu từ phân phối tiềm ẩn để tạo ra giá trị z.
* reconstruction = decoder(z): Decoder tái tạo lại dữ liệu từ z.
* reconstruction\_loss: Mất mát tái tạo, được tính bằng binary crossentropy giữa dữ liệu gốc và dữ liệu tái tạo.
  + tf.reduce\_mean(tf.reduce\_sum(...)): Lấy trung bình các giá trị của mất mát qua các trục không gian 2D (trục chiều cao và chiều rộng của ảnh).
* kl\_loss: Mất mát KL, đo lường sự khác biệt giữa phân phối tiềm ẩn đã học và phân phối chuẩn (Gaussian).
* total\_loss = reconstruction\_loss + tf.reduce\_mean(kl\_loss): Mất mát tổng hợp, là tổng của mất mát tái tạo và mất mát KL.
* grads = tape.gradient(total\_loss, self.trainable\_weights): Tính gradient của mất mát tổng hợp đối với các trọng số của mô hình.
* self.optimizer.apply\_gradients(zip(grads, self.trainable\_weights)): Cập nhật trọng số của mô hình bằng cách áp dụng gradient.

### **d. Cập nhật các bộ đếm**

Sau khi tính toán gradient và cập nhật trọng số, các bộ đếm cho tổng mất mát, mất mát tái tạo, và mất mát KL sẽ được cập nhật.

### **e. Trả về các kết quả**

Phương thức train\_step trả về các giá trị của các chỉ số theo dõi:

* total\_loss: Mất mát tổng hợp.
* reconstruction\_loss: Mất mát tái tạo.
* kl\_loss: Mất mát KL.

import numpy as np

(x\_train, \_), (x\_test, \_) = keras.datasets.mnist.load\_data()

mnist\_digits = np.concatenate([x\_train, x\_test], axis=0)

mnist\_digits = np.expand\_dims(mnist\_digits, -1).astype("float32") / 255

vae = VAE(encoder, decoder)

vae.compile(optimizer=keras.optimizers.Adam(), run\_eagerly=True)

vae.fit(mnist\_digits, epochs=3, batch\_size=128)

import matplotlib.pyplot as plt

n = 30

digit\_size = 28

figure = np.zeros((digit\_size \* n, digit\_size \* n))

grid\_x = np.linspace(-1, 1, n)

grid\_y = np.linspace(-1, 1, n)[::-1]

for i, yi in enumerate(grid\_y):

for j, xi in enumerate(grid\_x):

z\_sample = np.array([[xi, yi]])

x\_decoded = vae.decoder.predict(z\_sample)

digit = x\_decoded[0].reshape(digit\_size, digit\_size)

figure[

i \* digit\_size : (i + 1) \* digit\_size,

j \* digit\_size : (j + 1) \* digit\_size,

] = digit

plt.figure(figsize=(15, 15))

start\_range = digit\_size // 2

end\_range = n \* digit\_size + start\_range

pixel\_range = np.arange(start\_range, end\_range, digit\_size)

sample\_range\_x = np.round(grid\_x, 1)

sample\_range\_y = np.round(grid\_y, 1)

plt.xticks(pixel\_range, sample\_range\_x)

plt.yticks(pixel\_range, sample\_range\_y)

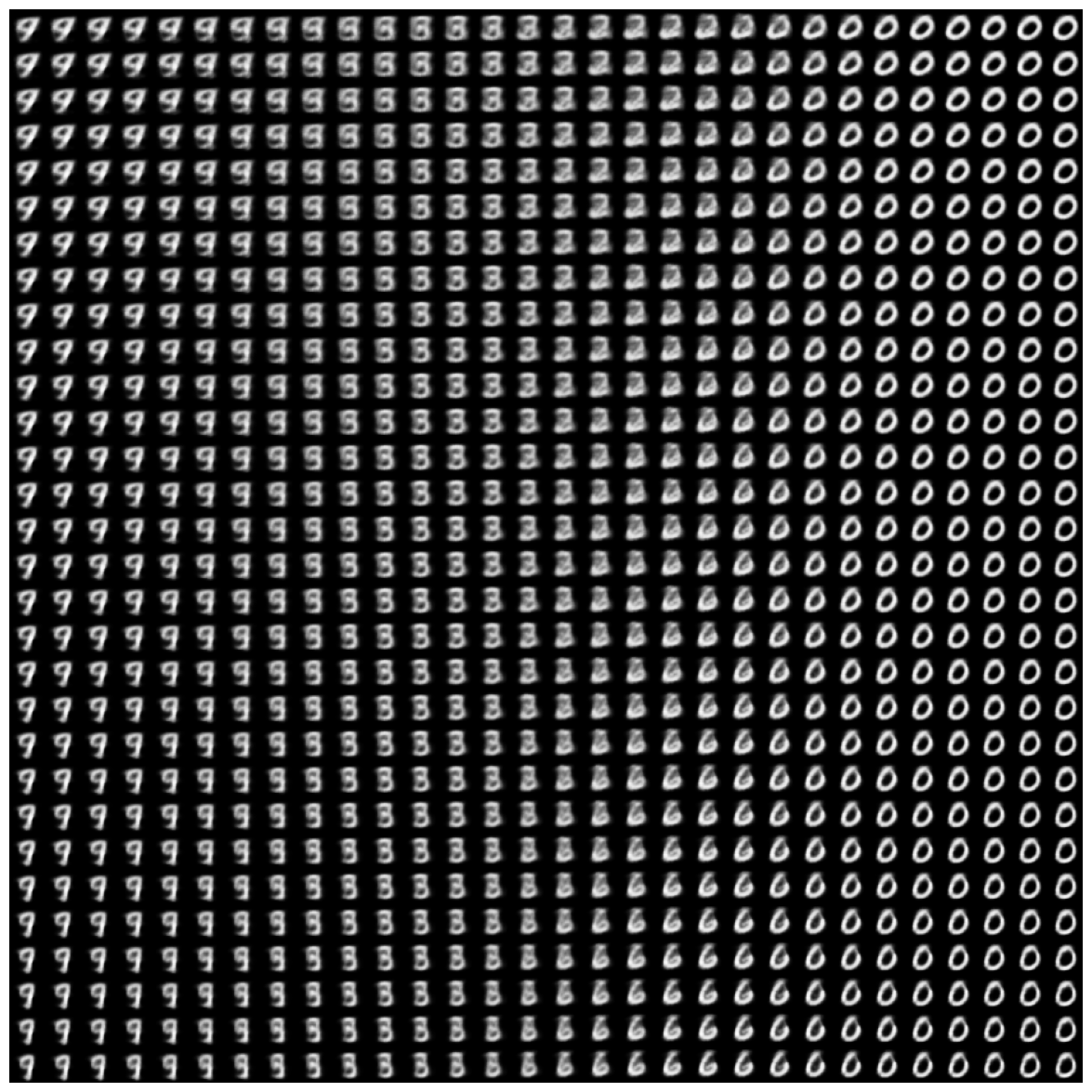
plt.xlabel("z[0]")

plt.ylabel("z[1]")

plt.axis("off")

plt.imshow(figure, cmap="Greys\_r")

Kết quả:



4.1.5 Introduction to generative adversarial networks

Code:

import gdown

*# Link Google Drive file ID*

file\_id = '1O7m1010EJjLE5QxLZiM9Fpjs7Oj6e684'

url = **f**'https://drive.google.com/uc?id={file\_id}'

gdown.download(url, 'celeba\_gan/data.zip', quiet=False)

import zipfile

*# Giải nén file ZIP*

with zipfile.ZipFile('celeba\_gan/data.zip', 'r') as zip\_ref:

zip\_ref.extractall('celeba\_gan')

from tensorflow import keras

dataset = keras.utils.image\_dataset\_from\_directory(

"celeba\_gan",

label\_mode=None,

image\_size=(64, 64),

batch\_size=32,

smart\_resize=True)

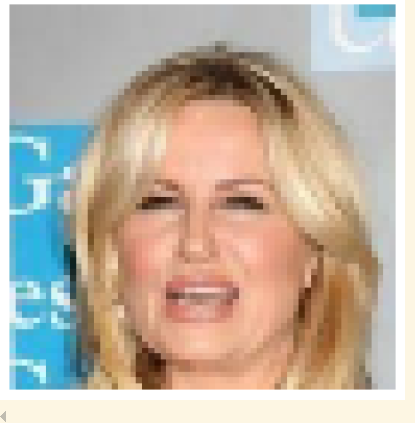
import matplotlib.pyplot as plt

for x in dataset:

plt.axis("off")

plt.imshow((x.numpy() \* 255).astype("int32")[0])

break



from tensorflow.keras import layers

discriminator = keras.Sequential(

[

keras.Input(shape=(64, 64, 3)),

layers.Conv2D(64, kernel\_size=4, strides=2, padding="same"),

layers.LeakyReLU(alpha=0.2),

layers.Conv2D(128, kernel\_size=4, strides=2, padding="same"),

layers.LeakyReLU(alpha=0.2),

layers.Conv2D(128, kernel\_size=4, strides=2, padding="same"),

layers.LeakyReLU(alpha=0.2),

layers.Flatten(),

layers.Dropout(0.2),

layers.Dense(1, activation="sigmoid"),

],

name="discriminator",

)

import tensorflow as tf

**class** GAN(keras.Model):

**def** \_\_init\_\_(self, discriminator, generator, latent\_dim):

super().\_\_init\_\_()

self.discriminator = discriminator

self.generator = generator

self.latent\_dim = latent\_dim

self.d\_loss\_metric = keras.metrics.Mean(name="d\_loss")

self.g\_loss\_metric = keras.metrics.Mean(name="g\_loss")

**def** compile(self, d\_optimizer, g\_optimizer, loss\_fn):

super(GAN, self).compile()

self.d\_optimizer = d\_optimizer

self.g\_optimizer = g\_optimizer

self.loss\_fn = loss\_fn

@property

**def** metrics(self):

return [self.d\_loss\_metric, self.g\_loss\_metric]

**def** train\_step(self, real\_images):

batch\_size = tf.shape(real\_images)[0]

random\_latent\_vectors = tf.random.normal(

shape=(batch\_size, self.latent\_dim))

generated\_images = self.generator(random\_latent\_vectors)

combined\_images = tf.concat([generated\_images, real\_images], axis=0)

labels = tf.concat(

[tf.ones((batch\_size, 1)), tf.zeros((batch\_size, 1))],

axis=0

)

labels += 0.05 \* tf.random.uniform(tf.shape(labels))

with tf.GradientTape() as tape:

predictions = self.discriminator(combined\_images)

d\_loss = self.loss\_fn(labels, predictions)

grads = tape.gradient(d\_loss, self.discriminator.trainable\_weights)

self.d\_optimizer.apply\_gradients(

zip(grads, self.discriminator.trainable\_weights)

)

random\_latent\_vectors = tf.random.normal(

shape=(batch\_size, self.latent\_dim))

misleading\_labels = tf.zeros((batch\_size, 1))

with tf.GradientTape() as tape:

predictions = self.discriminator(

self.generator(random\_latent\_vectors))

g\_loss = self.loss\_fn(misleading\_labels, predictions)

grads = tape.gradient(g\_loss, self.generator.trainable\_weights)

self.g\_optimizer.apply\_gradients(

zip(grads, self.generator.trainable\_weights))

self.d\_loss\_metric.update\_state(d\_loss)

self.g\_loss\_metric.update\_state(g\_loss)

return {"d\_loss": self.d\_loss\_metric.result(),

"g\_loss": self.g\_loss\_metric.result()}