

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO
THỰC TẬP CƠ SỞ

Họ Và Tên : Thảo A Bảy
Mã Sinh Viên : B22DCCN072
Nhóm Thực Tập : 30
Giảng Viên Hướng Dẫn : PGS. TS. Trần Đình Quế

HÀ NỘI 2025

MỤC LỤC

LỜI MỞ ĐẦU	3
NỘI DUNG	4
Chương 1: Trí Tuệ Nhân Tạo Và Các Ứng Dụng	4
1.1 Giới thiệu về Trí tuệ nhân tạo (AI)	4
1.2 Các lĩnh vực chính trong AI	4
1.2.1 Machine Learning (Học máy)	4
1.2.2 Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP)	9
1.2.3 Thị giác máy tính (Computer Vision)	10
1.2.4 Hệ chuyên gia (Expert Systems)	12
1.2.5 Robot học (Robotics)	14
Chương 2: Các Kỹ Thuật Học Sâu	16
2.1. Giới thiệu tổng quan về học sâu (Deep Learning)	16
2.2. Mạng nơ-ron nhân tạo (Artificial Neural Network – ANN)	17
2.3. Mạng nơ-ron tích chập (Convolutional Neural Network – CNN)	19
2.4. Mạng nơ-ron hồi tiếp (Recurrent Neural Network – RNN)	22
2.5. Mạng sinh đối kháng (Generative Adversarial Network – GAN)	25
2.6. Mạng nơ-ron Transformer	29
2.7. Autoencoder	33
2.8. Học tăng cường sâu (Deep Reinforcement Learning – DRL)	36
2.9. Bảng so sánh giữa các kỹ thuật học sâu phổ biến	40
Chương 3: Triển Khai Ứng Dụng Học Sâu	42
3.1. Mục tiêu triển khai	42
3.2. Quy trình triển khai ứng dụng học sâu	42
3.2.1. Chuẩn bị mô hình	42
3.2.2. Tạo API hoặc giao diện web	42
3.3. Minh họa mô hình ứng dụng cụ thể	42
3.3.1. Ứng dụng dự đoán địa danh bằng hình ảnh	42
3.3.2. Hệ thống chatbot hỏi đáp trong du lịch	55
KẾT LUẬN	63
TÀI LIỆU THAM KHẢO	64
TƯỜNG TRÌNH CHỈNH SỬA BÁO CÁO	65

LỜI MỞ ĐẦU

Trong thời đại chuyển đổi số mạnh mẽ hiện nay, trí tuệ nhân tạo (AI) không chỉ là một xu hướng công nghệ mà đã trở thành nền tảng cốt lõi thúc đẩy sự đổi mới sáng tạo trong nhiều lĩnh vực như y tế, giáo dục, tài chính, công nghiệp, và đặc biệt là công nghệ thông tin. Với tầm ảnh hưởng ngày càng sâu rộng, AI đang góp phần thay đổi cách con người sinh sống, làm việc và tương tác với thế giới xung quanh. Nhận thức được tầm quan trọng đó, việc tìm hiểu, nghiên cứu và ứng dụng AI đặc biệt là các kỹ thuật học sâu (Deep Learning) vào thực tế trở thành nhiệm vụ cấp thiết đối với sinh viên ngành Công nghệ Thông tin trong thời kỳ 4.0.

Báo cáo thực tập cơ sở này là kết quả quá trình học tập, nghiên cứu và vận dụng các kiến thức nền tảng về AI mà em đã tích lũy trong suốt thời gian học tập. Nội dung báo cáo tập trung vào việc trình bày các khái niệm, kỹ thuật và ứng dụng tiêu biểu của trí tuệ nhân tạo, từ các thuật toán học máy truyền thống đến các mô hình học sâu hiện đại như mạng nơ-ron tích chập (CNN), mạng hồi tiếp (RNN), GAN, Transformer,... Bên cạnh phần lý thuyết, em cũng đã triển khai thực tế hai mô hình ứng dụng: hệ thống nhận diện địa danh từ ảnh và chatbot hỏi đáp trong lĩnh vực du lịch – minh chứng cụ thể cho khả năng đưa mô hình học sâu vào phục vụ nhu cầu người dùng.

Thông qua báo cáo này, em không chỉ mong muốn thể hiện năng lực tiếp thu và vận dụng kiến thức chuyên ngành, mà còn thể hiện sự chủ động học hỏi, khả năng làm việc thực tế cũng như tinh thần nghiên cứu nghiêm túc. Dù còn nhiều hạn chế và khó khăn trong quá trình thực hiện, nhưng em tin rằng báo cáo này là một bước khởi đầu quan trọng trên con đường học tập và nghiên cứu về trí tuệ nhân tạo trong tương lai.

Em xin chân thành cảm ơn thầy PGS.TS. Trần Đình Quế và sự đồng hành của các bạn trong nhóm thực tập. Em hy vọng sẽ nhận được những góp ý quý báu từ Thầy để hoàn thiện hơn nữa kỹ năng và kiến thức của bản thân.

NỘI DUNG

Chương 1: Trí Tuệ Nhân Tạo Và Các Ứng Dụng

1.1 Giới thiệu về Trí tuệ nhân tạo (AI)

Trí tuệ nhân tạo (Artificial Intelligence – AI) là một nhánh của khoa học máy tính liên quan đến việc xây dựng các hệ thống có khả năng thực hiện những nhiệm vụ vốn dĩ đòi hỏi trí tuệ con người. Những nhiệm vụ này bao gồm học tập (learning), suy luận (reasoning), giải quyết vấn đề (problem solving), hiểu ngôn ngữ tự nhiên (natural language understanding), và nhận diện hình ảnh (computer vision). AI không chỉ là công nghệ, mà còn là trụ cột của cuộc cách mạng công nghiệp lần thứ tư.

Khái niệm AI đã được đề xuất từ những năm 1950, khi các nhà khoa học như Alan Turing và John McCarthy đặt ra câu hỏi: "Liệu máy tính có thể suy nghĩ như con người không?". Từ đó đến nay, AI đã có sự phát triển vượt bậc nhờ vào sức mạnh tính toán, dữ liệu lớn (Big Data) và các thuật toán tiên tiến, đặc biệt là các mô hình học máy (Machine Learning) và học sâu (Deep Learning).

AI được phân loại theo hai cách chính:

- ❖ Theo mức độ trí thông minh:
 - Narrow AI: Là loại AI được lập trình để thực hiện một nhiệm vụ cụ thể, ví dụ như trợ lý ảo Siri, hệ thống nhận diện khuôn mặt, hay các công cụ gợi ý sản phẩm. Đây là loại AI phổ biến nhất hiện nay.
 - General AI: Là hệ thống có khả năng tư duy, học hỏi, và thích nghi với mọi tình huống giống như con người. AI mạnh hiện vẫn chưa đạt được trong thực tế và còn là mục tiêu dài hạn trong nghiên cứu.
- ❖ Theo phương pháp hoạt động:
 - Symbolic AI: Dựa vào các quy tắc logic và tri thức chuyên gia.
 - Connectionist AI: Dựa vào mạng nơ-ron nhân tạo mô phỏng cách hoạt động của bộ não người.

1.2 Các lĩnh vực chính trong AI

1.2.1 Machine Learning (Học máy)

1. Machine Learning là gì?

Machine Learning (Học máy) là một nhánh của Trí tuệ nhân tạo, cho phép máy tính tự học từ dữ liệu và cải thiện độ chính xác mà không cần lập trình rõ ràng từng bước. Thay vì viết code để nói với máy rằng "nếu điều kiện A xảy ra thì làm B", ta cung cấp cho máy dữ liệu huấn luyện, và máy sẽ tự rút ra quy luật.

Ví dụ: Nếu bạn cung cấp cho máy ảnh của nhiều con mèo và chó, máy sẽ "học" cách phân biệt được con nào là mèo, con nào là chó.

2. Các loại Học máy cơ bản

2.1. Học có giám sát (Supervised Learning)

Supervised Learning là một nhánh của Machine Learning trong đó mô hình được huấn luyện bằng dữ liệu đã gán nhãn sẵn (labeled data). Mỗi đầu vào (input) đều đi kèm với một đầu ra mong muốn (output), và mục tiêu của mô hình là học được mối quan hệ giữa chúng để dự đoán đúng đầu ra cho các dữ liệu mới chưa từng thấy.

Kỹ thuật	Thư viện	Ứng dụng thực tế
K-Nearest Neighbors (KNN)	scikit-learn	Phân loại hình ảnh, nhận diện chữ viết tay (MNIST), dự đoán bệnh
Decision Tree	scikit-learn, XGBoost	Phân loại khách hàng, chẩn đoán y tế, cây quyết định đầu tư
Random Forest	scikit-learn, LightGBM	Dự đoán tín dụng, phân tích rủi ro, đánh giá chất lượng sản phẩm
Linear Regression	scikit-learn, statsmodels	Dự đoán giá nhà, doanh thu, điểm số, nhiệt độ
Support Vector Machine (SVM)	scikit-learn, libsvm	Nhận diện khuôn mặt, phân loại văn bản, phát hiện gian lận
Naive Bayes	scikit-learn, NLTK	Lọc thư rác, phân tích văn bản, phân loại tin tức
Neural Networks (MLP)	TensorFlow, Keras, PyTorch	Phân loại ảnh, nhận diện giọng nói, dự đoán chuỗi

Code demo: Phân loại (Classification) – KNN với dữ liệu Iris

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Bước 1: Tải dữ liệu hoa Iris
iris = load_iris()
X, y = iris.data, iris.target

# Bước 2: Chia dữ liệu train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Bước 3: Huấn luyện mô hình KNN
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
```

```
# Bước 4: Dự đoán và đánh giá
y_pred = model.predict(X_test)
print("Độ chính xác:", accuracy_score(y_test, y_pred))
```

Kết quả mẫu:

Độ chính xác: 1.0

2.2. Học không giám sát (Unsupervised Learning)

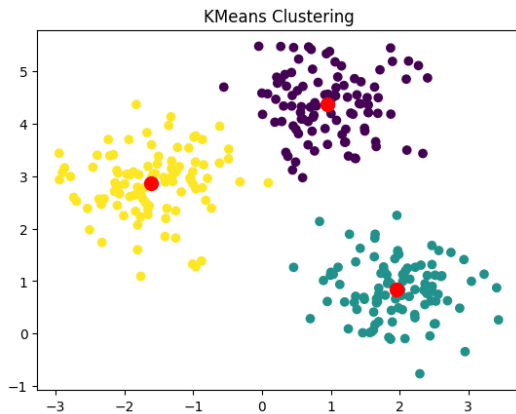
Học không giám sát là một phương pháp trong Machine Learning mà dữ liệu không có nhãn đầu ra. Mục tiêu của mô hình là tự tìm ra cấu trúc, mẫu hoặc nhóm trong dữ liệu mà không cần biết trước đáp án đúng.

Kỹ thuật	Thư viện	Ứng dụng thực tế
K-Means Clustering	scikit-learn, Spark MLlib	Phân nhóm khách hàng, nhóm ảnh, phân cụm thị trường
DBSCAN (Density-based Clustering)	scikit-learn, hdbscan	Phát hiện bất thường, phân cụm không xác định trước số cụm
Hierarchical Clustering	scipy, scikit-learn	Phân tích phân cấp nhóm người dùng, biểu đồ cây (dendrogram)
t-SNE / UMAP	scikit-learn, umap-learn	Trực quan hóa dữ liệu đa chiều trong không gian 2D/3D
Autoencoder	TensorFlow, PyTorch	Nén dữ liệu ảnh, phát hiện bất thường, giảm nhiễu ảnh hoặc dữ liệu
Gaussian Mixture Models (GMM)	scikit-learn	Phân tích xác suất cụm, phát hiện cấu trúc ẩn trong dữ liệu

Code demo: K-Means Clustering – Phân nhóm dữ liệu 2D

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Tạo dữ liệu giả lập
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.60,
random_state=0)
# Áp dụng KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
# Vẽ biểu đồ phân cụm
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1], s=100,
c='red')
plt.title("KMeans Clustering")
plt.show()
```



2.3. Học tăng cường (Reinforcement Learning)

Học tăng cường là một nhánh của Machine Learning, nơi một "tác nhân" (agent) học cách hành động trong một môi trường để tối đa hóa phần thưởng tích lũy theo thời gian. Khác với Supervised Learning (có nhãn) và Unsupervised Learning (không nhãn), trong Reinforcement Learning:

- Agent tự khám phá thông qua thử-sai (trial-and-error).
- Không có "đáp án đúng" rõ ràng.
- Chỉ có phần thưởng hoặc phạt sau mỗi hành động.

Kỹ thuật	Thư viện	Ứng dụng thực tế
Q-Learning	OpenAI Gym, custom Python	Robot học đi, điều hướng maze, học quy tắc đơn giản
Deep Q-Network (DQN)	TensorFlow, PyTorch, Stable-Baselines3	AI chơi game Atari, điều hướng xe đơn giản
PPO (Proximal Policy Optimization)	Stable-Baselines3, RLlib	Robot tự học tối ưu hành động, AI chơi game 3D (Minecraft, Dota)
Actor-Critic / A2C / DDPG	Stable-Baselines3, SpinningUp	Điều khiển cánh tay robot, mô phỏng drone bay
Environment Simulation	OpenAI Gym, Unity ML-Agents, Isaac Gym	Mô phỏng môi trường ảo cho robot, game, giao thông

Code Demo: Mô phỏng Robot học thoát khỏi mê cung 4x4

- Môi trường (Environment)

Grid 4x4.

S là điểm xuất phát, G là đích, X là chướng ngại.

S - - -

- X - -

- - X -

- - - G

▪ Code Q-Learning mê cung

```
import numpy as np
import random
# Thiết lập môi trường
maze = np.array([
    [0, 0, 0, 0],
    [0, -100, 0, 0],
    [0, 0, -100, 0],
    [0, 0, 0, 100]
])
actions = ['up', 'down', 'left', 'right']
q_table = np.zeros((4, 4, len(actions)))
# Tham số học
alpha = 0.1    # learning rate
gamma = 0.9    # discount
epsilon = 0.1  # exploration rate
episodes = 500
def is_valid(pos):
    i, j = pos
    return 0 <= i < 4 and 0 <= j < 4 and maze[i][j] != -100
def get_next_state(state, action):
    i, j = state
    if action == 'up': i -= 1
    if action == 'down': i += 1
    if action == 'left': j -= 1
    if action == 'right': j += 1
    return (i, j) if is_valid((i, j)) else state
def train():
    for _ in range(episodes):
        state = (0, 0)
        while state != (3, 3):
            i, j = state
            if random.uniform(0, 1) < epsilon:
```



```

        action_idx = random.randint(0, 3)
    else:
        action_idx = np.argmax(q_table[i][j])
    action = actions[action_idx]
    next_state = get_next_state(state, action)
    ni, nj = next_state
    reward = maze[ni][nj]
    old_value = q_table[i][j][action_idx]
    next_max = np.max(q_table[ni][nj])
    new_value = (1 - alpha) * old_value + alpha * (reward + gamma *
next_max)
    q_table[i][j][action_idx] = new_value
    state = next_state
# Huấn luyện
train()
# In chiến lược học được
print("Chính sách tốt nhất học được:")
for i in range(4):
    row = ''
    for j in range(4):
        if maze[i][j] == -100:
            row += ' X '
        elif maze[i][j] == 100:
            row += ' G '
        else:
            best_action = actions[np.argmax(q_table[i][j])]
            row += f'{best_action[0].upper()} '
    print(row)

```

▪ **Kết quả:**

```

D R R D
D X D D
D R X D
R R D G

```

1.2.2 Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP)

Xử lý ngôn ngữ tự nhiên (NLP) là lĩnh vực của Trí tuệ nhân tạo (AI) giúp máy tính hiểu và xử lý ngôn ngữ của con người dưới dạng văn bản hoặc lời nói. NLP kết hợp giữa:

- Ngôn ngữ học (hiểu cấu trúc và nghĩa của ngôn ngữ),

- Khoa học máy tính (mã hóa, tính toán),
- Học máy (tự học từ dữ liệu văn bản).

Kỹ thuật	Thư viện	Ứng dụng thực tế
Tokenization	NLTK, spaCy	Tách từ, xử lý cú pháp trong chatbot, phân tích văn bản
Stop Words Removal	NLTK, TextBlob	Loại bỏ từ dư thừa trong tìm kiếm, trích xuất thông tin
Stemming / Lemmatization	NLTK, spaCy, TextBlob	Chuẩn hóa văn bản, phân tích nội dung
NER (Named Entity Recognition)	spaCy, transformers (BERT)	Nhận diện tên người, địa danh, tổ chức trong tài liệu hoặc tin tức
Sentiment Analysis	TextBlob, VADER, transformers	Phân tích cảm xúc bình luận, đánh giá sản phẩm
Machine Translation	transformers, OpenNMT	Dịch máy Anh–Việt, Việt–Anh, đa ngôn ngữ
Question Answering (QA)	BERT, RoBERTa, transformers	Trả lời câu hỏi tự động, trợ lý ảo

Code Demo

Phân tích cảm xúc bằng TextBlob

```
from textblob import TextBlob
text = "I absolutely love this new AI feature!"
blob = TextBlob(text)
print("Cảm xúc:", blob.sentiment)
```

Kết quả:

Cảm xúc: Sentiment(polarity=0.3352272727272727,
subjectivity=0.5272727272727272)

1.2.3 Thị giác máy tính (Computer Vision)

Computer Vision (CV) là một nhánh của Trí tuệ nhân tạo (AI), giúp máy tính “nhìn” và hiểu hình ảnh hoặc video giống như con người.

Kỹ thuật	Thư viện tiêu biểu	Ứng dụng thực tế
Image Classification	TensorFlow, PyTorch, Keras	Phân loại ảnh y tế, ảnh sản phẩm, ảnh động vật

Object Detection	OpenCV, YOLOv5, Detectron2, MMDetection	Camera giám sát, xe tự lái, đếm người, phát hiện hàng hóa
Semantic Segmentation	DeepLabV3, U-Net, SegFormer	Xe tự hành phân tích làn đường, ảnh vệ tinh, ảnh y tế
Instance Segmentation	Mask R-CNN, Detectron2	Nhận diện từng vật thể riêng biệt: từng người, từng chiếc xe
Face Recognition	dlib, face_recognition, OpenCV	Mở khóa điện thoại, điểm danh tự động, hệ thống an ninh
OCR (Text Recognition)	Tesseract OCR, EasyOCR, PaddleOCR	Nhận diện biển số xe, đọc hóa đơn, số CMND từ ảnh

Code demo

1. Phân loại ảnh – Keras + CIFAR-10

Phân loại ảnh với mô hình đơn giản trên tập CIFAR-10

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)
model = Sequential([
    Flatten(input_shape=(32, 32, 3)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=3, validation_data=(X_test, y_test))
```

Kết quả

```
Epoch 1/3
1563/1563 ————— 14s 7ms/step - accuracy: 0.2679 - loss: 2.0363 - val_accuracy: 0.3620 - val_loss: 1.7915
Epoch 2/3
1563/1563 ————— 11s 7ms/step - accuracy: 0.3694 - loss: 1.7700 - val_accuracy: 0.3610 - val_loss: 1.7729
Epoch 3/3
1563/1563 ————— 11s 7ms/step - accuracy: 0.3859 - loss: 1.7157 - val_accuracy: 0.4000 - val_loss: 1.6840
```

2. Nhận diện khuôn mặt – OpenCV

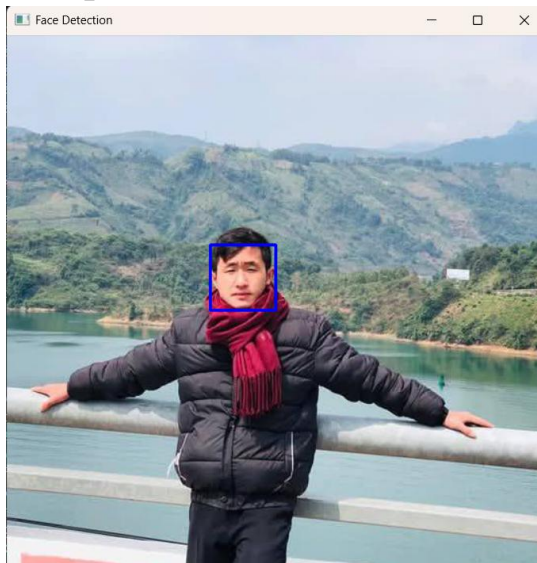
```
import cv2
```

```

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
    "haarcascade_frontalface_default.xml")
img = cv2.imread("face.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
cv2.imshow("Face Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Kết quả



1.2.4 Hệ chuyên gia (Expert Systems)

Hệ chuyên gia (Expert System) là một chương trình AI mô phỏng khả năng suy nghĩ và quyết định của một chuyên gia con người trong một lĩnh vực cụ thể. Mục tiêu: Tư vấn, chẩn đoán hoặc đưa ra quyết định như một chuyên gia.

Kỹ thuật	Thư viện	Ứng dụng thực tế
Rule-based Reasoning (suy luận luật)	CLIPS, PyKnow, Drools	Chẩn đoán bệnh, kiểm tra lỗi thiết bị, hệ thống tư vấn y tế
Forward Chaining (suy diễn tiến)	PyKnow, experta	Hệ chuyên gia về luật, phân tích tình huống an ninh
Backward Chaining (suy diễn lùi)	PyKnow, CLIPS	Hệ thống khuyến nghị, tư vấn tài chính
Rule Engine (Bộ luật IF–THEN)	experta, Jess, Drools	Hệ thống khuyến nghị sản phẩm, đánh giá tín dụng

Explanation Facility	Xây dựng thủ công hoặc nội bộ	Giải thích lý do đưa ra chẩn đoán, đặc biệt trong y tế và luật
Inference Engine (Bộ suy diễn)	PyKnow, CLIPS, experta	Suy luận dựa trên luật và dữ kiện để đưa ra kết luận tự động

Code demo

1. Cài thư viện

pip install experta

2. Code hệ chuyên gia chẩn đoán bệnh

*from experta import **

class BieuHien(Fact):

"""Dữ kiện: biểu hiện của người bệnh"""

pass

class HeChuyenGia(Fact):

"""Luật suy luận"""

pass

class ChuanDoanBenh(KnowledgeEngine):

@Rule(BieuHien(ho="có", sot="có", hat_hoi="không"))

def benh_cum(self):

print("Hệ chuyên gia: Bạn có thể bị CÚM.")

@Rule(BieuHien(ho="có", sot="không", hat_hoi="có"))

def benh_cam_lanh(self):

print("Hệ chuyên gia: Bạn có thể bị CẢM LẠNH.")

@Rule(BieuHien(ho="không", hat_hoi="có"))

def benh_di_ung(self):

print("Hệ chuyên gia: Bạn có thể bị DỊ ỨNG.")

@Rule(BieuHien())

def khong_ket_luan(self):

print("Hệ chuyên gia: Không thể đưa ra kết luận. Bạn nên đi khám bác sĩ.")

Chạy hệ thống

engine = ChuanDoanBenh()

engine.reset()

Nhập triệu chứng

ho = input("Bạn có bị ho không? (có/không): ").strip().lower()

sot = input("Bạn có bị sốt không? (có/không): ").strip().lower()

```

hat_hoi = input("Bạn có hắt hơi không? (có/không): ").strip().lower()
engine.declare(BieuHien(ho=ho, sot=sot, hat_hoi=hat_hoi))
engine.run()

```

Chạy code:

Bạn có bị ho không? có

Bạn có bị sốt không? có

Bạn có hắt hơi không? không

Hệ chuyên gia: Bạn có thể bị CÚM.

1.2.5 Robot học (Robotics)

Robotics là một ngành liên ngành của Kỹ thuật, AI và Tự động hóa, nghiên cứu về thiết kế, chế tạo, lập trình và vận hành robot.

Mục tiêu: Tạo ra các robot có khả năng cảm nhận (perception), ra quyết định (decision-making) và hành động (action) một cách tự động hoặc bán tự động.

Kỹ thuật	Thư viện	Ứng dụng thực tế
Cảm biến & Nhận thức (Perception)	OpenCV, ROS Sensor Drivers, MediaPipe	Camera, lidar, cảm biến siêu âm cho robot định hướng, phát hiện vật thể
Điều khiển robot (Motion Control)	Arduino, ROS Control, Gazebo Controllers	Điều khiển tay gấp, bánh xe, servo, quadcopter
Lập kế hoạch chuyển động (Path Planning)	A* Algorithm, Dijkstra, MoveIt (ROS)	Điều hướng robot tránh vật cản, lập lộ trình
SLAM (Simultaneous Localization and Mapping)	GMapping, Cartographer, RTAB-Map	Robot tự xây bản đồ, định vị bản thân trong môi trường mới
Tự động hóa bằng AI	TensorFlow, PyTorch, Reinforcement Learning	Robot tự học hành vi (nhắm, đi, né...) qua thử sai
Robot học tăng cường	Stable-Baselines3, OpenAI Gym, RLlib	Dạy robot chơi game, gấp đồ vật, tự học cân bằng
Giao tiếp người – robot	MQTT, ROS Topics, WebSockets, gRPC	Gửi lệnh từ web/app đến robot, nhận phản hồi từ robot
Mô phỏng robot	Gazebo, Webots, V-REP, CoppeliaSim	Mô phỏng thế giới ảo, kiểm tra thuật toán trước khi lắp ráp thật

Code demo

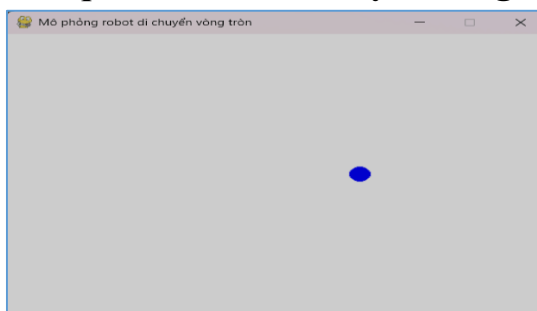
1. Cài đặt thư viện

pip install pygame

2. Code Python mô phỏng robot chạy vòng tròn

```
import pygame
import math
# Khởi tạo màn hình
pygame.init()
width, height = 500, 500
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Mô phỏng robot di chuyển vòng tròn")
clock = pygame.time.Clock()
# Thông số ban đầu
x, y = 250, 250
angle = 0
radius = 100
running = True
while running:
    screen.fill((255, 255, 255)) # Xóa màn hình với màu trắng
    # Tính vị trí mới theo công thức đường tròn
    x = 250 + radius * math.cos(angle)
    y = 250 + radius * math.sin(angle)
    angle += 0.05 # Góc quay
    # Vẽ robot (chấm tròn màu xanh)
    pygame.draw.circle(screen, (0, 0, 255), (int(x), int(y)), 10)
    # Cập nhật màn hình
    pygame.display.flip()
    clock.tick(30) # 30 FPS
    # Thoát khi đóng cửa sổ
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

Kết quả: Robot di chuyển vòng tròn



Chương 2: Các Kỹ Thuật Học Sâu

2.1. Giới thiệu tổng quan về học sâu (Deep Learning)

1. Định nghĩa Deep Learning

Deep Learning (học sâu) là một nhánh của Machine Learning (học máy) sử dụng các mô hình gọi là mạng nơ-ron nhân tạo có nhiều lớp ẩn (hidden layers) để học biểu diễn dữ liệu và thực hiện dự đoán hoặc phân loại.

Đặc điểm chính: học trực tiếp đặc trưng (features) từ dữ liệu mà không cần can thiệp thủ công từ con người như trong học máy truyền thống.

2. Lịch sử và sự phát triển

- 1943: McCulloch & Pitts đề xuất mô hình nơ-ron nhân tạo đầu tiên.
- 1980s: Thuật toán lan truyền ngược (backpropagation) giúp huấn luyện mạng nhiều lớp.
- 2012: Deep Learning bùng nổ khi mạng AlexNet chiến thắng cuộc thi ImageNet, vượt xa các phương pháp truyền thống.

3. Sự khác biệt giữa Machine Learning và Deep Learning

Tiêu chí	Machine Learning	Deep Learning
Phụ thuộc đặc trưng thủ công	Cao	Thấp
Cấu trúc mô hình	Đơn giản (SVM, Decision Trees)	Mạng nhiều lớp (Deep Neural Networks)
Hiệu quả với dữ liệu lớn	Không cao	Rất cao
Yêu cầu tài nguyên	Ít	Cao (GPU, RAM, v.v.)
Khả năng tự học	Có giới hạn	Tốt hơn nhiều

4. Ưu điểm của Deep Learning

- Tự động trích xuất đặc trưng từ dữ liệu thô (ảnh, âm thanh, văn bản).
- Hiệu suất cao trong các bài toán phức tạp như: nhận dạng hình ảnh, giọng nói, dịch máy, chơi game...
- Khả năng tổng quát tốt khi được huấn luyện với đủ dữ liệu.

5. Hạn chế của Deep Learning

- Cần lượng lớn dữ liệu và tài nguyên tính toán.
- Khó giải thích (black box): Mô hình phức tạp nên khó hiểu được tại sao nó đưa ra quyết định.
- Nguy cơ overfitting nếu không có biện pháp regularization tốt.

6. Các lĩnh vực ứng dụng

- Thị giác máy tính (Computer Vision): Nhận diện khuôn mặt, xe tự lái...

- Xử lý ngôn ngữ tự nhiên (NLP): Dịch máy, chatbot, phân tích cảm xúc...
- Nhận dạng giọng nói (Speech Recognition)
- Y tế: Phát hiện ung thư, chẩn đoán hình ảnh...
- Tài chính: Dự đoán rủi ro, phát hiện gian lận...

2.2. Mạng nơ-ron nhân tạo (Artificial Neural Network – ANN)

1. Khái niệm

Mạng nơ-ron nhân tạo (ANN) là một mô hình học máy được lấy cảm hứng từ cấu trúc của bộ não con người. Mạng này bao gồm các nơ-ron (neurons) liên kết với nhau qua các trọng số (weights) và các hàm kích hoạt. Mục tiêu của ANN là học các mối quan hệ phức tạp giữa các đầu vào và đầu ra thông qua huấn luyện.

2. Cấu trúc

- Lớp đầu vào: Nhận dữ liệu đầu vào.
- Lớp ẩn: Xử lý và học đặc trưng từ dữ liệu.
- Lớp đầu ra: Trả về kết quả dự đoán.

3. Hoạt động của Nơ-Ron

Mỗi nơ-ron trong mạng thực hiện các bước sau:

- Nhận đầu vào từ các nơ-ron khác hoặc dữ liệu trực tiếp.
- Tính toán trọng số kết hợp với bias (hệ số lệch).
- Áp dụng hàm kích hoạt (activation function) để quyết định đầu ra của nơ-ron.

Công thức:

$$z = \sum_i w_i x_i + b$$

$$a = f(z)$$

Trong đó:

- x_i : Đầu vào.
- w_i : Trọng số.
- b : Bias.
- f : Hàm kích hoạt.
- a : Đầu ra của nơ-ron.

4. Hàm Kích Hoạt (Activation Function)

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ (thường dùng trong phân loại nhị phân).
- Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (giá trị từ -1 đến 1).
- ReLU: $\text{ReLU}(x) = \max(0, x)$ (phổ biến trong học sâu).
- Softmax: Thường dùng trong lớp đầu ra cho phân loại đa lớp.

5. Quá trình Huấn Luyện

- Forward Propagation: Tín hiệu được truyền từ lớp đầu vào đến lớp đầu ra.
- Tính toán mất mát (Loss Calculation): Tính toán sự khác biệt giữa giá trị dự đoán và giá trị thực tế.
- Backpropagation: Tính toán gradient của hàm mất mát đối với trọng số và cập nhật trọng số bằng Gradient Descent.

6. Ứng Dụng

- Nhận dạng hình ảnh: Phân loại ảnh, nhận diện khuôn mặt.
- Xử lý ngôn ngữ tự nhiên (NLP): Dịch máy, chatbot.
- Dự đoán tài chính: Dự báo giá trị cổ phiếu, phân tích dữ liệu thị trường.
- Y tế: Phát hiện bệnh qua ảnh y khoa, phân tích dữ liệu gen.

7. Code Demo: Sử dụng Keras để xây dựng một Mạng Nơ-Ron Nhân Tạo (ANN) cơ bản để giải quyết bài toán phân loại dữ liệu MNIST (chữ số viết tay).

1. Cài đặt thư viện

```
pip install tensorflow numpy matplotlib
```

2. Code Demo ANN cho Phân loại MNIST

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
# 1. Tải dữ liệu MNIST (chữ số viết tay)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# 2. Tiền xử lý dữ liệu
x_train = x_train / 255.0 # Chuẩn hóa dữ liệu về [0, 1]
x_test = x_test / 255.0
# Chuyển dữ liệu thành dạng (batch_size, 28, 28, 1)
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
# 3. Xây dựng mô hình ANN
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28, 1)), # Chuyển ảnh 28x28 thành
    vector 1D
    layers.Dense(128, activation='relu'), # Lớp ẩn với 128 nơ-ron
    layers.Dropout(0.2), # Regularization: Dropout
    layers.Dense(10, activation='softmax') # Lớp đầu ra (10 lớp cho 10
    chữ số)
])
# 4. Compile mô hình
```

```

model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# 5. Huấn luyện mô hình
model.fit(x_train, y_train, epochs=5, batch_size=32,
validation_data=(x_test, y_test))
# 6. Đánh giá mô hình
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
# 7. Dự đoán cho một số mẫu
predictions = model.predict(x_test[:5])
# Hiển thị ảnh và dự đoán
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {np.argmax(predictions[i])}, True: {y_test[i]}")
    plt.show()

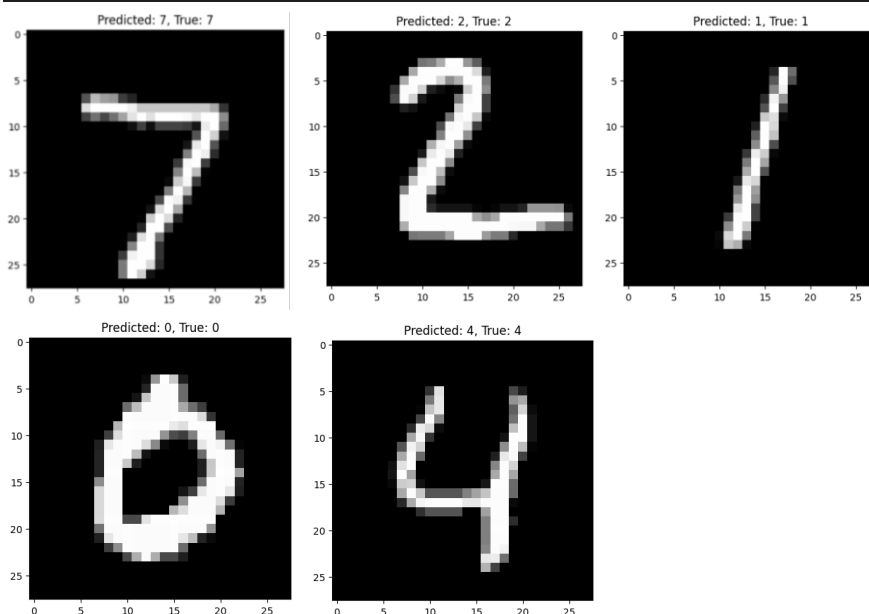
```

Kết quả

```

Epoch 1/5
1875/1875 ————— 9s 4ms/step - accuracy: 0.8613 - loss: 0.4829 - val_accuracy: 0.9582 - val_loss: 0.1392
Epoch 2/5
1875/1875 ————— 6s 3ms/step - accuracy: 0.9555 - loss: 0.1483 - val_accuracy: 0.9702 - val_loss: 0.1012
Epoch 3/5
1875/1875 ————— 6s 3ms/step - accuracy: 0.9669 - loss: 0.1085 - val_accuracy: 0.9741 - val_loss: 0.0856
Epoch 4/5
1875/1875 ————— 10s 3ms/step - accuracy: 0.9732 - loss: 0.0889 - val_accuracy: 0.9753 - val_loss: 0.0796
Epoch 5/5
1875/1875 ————— 6s 3ms/step - accuracy: 0.9775 - loss: 0.0746 - val_accuracy: 0.9775 - val_loss: 0.0707
313/313 ————— 1s 2ms/step - accuracy: 0.9731 - loss: 0.0837
Test accuracy: 0.9775
1/1 ————— 0s 66ms/step

```



2.3. Mạng nơ-ron tích chập (Convolutional Neural Network – CNN)

1. Khái niệm

Mạng nơ-ron tích chập (CNN) là một loại mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc dạng lưới, chẳng hạn như ảnh (hình vuông hoặc hình chữ nhật). CNN chủ yếu được sử dụng trong các bài toán thị giác máy tính (computer vision) như nhận dạng hình ảnh, phân loại ảnh, phát hiện vật thể, v.v.

2. Cấu trúc chính của CNN

CNN có ba thành phần chính:

- Lớp tích chập (Convolutional Layer): Dùng filter (hoặc kernel) để tìm kiếm các đặc trưng (features) như các cạnh, góc, hình dạng trong ảnh.
- Lớp pooling: Giảm kích thước của ảnh (down-sampling), giúp giảm độ phức tạp tính toán và tránh overfitting.
- Lớp fully connected: Dùng để phân loại kết quả sau khi đã trích xuất đặc trưng.

3. Cơ chế hoạt động của CNN

- Lớp tích chập (Convolution): Dùng filter để trượt qua ảnh, tính toán tích chập tại mỗi điểm, giúp nhận diện các đặc trưng (ví dụ: cạnh, đường chéo).
- Lớp kích hoạt (Activation): Hàm ReLU (Rectified Linear Unit) thường được dùng để thêm tính phi tuyến vào mô hình.
- Lớp pooling: Thường sử dụng max pooling để giảm kích thước ảnh, giữ lại các đặc trưng quan trọng. Ví dụ: chia ảnh thành các ô nhỏ và lấy giá trị lớn nhất trong mỗi ô.
- Lớp fully connected: Sau khi ảnh đã qua nhiều lớp tích chập và pooling, dữ liệu được chuyển qua các lớp fully connected để phân loại (classification).

4. Ứng dụng

- Nhận dạng hình ảnh: Phân loại ảnh (ví dụ: phân loại chó, mèo).
- Phát hiện vật thể: Tìm các vật thể trong ảnh (ví dụ: phát hiện mặt người).
- Nhận dạng chữ viết tay: Sử dụng trong các bài toán như phân loại chữ số viết tay (MNIST).
- Xử lý video: Phân tích video để nhận diện hành động.

5. Code Demo: Sử dụng mạng nơ-ron tích chập (CNN) cho một bài toán phân loại ảnh sử dụng bộ dữ liệu Fashion MNIST

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
# Tải bộ dữ liệu Fashion MNIST
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
# Tiền xử lý dữ liệu
```

```

x_train = x_train.astype('float32') / 255.0 # Chuẩn hóa dữ liệu thành 0-1
x_test = x_test.astype('float32') / 255.0
# Định dạng lại dữ liệu thành (batch_size, height, width, channels)
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
# Chuyển đổi nhãn sang dạng one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Xây dựng mô hình CNN
model = models.Sequential()
# Lớp Conv2D với 32 bộ lọc và kích thước bộ lọc 3x3
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
# Lớp Conv2D với 64 bộ lọc và kích thước bộ lọc 3x3
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# Lớp Conv2D với 128 bộ lọc và kích thước bộ lọc 3x3
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# Lớp Flatten để chuyển dữ liệu 2D thành 1D
model.add(layers.Flatten())
# Lớp Dense với 128 neuron
model.add(layers.Dense(128, activation='relu'))
# Lớp đầu ra với 10 neuron cho 10 lớp (sản phẩm thời trang)
model.add(layers.Dense(10, activation='softmax'))
# Biên dịch mô hình
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Huấn luyện mô hình
model.fit(x_train, y_train, epochs=10, batch_size=128,
validation_data=(x_test, y_test))
# Đánh giá mô hình trên bộ dữ liệu test
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Accuracy on test data: {test_acc}")
# Hiển thị một số hình ảnh trong bộ dữ liệu test
fig, axes = plt.subplots(1, 5, figsize=(15, 15))
for i in range(5):
    ax = axes[i]

```

```
ax.imshow(x_test[i].reshape(28, 28), cmap='gray')
ax.set_title(f'Label: {y_test[i].argmax()}')
ax.axis('off')

plt.show()
```

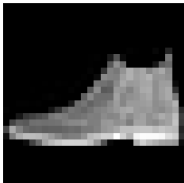
Kết quả

```


Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 — 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 — 3s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 — 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 — 1s 0us/step
c:\Users\lenovo\Downloads\AI\code\newenv\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
469/469 — 16s 30ms/step - accuracy: 0.6473 - loss: 0.9799 - val_accuracy: 0.8065 - val_loss: 0.5235
Epoch 2/10
469/469 — 14s 30ms/step - accuracy: 0.8315 - loss: 0.4670 - val_accuracy: 0.8463 - val_loss: 0.4260
Epoch 3/10
469/469 — 14s 30ms/step - accuracy: 0.8515 - loss: 0.4048 - val_accuracy: 0.8622 - val_loss: 0.3849
Epoch 4/10
469/469 — 15s 31ms/step - accuracy: 0.8666 - loss: 0.3641 - val_accuracy: 0.8614 - val_loss: 0.3804
Epoch 5/10
469/469 — 14s 29ms/step - accuracy: 0.8792 - loss: 0.3299 - val_accuracy: 0.8724 - val_loss: 0.3508
Epoch 6/10
469/469 — 14s 29ms/step - accuracy: 0.8868 - loss: 0.3064 - val_accuracy: 0.8750 - val_loss: 0.3375
Epoch 7/10
469/469 — 14s 30ms/step - accuracy: 0.8949 - loss: 0.2844 - val_accuracy: 0.8713 - val_loss: 0.3459
Epoch 8/10
469/469 — 14s 30ms/step - accuracy: 0.8999 - loss: 0.2698 - val_accuracy: 0.8859 - val_loss: 0.3114
Epoch 9/10
469/469 — 14s 30ms/step - accuracy: 0.9070 - loss: 0.2535 - val_accuracy: 0.8848 - val_loss: 0.3214
Epoch 10/10
469/469 — 21s 30ms/step - accuracy: 0.9106 - loss: 0.2388 - val_accuracy: 0.8920 - val_loss: 0.2962
313/313 — 2s 6ms/step - accuracy: 0.8908 - loss: 0.3023
Accuracy on test data: 0.8920000195503235

```

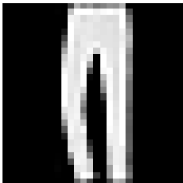
Label: 9



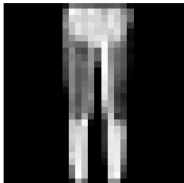
Label: 2




Label: 1



Label: 1



Label: 6



2.4. Mạng nơ-ron hồi tiếp (Recurrent Neural Network – RNN)

1. Khái niệm cơ bản

Mạng Nơ-Ron Hồi Tiếp (RNN) là một loại mạng nơ-ron được thiết kế để xử lý dữ liệu chuỗi (sequential data), như văn bản, âm thanh, hay chuỗi thời gian. Khác với mạng nơ-ron truyền thống, RNN có khả năng lưu trữ thông tin từ các bước trước thông qua trạng thái ẩn (hidden state), giúp xử lý các dữ liệu có tính chất thứ tự và phụ thuộc vào nhau.

2. Cấu trúc RNN

- Lớp đầu vào (Input Layer): Nhận dữ liệu theo từng bước trong chuỗi (ví dụ, từ văn bản, video, chuỗi thời gian).
- Lớp ẩn (Hidden Layer): Tại mỗi thời điểm, nơ-ron trong lớp ẩn không chỉ nhận đầu vào từ dữ liệu tại thời điểm đó mà còn từ trạng thái ẩn (hidden state) ở thời điểm trước.
- Lớp đầu ra (Output Layer): Trả về dự đoán cho mỗi bước thời gian hoặc sau khi xử lý toàn bộ chuỗi.

Công thức tính toán:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

Trong đó:

- h_t : Trạng thái ẩn tại thời điểm t
- W_{hh}, W_{hx} : Ma trận trọng số
- x_t : Đầu vào tại thời điểm t
- b : Bias

3. Các vấn đề trong RNN

- Vanishing Gradient: Gradients quá nhỏ khi huấn luyện qua nhiều bước, khiến mạng không học được hiệu quả.
- Exploding Gradient: Gradients quá lớn gây mất ổn định trong huấn luyện.

4. Các biến thể của RNN

- LSTM (Long Short-Term Memory): Khắc phục vấn đề vanishing gradient bằng cách duy trì trạng thái dài hạn (cell state).
- GRU (Gated Recurrent Unit): Phiên bản đơn giản hơn của LSTM, sử dụng ít tham số hơn.

5. Ứng dụng của RNN

- Xử lý ngôn ngữ tự nhiên (NLP): Dịch máy, chatbot, phân tích cảm xúc.
- Dự đoán chuỗi thời gian: Dự báo giá cổ phiếu, nhiệt độ.
- Nhận diện giọng nói: Chuyển từ giọng nói thành văn bản.
- Phân loại video: Dự đoán hành động trong video.

6. Code demo: Sử dụng Keras để xây dựng một mô hình RNN cho bài toán phân loại văn bản (text classification):

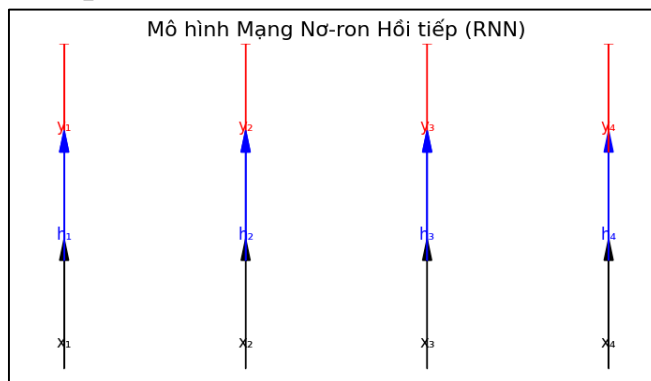
```
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
# Tạo hình ảnh minh họa cho một mạng RNN đơn giản
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
# Vẽ các bước thời gian và các chuỗi dữ liệu
input_sequence = ['x1', 'x2', 'x3', 'x4']
hidden_states = ['h1', 'h2', 'h3', 'h4']
output_sequence = ['y1', 'y2', 'y3', 'y4']
# Vẽ các mũi tên biểu diễn quá trình truyền thông tin
for i in range(len(input_sequence)):
    ax.arrow(i, 0, 0, 0.5, head_width=0.05, head_length=0.1, fc='black',
ec='black')
    ax.text(i, 0.1, input_sequence[i], ha='center', fontsize=12, color='black')
for i in range(len(hidden_states)):
    ax.arrow(i, 0.5, i, 0.5, head_width=0.05, head_length=0.1, fc='black',
ec='black')
    ax.text(i, 0.6, hidden_states[i], ha='center', fontsize=12, color='black')
```

```

ax.arrow(i, 0.5, 0, 0.5, head_width=0.05, head_length=0.1, fc='blue',
ec='blue')
ax.text(i, 0.6, hidden_states[i], ha='center', fontsize=12, color='blue')
for i in range(len(output_sequence)):
    ax.arrow(i, 1, 0, 0.5, head_width=0.05, head_length=0.1, fc='red',
ec='red')
    ax.text(i, 1.1, output_sequence[i], ha='center', fontsize=12, color='red')
# Vẽ mũi tên nối trạng thái ẩn và các đầu ra
for i in range(1, len(hidden_states)):
    ax.arrow(i - 1, 0.5, 0, 0.5, head_width=0.05, head_length=0.1, fc='blue',
ec='blue')
# Đặt tiêu đề và các nhãn
ax.set_title("Mô hình Mạng Nơ-ron Hồi tiếp (RNN)", fontsize=16)
ax.set_xlim(-0.5, 3.5)
ax.set_ylim(-0.5, 1.5)
ax.axis('off')
# Hiển thị hình ảnh
plt.show()

```

Kết quả:



Giải thích:

1. Input Sequence (Chuỗi đầu vào):

Các giá trị đầu vào (x_1, x_2, x_3, x_4) là những thông tin mà RNN sẽ nhận và xử lý theo thứ tự. Đây có thể là các từ trong văn bản, các giá trị trong chuỗi thời gian, v.v.

2. Hidden States (Trạng thái ẩn):

Các trạng thái ẩn (h_1, h_2, h_3, h_4) là những thông tin được RNN lưu trữ từ bước thời gian trước đó, và chúng được cập nhật sau mỗi bước. Mỗi trạng thái ẩn phụ thuộc vào giá trị đầu vào của bước hiện tại và trạng thái ẩn của bước trước đó.

3. Output Sequence (Đầu ra):

Các giá trị đầu ra (y_1, y_2, y_3, y_4) được sinh ra từ trạng thái ẩn của mỗi bước. Đây là kết quả mà mô hình trả về sau khi xử lý chuỗi đầu vào.

4. Quá trình Truyền Thông Tin:

Các mũi tên chỉ ra quá trình thông tin được truyền từ các bước trước đó (input và hidden states) để tạo ra kết quả đầu ra.

Kết luận:

- RNN là một mô hình mạnh mẽ cho các bài toán xử lý chuỗi, và nó có khả năng "nhớ" thông tin từ các bước trước để ra quyết định cho các bước sau.
- Trong hình, chúng ta thấy rõ cách mà thông tin được truyền qua các bước thời gian, với sự thay đổi của trạng thái ẩn và cách mà đầu ra được tạo ra.

2.5. Mạng sinh đối kháng (Generative Adversarial Network – GAN)

1. Khái niệm

Generative Adversarial Networks (GANs) là một kiến trúc học sâu gồm hai mạng nơ-ron đối kháng:

- Generator (G): Tạo dữ liệu giả (ví dụ: ảnh).
- Discriminator (D): Phân biệt dữ liệu thật và giả.

Quá trình huấn luyện:

- Generator cố gắng đánh lừa Discriminator tạo ra dữ liệu giả giống thật.
- Discriminator cố gắng phân biệt dữ liệu thật và giả.

2. Cách thức hoạt động

- Generator nhận đầu vào ngẫu nhiên (noise) và tạo dữ liệu giả.
- Discriminator nhận dữ liệu thật và giả, phân loại chúng là "thật" (1) hay "giả" (0).
- Mục tiêu: Cả hai mạng cải thiện lẫn nhau qua việc tối ưu hóa hàm mất mát.

3. Hàm mất mát

- Generator: Tối đa hóa khả năng đánh lừa Discriminator (tạo dữ liệu giả).
- Discriminator: Tối đa hóa khả năng phân biệt đúng dữ liệu thật và giả.

4. Ứng dụng thực tế

- Tạo ảnh giả (DeepFake, nghệ thuật AI).
- Tạo dữ liệu huấn luyện cho các mô hình khác.
- Tạo âm thanh giả, văn bản tự động.

5. Code Demo: Sử dụng TensorFlow và Keras để huấn luyện mô hình tạo hình ảnh giả từ bộ dữ liệu MNIST (các chữ số viết tay)

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
```

```

import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
# Tải bộ dữ liệu MNIST
(x_train, _), (_, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
# Định nghĩa kích thước cho nhiều đầu vào của Generator
latent_dim = 100
# Xây dựng Generator
def build_generator():
    model = models.Sequential()
    model.add(layers.Dense(128, input_dim=latent_dim))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dense(784, activation='sigmoid'))
    model.add(layers.Reshape((28, 28, 1)))
    return model
# Xây dựng Discriminator
def build_discriminator():
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28, 1)))
    model.add(layers.Dense(128))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
# Biên dịch và xây dựng GAN
def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = models.Sequential()
    model.add(generator)
    model.add(discriminator)
    return model
# Tạo các mô hình
generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)
# Biên dịch Discriminator
discriminator.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Biên dịch GAN (chỉ tối ưu hóa Generator)

```

```

gan.compile(optimizer='adam', loss='binary_crossentropy')
# Huấn luyện GAN
def train_gan(epochs, batch_size=128):
    half_batch = batch_size // 2
    # Tạo nhãn cho dữ liệu thật (1) và giả (0)
    valid = np.ones((half_batch, 1))
    fake = np.zeros((half_batch, 1))
    for epoch in range(epochs):
        # Chọn ngẫu nhiên một nửa dữ liệu huấn luyện để huấn luyện
        Discriminator
        idx = np.random.randint(0, x_train.shape[0], half_batch)
        imgs = x_train[idx]
        # Sinh dữ liệu giả từ Generator
        noise = np.random.normal(0, 1, (half_batch, latent_dim))
        gen_imgs = generator.predict(noise)
        # Huấn luyện Discriminator
        d_loss_real = discriminator.train_on_batch(imgs, valid)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        # Huấn luyện Generator (khi huấn luyện GAN, Discriminator được giữ cố định)
        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        y_gen = np.ones((batch_size, 1)) # Generator muốn đánh lừa
        Discriminator
        g_loss = gan.train_on_batch(noise, y_gen)
        # In kết quả mỗi 100 epoch
        if epoch % 100 == 0:
            print(f'{epoch} [D loss: {d_loss[0]} | D accuracy: {100*d_loss[1]}]
            [G loss: {g_loss}]')
            # Hiển thị hình ảnh tạo ra mỗi 100 epoch
            if epoch % 1000 == 0:
                plot_generated_images(epoch)
# Hàm hiển thị hình ảnh tạo ra
def plot_generated_images(epoch, examples=10, dim=(1, 10), figsize=(10, 1)):
    noise = np.random.normal(0, 1, (examples, latent_dim)) # Tạo nhiễu ngẫu nhiên
    gen_imgs = generator.predict(noise) # Sinh ảnh từ Generator
    # Chuyển giá trị đầu ra từ [0, 1] sang [0, 1] cho việc hiển thị

```

```

gen_imgs = 0.5 * gen_imgs + 0.5
# Hiển thị các hình ảnh sinh ra
plt.figure(figsize=figsize)
for i in range(examples):
    plt.subplot(dim[0], dim[1], i + 1)
    plt.imshow(gen_imgs[i, :, :, 0], cmap='gray') # Hiển thị ảnh tạo ra
    plt.axis('off')
# Đảm bảo không có sự chồng chéo giữa các ảnh
plt.tight_layout()
# Lưu lại hình ảnh
plt.savefig(f"gan_generated_image_epoch_{epoch}.png")
plt.close()
# Hiển thị hình ảnh trong notebook (hoặc môi trường tương tự)
plt.show() # Hiển thị hình ảnh trực tiếp trong Jupyter Notebook hoặc môi
trường hỗ trợ
# Đảm bảo huấn luyện được bắt đầu
train_gan(epochs=1000, batch_size=64)

```

Kết quả

```

1/1 ————— 0s 77ms/step
0 [D loss: 0.7806081771850586 | D accuracy: 44.53125] [G loss: 0.4081157147884369]
1/1 ————— 0s 68ms/step
1/1 ————— 0s 36ms/step
1/1 ————— 0s 25ms/step
1/1 ————— 0s 31ms/step
1/1 ————— 0s 31ms/step
1/1 ————— 0s 24ms/step
1/1 ————— 0s 34ms/step
1/1 ————— 0s 44ms/step
1/1 ————— 0s 44ms/step
1/1 ————— 0s 39ms/step
1/1 ————— 0s 25ms/step
1/1 ————— 0s 40ms/step
1/1 ————— 0s 40ms/step
1/1 ————— 0s 36ms/step
1/1 ————— 0s 27ms/step
1/1 ————— 0s 34ms/step
1/1 ————— 0s 30ms/step
1/1 ————— 0s 21ms/step
1/1 ————— 0s 22ms/step
1/1 ————— 0s 23ms/step
1/1 ————— 0s 34ms/step
1/1 ————— 0s 44ms/step
1/1 ————— 0s 32ms/step
...
1/1 ————— 0s 41ms/step
1/1 ————— 0s 45ms/step
1000 [D loss: 4.528819561004639 | D accuracy: 28.085187911987305] [G loss: 0.0025916281156241894]
1/1 ————— 0s 54ms/step

```

Hình ảnh 1. gan_generated_image_epoch_0.png



Hình ảnh 2. gan_generated_image_epoch_1000.png



Giải thích:

1. Generator:

- Nhận một vector nhiễu ngẫu nhiên (kích thước $\text{latent_dim} = 100$) và tạo ra một hình ảnh giả có kích thước 28×28 .
- Sử dụng LeakyReLU để tăng cường quá trình huấn luyện và sigmoid ở lớp cuối để đảm bảo rằng giá trị đầu ra nằm trong phạm vi $[0, 1]$.

2. Discriminator:

- Nhận đầu vào là hình ảnh và phân loại nó là thật (1) hay giả (0).
- Sử dụng LeakyReLU và sigmoid để phân loại ảnh thật và giả.

3. GAN:

- Discriminator bị khóa khi huấn luyện Generator qua GAN, vì chúng ta chỉ muốn tối ưu hóa Generator khi huấn luyện GAN.
- Generator được huấn luyện để "đánh lừa" Discriminator.

4. Huấn luyện:

- Mỗi 100 epoch, bạn sẽ thấy các kết quả từ Discriminator loss và Generator loss. Sau mỗi 1000 epoch, hình ảnh được sinh ra từ Generator sẽ được hiển thị và lưu lại.

Kết luận:

- GAN là một mô hình rất mạnh mẽ và có khả năng sinh ra các mẫu dữ liệu mới.
- Ứng dụng của GAN rất rộng rãi trong việc tạo ra hình ảnh giả, video, âm thanh, và thậm chí văn bản.

2.6. Mạng nơ-ron Transformer

1. Khái niệm

Transformer là một mô hình học sâu mạnh mẽ, được phát triển để xử lý dữ liệu chuỗi, đặc biệt là trong các bài toán xử lý ngôn ngữ tự nhiên (NLP). Khác với RNN hay LSTM, Transformer không sử dụng cơ chế hồi tiếp mà thay vào đó dựa vào cơ chế Attention, cho phép xử lý các chuỗi song song, giảm thiểu độ phức tạp tính toán.

2. Cấu trúc chính

Mô hình Transformer gồm hai phần chính:

- Encoder: Mã hóa dữ liệu đầu vào thành các vector đại diện.
- Decoder: Sử dụng các vector này để tạo ra kết quả đầu ra.

Cả Encoder và Decoder đều bao gồm các lớp Self-Attention và Feed Forward Neural Networks.

3. Cơ chế Attention

Cơ chế Self-Attention giúp mỗi từ trong câu chú ý đến tất cả các từ khác để hiểu mối quan hệ giữa chúng. Cụ thể:

- Query: Từ cần được chú ý.
- Key: Các từ mà Query có thể chú ý đến.
- Value: Nội dung thực sự mà mỗi từ mang lại khi được chú ý.

Công thức tính Attention:

4. Ứng dụng

- Dịch máy: Như trong mô hình Google Translate.
- Tóm tắt văn bản: Tạo tóm tắt ngắn gọn từ văn bản dài.
- Hỏi đáp: Các mô hình như BERT và GPT có thể trả lời câu hỏi từ văn bản.
- Sinh văn bản: Mô hình GPT-3 có thể tạo ra văn bản tự nhiên rất giống con người.

5. Code demo: Mô hình Transformer đơn giản gồm encoder và decoder, áp dụng cho bài toán dịch máy song ngữ Anh–Pháp.

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
# ===== Dữ liệu toy =====
eng_sentences = ["hello", "how are you", "good morning"]
fra_sentences = ["<sos> bonjour <eos>", "<sos> comment ça va <eos>",
"<sos> bon matin <eos>"]
source_tokenizer = tf.keras.preprocessing.text.Tokenizer(filters="")
target_tokenizer = tf.keras.preprocessing.text.Tokenizer(filters="")
source_tokenizer.fit_on_texts(eng_sentences)
target_tokenizer.fit_on_texts(fra_sentences)
input_sequences = source_tokenizer.texts_to_sequences(eng_sentences)
target_sequences = target_tokenizer.texts_to_sequences(fra_sentences)
max_len_in = max(len(seq) for seq in input_sequences)
max_len_out = max(len(seq) for seq in target_sequences)
encoder_input =
tf.keras.preprocessing.sequence.pad_sequences(input_sequences,
maxlen=max_len_in, padding='post')
decoder_input =
tf.keras.preprocessing.sequence.pad_sequences(target_sequences,
maxlen=max_len_out, padding='post')
decoder_target = np.zeros_like(decoder_input)
decoder_target[:, :-1] = decoder_input[:, 1:]
# ===== Tham số =====
embed_dim = 64
latent_dim = 128
vocab_in = len(source_tokenizer.word_index) + 1
```

```

vocab_out = len(target_tokenizer.word_index) + 1
# ===== Positional Encoding (đã sửa) =====
class PositionalEncoding(layers.Layer):
    def __init__(self, maxlen, d_model):
        super().__init__()
        pos = np.arange(maxlen)[: , np.newaxis]
        i = np.arange(d_model)[np.newaxis, :]
        angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
        angle_rads = pos * angle_rates
        angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
        angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
        self.pos_encoding = tf.constant(angle_rads[np.newaxis, ...],
dtype=tf.float32)
    def call(self, x):
        seq_len = tf.shape(x)[1]
        return x + self.pos_encoding[:, :seq_len, :]
# ===== Encoder =====
def transformer_encoder(inputs):
    x = layers.Embedding(vocab_in, embed_dim)(inputs)
    x = PositionalEncoding(max_len_in, embed_dim)(x)
    x = layers.MultiHeadAttention(num_heads=2, key_dim=embed_dim)(x, x)
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    ff = layers.Dense(latent_dim, activation='relu')(x)
    x = layers.Dense(embed_dim)(ff)
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    return x
# ===== Decoder =====
def transformer_decoder(inputs, enc_output):
    x = layers.Embedding(vocab_out, embed_dim)(inputs)
    x = PositionalEncoding(max_len_out, embed_dim)(x)
    attn1 = layers.MultiHeadAttention(num_heads=2,
key_dim=embed_dim)(x, x)
    x = layers.LayerNormalization(epsilon=1e-6)(x + attn1)
    attn2 = layers.MultiHeadAttention(num_heads=2,
key_dim=embed_dim)(x, enc_output)
    x = layers.LayerNormalization(epsilon=1e-6)(x + attn2)
    ff = layers.Dense(latent_dim, activation='relu')(x)
    x = layers.Dense(embed_dim)(ff)
    x = layers.LayerNormalization(epsilon=1e-6)(x)

```

```

    return x

# ===== Mô hình tổng thể =====
enc_inputs = layers.Input(shape=(max_len_in,), name="encoder_input")
enc_output = transformer_encoder(enc_inputs)
dec_inputs = layers.Input(shape=(max_len_out,), name="decoder_input")
dec_output = transformer_decoder(dec_inputs, enc_output)
outputs = layers.Dense(vocab_out, activation='softmax')(dec_output)
model = tf.keras.Model([enc_inputs, dec_inputs], outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
# ===== Huấn luyện =====
model.fit([encoder_input, decoder_input], decoder_target[..., np.newaxis],
epochs=300, verbose=0)
# ===== Hàm dịch =====
def translate(sentence):
    seq = source_tokenizer.texts_to_sequences([sentence])
    seq = tf.keras.preprocessing.sequence.pad_sequences(seq,
maxlen=max_len_in, padding='post')
    target = [target_tokenizer.word_index["<sos>"]]
    result = []
    for _ in range(max_len_out):
        target_seq = tf.keras.preprocessing.sequence.pad_sequences([target],
maxlen=max_len_out, padding='post')
        preds = model.predict([seq, target_seq], verbose=0)
        next_id = np.argmax(preds[0, len(target) - 1])
        if next_id == target_tokenizer.word_index["<eos>"]:
            break
        result.append(next_id)
        target.append(next_id)
    return ' '.join([target_tokenizer.index_word[i] for i in result])
# ===== Kiểm thử =====
print("Dịch 'hello' ->", translate("hello"))
print("Dịch 'how are you' ->", translate("how are you"))
print("Dịch 'good morning' ->", translate("good morning"))

```

Kết quả:

```

Dịch 'hello' -> bonjour
Dịch 'how are you' -> comment ça va
Dịch 'good morning' -> bonjour

```


Giải thích cách hoạt động của GID:

Thành phần	Vai trò
Token hóa	Chuyển câu thành chuỗi số (index từ)
Embedding	Biểu diễn từ dưới dạng vector học được
Positional Encoding	Thêm thông tin về vị trí từ vào vector
Multi-head Attention	Cho phép mô hình nhìn từ nhiều góc độ của câu
Encoder	Biểu diễn câu đầu vào thành vector ngữ nghĩa
Decoder	Dự đoán từng từ đầu ra một cách tuần tự
Loop dịch	Bắt đầu bằng <sos> và kết thúc khi gặp <eos>

Kết luận:

Mô hình GID (Grammar-Infused Demo) là một ví dụ tối giản của Transformer dùng để dịch máy, giúp bạn:

1. Hiểu rõ kiến trúc Transformer từ đầu đến cuối
2. Thực hành xây dựng mô hình từ đầu thay vì chỉ dùng pre-trained
3. Thấy rõ vai trò của Attention, Positional Encoding, và quá trình sinh chuỗi

Mặc dù chỉ học trên vài câu toy, mô hình GID vẫn dịch được đầy đủ câu như "how are you" → "comment ủa va", minh chứng cho sức mạnh của kiến trúc Transformer—even in miniature.

2.7. Autoencoder

1. Khái niệm

Autoencoder là một loại mạng nơ-ron nhân tạo dùng để học mã hóa (encoding) và giải mã (decoding) dữ liệu. Mục tiêu chính của Autoencoder là nén dữ liệu đầu vào thành một biểu diễn có kích thước nhỏ hơn (gọi là latent space) và sau đó tái tạo lại dữ liệu gốc từ biểu diễn nén đó.

2. Cấu trúc Autoencoder

Autoencoder có 3 phần chính:

- Encoder: Mã hóa dữ liệu đầu vào thành một vector nhỏ hơn.
- Latent Space (Bottleneck): Biểu diễn dữ liệu trong không gian nén.
- Decoder: Giải mã từ latent space để tái tạo lại dữ liệu ban đầu.

3. Ứng dụng

- Giảm chiều dữ liệu (Dimensionality Reduction): Nén dữ liệu nhưng vẫn giữ lại các đặc trưng quan trọng.
- Dự đoán và tái tạo dữ liệu: Dự đoán giá trị bị thiếu, tái tạo hình ảnh hoặc âm thanh.

- Phát hiện bất thường (Anomaly Detection): So sánh sự khác biệt giữa dữ liệu đầu vào và dữ liệu tái tạo để phát hiện bất thường.

4. **Code Demo: Một kiến trúc mạng nơ-ron thường dùng để học đặc trưng ẩn (feature representation) và giảm chiều dữ liệu.**

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
# ===== 1. Tải dữ liệu ảnh MNIST =====
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.
x_test = x_test.astype("float32") / 255.
# Thêm chiều channel
x_train = x_train.reshape((len(x_train), 28, 28, 1))
x_test = x_test.reshape((len(x_test), 28, 28, 1))
# ===== 2. Xây dựng Autoencoder =====
input_img = layers.Input(shape=(28, 28, 1))
# Encoder
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)
# Decoder
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid',
padding='same')(x)
# Mô hình tổng thể
autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# ===== 3. Huấn luyện mô hình =====
autoencoder.fit(x_train, x_train,
                epochs=10,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test, x_test))
# ===== 4. Hiển thị kết quả =====
decoded_imgs = autoencoder.predict(x_test[:10])
```

```
plt.figure(figsize=(20, 4))
for i in range(10):
    # Ảnh gốc
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Gốc")
    plt.axis('off')
    # Ảnh tái tạo
    ax = plt.subplot(2, 10, i + 11)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Tái tạo")
    plt.axis('off')
plt.show()
```

Kết quả:



Mô hình hoạt động như thế nào?

- Ảnh chữ số 28x28 được nén lại thành không gian ẩn 7x7x8 → học ra đặc trưng.
- Sau đó được giải mã (decode) trở lại kích thước ban đầu 28x28.
- Mô hình học cách giữ lại thông tin quan trọng nhất để tái tạo lại ảnh.

Kết luận

Autoencoder là kỹ thuật nền tảng quan trọng trong:

- Giảm chiều dữ liệu

- Nén ảnh
- Phát hiện dị thường (anomaly detection)
- Tiền xử lý đầu vào cho mô hình khác

2.8. Học tăng cường sâu (Deep Reinforcement Learning – DRL)

1. Khái niệm

Học tăng cường sâu (Deep Reinforcement Learning - DRL) là sự kết hợp giữa học sâu (deep learning) và học tăng cường (reinforcement learning - RL). Trong DRL, các mô hình học sâu được sử dụng để học cách đưa ra quyết định dựa trên các tương tác với môi trường và phản hồi từ môi trường đó (thưởng hoặc phạt).

2. Thành phần chính

- Agent: Hệ thống đưa ra quyết định.
- Môi trường (Environment): Bối cảnh mà agent hoạt động.
- Hành động (Action): Các hành động mà agent có thể thực hiện.
- Trạng thái (State): Tình huống mà agent gặp phải.
- Thưởng (Reward): Phản hồi từ môi trường sau mỗi hành động.
- Chính sách (Policy): Hàm quyết định hành động dựa trên trạng thái.
- Giá trị (Value): Đo lường "tốt" của trạng thái/hành động.

3. Quá trình DRL

- Khám phá: Agent thử nghiệm hành động để học hỏi.
- Khai thác: Agent sử dụng kiến thức đã học để tối đa hóa thưởng.
- Học: Agent cập nhật chính sách dựa trên phản hồi từ môi trường.

4. Thuật toán DRL

- Q-learning: Học tối ưu giá trị hành động qua Q-table.
- DQN (Deep Q-Network): Kết hợp Q-learning và mạng nơ-ron để dự đoán giá trị Q.
- Policy Gradient: Học trực tiếp chính sách mà không cần bảng Q.
- PPO (Proximal Policy Optimization): Thuật toán tối ưu chính sách hiệu quả.
- A3C (Asynchronous Advantage Actor-Critic): Dùng nhiều agent để cải thiện hiệu suất.

5. Ứng dụng

- Game: AlphaGo, OpenAI Five.
- Robotics: Di chuyển và thao tác.
- Tối ưu hóa tài chính: Đầu tư và dự đoán thị trường.
- Xe tự lái: Lái xe tự động.
- Quản lý năng lượng: Tối ưu hóa sử dụng năng lượng.

6. Code Demo: Trực quan hóa kỹ thuật Deep Q-Learning (DQN) áp dụng vào môi trường CartPole-v1.

```

import gym
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from collections import deque
import random
import matplotlib.pyplot as plt
# ===== 1. Tạo môi trường với render_mode để lấy hình =====
env = gym.make("CartPole-v1", render_mode="rgb_array")
state_shape = env.observation_space.shape[0]
n_actions = env.action_space.n
# ===== 2. Xây dựng Q-network =====
def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(state_shape,)),
        layers.Dense(24, activation='relu'),
        layers.Dense(24, activation='relu'),
        layers.Dense(n_actions, activation='linear')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
model = build_model()
target_model = build_model()
target_model.set_weights(model.get_weights())
# ===== 3. Tham số huấn luyện =====
gamma = 0.99
epsilon = 1.0
epsilon_min = 0.01
epsilon_decay = 0.995
batch_size = 64
memory = deque(maxlen=2000)
# ===== 4. Hàm chọn hành động =====
def act(state):
    if np.random.rand() < epsilon:
        return np.random.choice(n_actions)
    q_values = model.predict(state[np.newaxis], verbose=0)
    return np.argmax(q_values[0])
# ===== 5. Huấn luyện với replay buffer =====
def train():

```

```

if len(memory) < batch_size:
    return
minibatch = random.sample(memory, batch_size)
for state, action, reward, next_state, done in minibatch:
    target = reward
    if not done:
        target += gamma *
np.max(target_model.predict(next_state[np.newaxis], verbose=0)[0])
    target_q = model.predict(state[np.newaxis], verbose=0)
    target_q[0][action] = target
    model.fit(state[np.newaxis], target_q, epochs=1, verbose=0)
    target_model.set_weights(model.get_weights())
# ===== 6. Vòng lặp huấn luyện chính =====
episodes = 3
for ep in range(episodes):
    state = env.reset()[0]
    total_reward = 0
    done = False
    while not done:
        action = act(state)
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        memory.append((state, action, reward, next_state, done))
        state = next_state
        total_reward += reward
        train()
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
    print(f"Tập {ep+1}: Tổng thưởng = {total_reward:.2f} | Epsilon = {epsilon:.2f}")
# ===== 7. Hiện thị hình ảnh sau khi học xong =====
print("\n🎮 Đang chạy một tập để hiển thị ảnh:")
frames = []
state = env.reset()[0]
done = False
while not done:
    frame = env.render()
    frames.append(frame)
    action = act(state)

```

```

next_state, _, terminated, truncated, _ = env.step(action)
done = terminated or truncated
state = next_state
env.close()
# ===== 8. Hiển thị các khung hình (ảnh động đơn giản) =====
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
for i in range(min(20, len(frames))):
    plt.imshow(frames[i])
    plt.title(f"Bước {i+1}")
    plt.axis('off')
    plt.pause(0.2)
plt.show()

```

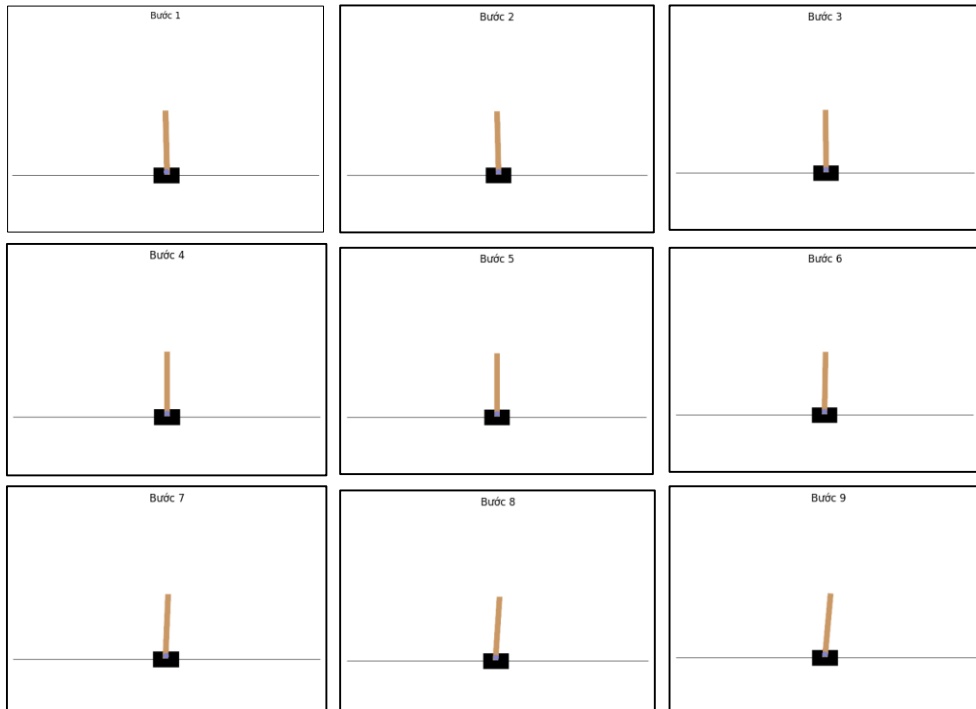
Kết quả:

```

Tập 1: Tổng thưởng = 13.00 | Epsilon = 0.99
Tập 2: Tổng thưởng = 39.00 | Epsilon = 0.99
Tập 3: Tổng thưởng = 17.00 | Epsilon = 0.99

🖼️ Đang chạy một tập để hiển thị ảnh:

```



Code hoạt động:

1. Huấn luyện agent bằng thuật toán DQN:

Agent dùng mạng nơ-ron để học chọn hành động tốt nhất từ trạng thái đầu vào.

- Trải nghiệm được lưu lại trong memory, dùng cho experience replay.
- Giảm dần xác suất hành động ngẫu nhiên (epsilon):
Từ 1.0 xuống 0.01 → giúp agent chuyển từ khám phá sang khai thác chính sách đã học.
 - Sau khi học xong:
Agent chơi lại một tập (episode) mới, và lưu từng khung hình (frame) của môi trường.
 - Hiển thị ảnh động:
Dùng matplotlib.pyplot để hiện ảnh từng bước như một GIF đơn giản (nhưng hiển thị trực tiếp chứ không tạo file .gif).

Kết luận:

Code trên minh họa kỹ thuật Deep Q-Learning (DQN) giúp agent học giữ cột thăng bằng trong môi trường CartPole-v1. Sau quá trình huấn luyện, agent có thể tự chơi tốt và kết quả được hiển thị bằng ảnh động. Đây là ví dụ đơn giản nhưng hiệu quả của học tăng cường sâu (DRL).

2.9. Bảng so sánh giữa các kỹ thuật học sâu phổ biến

Mạng / Kiến trúc	Ứng dụng chính	Ưu điểm	Nhược điểm	Yêu cầu tài nguyên
ANN (Artificial Neural Network)	<ul style="list-style-type: none"> - Phân loại cơ bản - Dự đoán tài chính - Nhận dạng mẫu đơn giản 	<ul style="list-style-type: none"> - Cấu trúc đơn giản, dễ hiểu - Áp dụng tốt với dữ liệu không gian thấp 	<ul style="list-style-type: none"> - Không hiệu quả với dữ liệu có cấu trúc phức tạp (hình ảnh, chuỗi) - Khó mở rộng cho dữ liệu lớn 	<ul style="list-style-type: none"> - Trung bình, phù hợp với GPU cơ bản
CNN (Convolutional Neural Network)	<ul style="list-style-type: none"> - Thị giác máy tính (ảnh, video) - Nhận diện vật thể, phân loại ảnh - Nhận 	<ul style="list-style-type: none"> - Tự động trích xuất đặc trưng không gian - Hiệu quả cao với dữ liệu hình ảnh 	<ul style="list-style-type: none"> - Không xử lý tốt dữ liệu chuỗi hoặc thứ tự thời gian - Yêu cầu lượng dữ liệu lớn để tránh overfitting 	<ul style="list-style-type: none"> - Cao, cần GPU mạnh để huấn luyện mô hình lớn

	dạng chữ viết tay			
RNN (Recurrent Neural Network) và biến thể LSTM, GRU	<ul style="list-style-type: none"> - Xử lý dữ liệu chuỗi (NLP, âm thanh) - Dự báo chuỗi thời gian - Nhận dạng giọng nói 	<ul style="list-style-type: none"> - Lưu trữ thông tin trạng thái ẩn để xử lý chuỗi - Xử lý dữ liệu có tính thứ tự tốt 	<ul style="list-style-type: none"> - Dễ bị vanishing/exploding gradient - Huấn luyện chậm do tính tuần tự - Khó học các phụ thuộc dài hạn 	<ul style="list-style-type: none"> - Cao, yêu cầu tài nguyên tính toán và bộ nhớ lớn
Transformer	<ul style="list-style-type: none"> - Dịch máy, NLP nâng cao - Sinh văn bản, hỏi đáp - Tóm tắt văn bản - Thị giác máy tính (phiên bản mở rộng) 	<ul style="list-style-type: none"> - Xử lý song song, tăng tốc huấn luyện - Cơ chế Attention giúp mô hình hiểu ngữ cảnh dài hạn - Hiệu suất rất cao trong NLP 	<ul style="list-style-type: none"> - Yêu cầu bộ nhớ rất lớn - Mô hình phức tạp, khó tối ưu - Cần dữ liệu lớn để đạt hiệu quả cao 	<ul style="list-style-type: none"> - Rất cao, cần GPU/TPU chuyên dụng, bộ nhớ lớn

Giải thích ngắn gọn:

- **ANN** là mô hình nền tảng, phù hợp với các bài toán đơn giản hoặc dữ liệu dạng vector phẳng.
- **CNN** là chuẩn mực trong xử lý ảnh, có khả năng tự động trích xuất đặc trưng không gian, rất mạnh trong các bài toán thị giác.
- **RNN** và các biến thể (LSTM, GRU) được thiết kế để xử lý dữ liệu chuỗi có phụ thuộc thời gian nhưng gặp khó khăn trong học các mối quan hệ dài hạn.
- **Transformer** là bước tiến lớn trong học sâu, đặc biệt trong NLP, với cơ chế Attention cho phép xử lý chuỗi song song, giúp mô hình học hiệu quả hơn và xử lý ngữ cảnh dài hạn rất tốt, tuy nhiên tốn tài nguyên.

Chương 3: Triển Khai Ứng Dụng Học Sâu

3.1. Mục tiêu triển khai

Triển khai ứng dụng học sâu là bước đưa mô hình từ môi trường nghiên cứu sang thực tế để phục vụ người dùng cuối. Mục tiêu của việc triển khai bao gồm:

- Đưa mô hình học sâu vào môi trường thực tế để phục vụ các tác vụ như dự đoán, phân loại, nhận diện, sửa lỗi...
- Tạo ra giao diện để người dùng có thể nhập dữ liệu đầu vào và nhận kết quả đầu ra từ mô hình.
- Đảm bảo khả năng mở rộng, bảo trì và hoạt động ổn định của hệ thống khi triển khai thực tế.

3.2. Quy trình triển khai ứng dụng học sâu

3.2.1. Chuẩn bị mô hình

- Huấn luyện mô hình sử dụng thư viện học sâu như TensorFlow, Keras, hoặc PyTorch.
- Đánh giá mô hình về độ chính xác, loss, precision, recall... với tập test/validation.
- Lưu mô hình đã huấn luyện ra tệp (ví dụ .h5, .pth, .pkl, hoặc .joblib) để phục vụ inference (dự đoán).

3.2.2. Tạo API hoặc giao diện web

- Đọc mô hình đã lưu và nạp vào bộ nhớ (sử dụng load_model, torch.load...).
- Tạo API với FastAPI hoặc Flask có các endpoint nhận dữ liệu và trả kết quả dự đoán.
- Xây dựng giao diện web đơn giản bằng HTML/CSS/JS hoặc React/Vue để người dùng nhập dữ liệu.
- Kết nối frontend ↔ backend thông qua gọi HTTP (fetch, axios).

3.3. Minh họa mô hình ứng dụng cụ thể

3.3.1. Ứng dụng dự đoán địa danh bằng hình ảnh

1. Mục đích

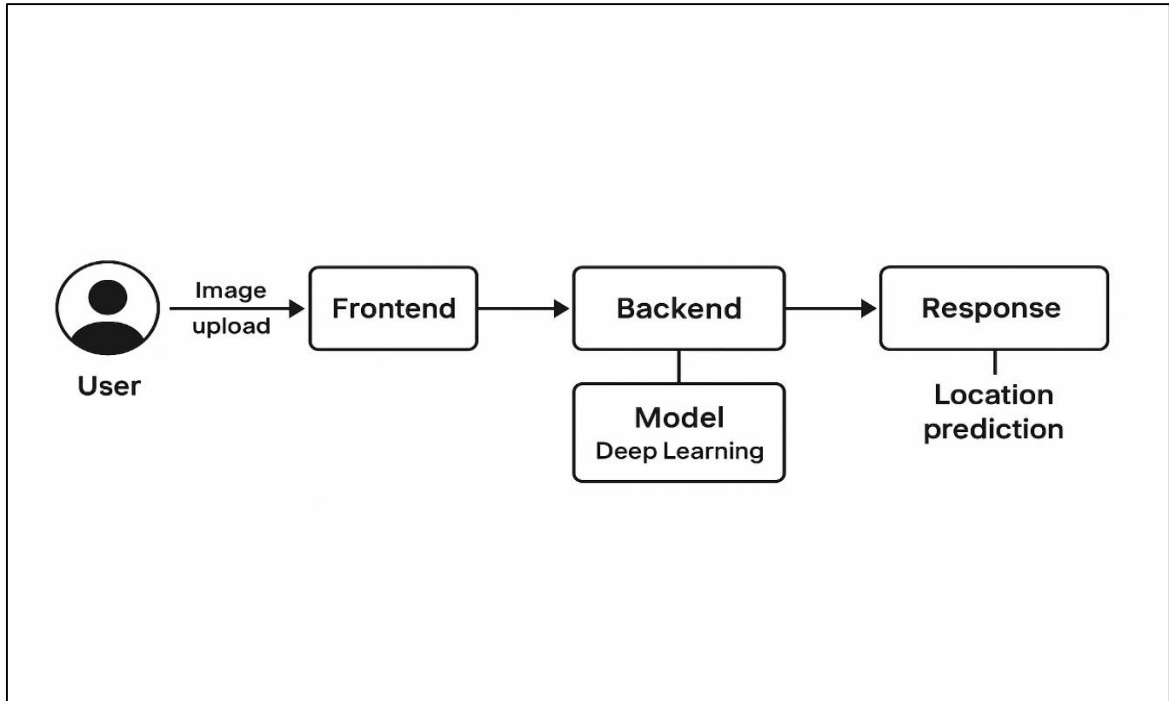
Ứng dụng nhằm xây dựng một hệ thống dự đoán và phân loại địa danh du lịch dựa trên ảnh đầu vào của người dùng. Người dùng có thể tải lên một hình ảnh liên quan đến các địa danh nổi tiếng (ví dụ: hình ảnh cảnh quan ở Sapa, Hạ Long, Đà Lạt) và hệ thống sẽ phân tích, nhận diện ảnh đó thuộc địa danh nào với độ chính xác cao.

Mục tiêu cụ thể:

- Tự động nhận diện địa danh từ hình ảnh mà không cần người dùng nhập văn bản.
- Hỗ trợ du khách và người dùng trong việc tra cứu thông tin du lịch nhanh chóng, thuận tiện.

- Minh họa ứng dụng thực tế của kỹ thuật học sâu (deep learning) trong lĩnh vực thị giác máy tính (computer vision).
- Tạo nền tảng để phát triển thêm các tính năng du lịch thông minh như chatbot, gợi ý lịch trình, phân tích hình ảnh động, v.v.

2. Sơ đồ hệ thống



3. Triển khai

🔧 Xây dựng mô hình huấn luyện:

- Import thư viện

```
from tensorflow.keras.applications import MobileNetV2
```

```
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
import json
```

- Các thư viện TensorFlow Keras phục vụ cho việc tải model, xây dựng layers, tăng cường dữ liệu, tối ưu hóa và callbacks.
- json để lưu mapping giữa tên lớp và index.

- Cấu hình thông số

```
IMAGE_SIZE = (224, 224) # Kích thước ảnh đầu vào cho model
```

```
BATCH_SIZE = 32 # Số ảnh xử lý trong 1 lần (batch)
```

```
EPOCHS = 20 # Số epoch huấn luyện (lặp qua toàn bộ dữ liệu)
```

```
NUM_CLASSES = 3 # Số lớp phân loại (sẽ cập nhật sau)
```

- 224x224 là kích thước tiêu chuẩn đầu vào của MobileNetV2.

- BATCH_SIZE và EPOCHS điều chỉnh quá trình huấn luyện.
- Tải base model MobileNetV2
`base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(*IMAGE_SIZE, 3))`
 - Tải model MobileNetV2 đã huấn luyện trên ImageNet, bỏ phần classifier (phần top), chỉ giữ phần trích xuất đặc trưng.
 - Input là ảnh RGB kích thước 224x224.
- Đóng băng tất cả lớp của base_model
`for layer in base_model.layers:`
`layer.trainable = False`
 Không cập nhật trọng số các lớp base_model trong bước đầu huấn luyện (giúp tránh overfitting, giảm thời gian).
- Thêm các lớp phân loại phía trên base_model
`x = base_model.output`
`x = GlobalAveragePooling2D()(x)`
`x = Dense(64, activation='relu')(x)`
 - GlobalAveragePooling2D: chuyển tensor đặc trưng cuối cùng thành vector 1 chiều.
 - Dense(64): lớp fully connected với 64 neuron, activation ReLU để tăng khả năng trích xuất đặc trưng mới.
- Tạo bộ tăng cường dữ liệu (Data Augmentation) cho training
`train_datagen = ImageDataGenerator(`
`rescale=1./255, # Chuẩn hóa ảnh về [0,1]`
`rotation_range=30, # Xoay ảnh ngẫu nhiên 30 độ`
`width_shift_range=0.2, # Dịch ngang 20%`
`height_shift_range=0.2, # Dịch dọc 20%`
`shear_range=0.2, # Biến đổi hình học shear`
`zoom_range=0.2, # Phóng to thu nhỏ 20%`
`horizontal_flip=True, # Lật ngang`
`fill_mode='nearest' # Cách điền pixel khi biến đổi`
`)`
 Mục đích: làm đa dạng hóa dữ liệu, tránh overfitting, giúp model tổng quát tốt hơn.
- Load dữ liệu training từ thư mục
`train_generator = train_datagen.flow_from_directory(`
`'dataset/train', # Thư mục dữ liệu training`
`target_size=IMAGE_SIZE,`
`batch_size=BATCH_SIZE,`
`class_mode='categorical' # Phân loại đa lớp (one-hot encoding)`

-)
- Đọc dữ liệu ảnh từ thư mục có cấu trúc phân loại theo folder con.
- Lưu mapping class_name → index ra file JSON
`with open("ethnic_class_indices.json", "w", encoding="utf-8") as f:`
`json.dump(train_generator.class_indices, f, ensure_ascii=False, indent=2)`
 Lưu thông tin lớp để sử dụng khi dự đoán (mapping tên lớp ↔ số nguyên).
 - Cập nhật số lớp chính xác
`NUM_CLASSES = train_generator.num_classes`
 Cập nhật số lớp từ dữ liệu thực tế (đọc được từ thư mục).
 - Thêm lớp output softmax cuối cùng
`predictions = Dense(NUM_CLASSES, activation='softmax')(x)`
 Lớp phân loại với số output = số lớp, activation softmax trả về xác suất mỗi lớp.
 - Tạo mô hình hoàn chỉnh
`model = Model(inputs=base_model.input, outputs=predictions)`
 Kết hợp base_model với lớp phân loại mới tạo thành mô hình đầy đủ.
 - Compile mô hình với optimizer, hàm mất mát và metric
`model.compile(optimizer=Adam(learning_rate=1e-4),`
`loss='categorical_crossentropy',`
`metrics=['accuracy'])`
 - Dùng Adam optimizer với learning rate nhỏ (do base model đóng băng).
 - Mất mát categorical crossentropy cho bài toán đa lớp.
 - Tạo bộ dữ liệu validation (không tăng cường, chỉ chuẩn hóa)
`val_datagen = ImageDataGenerator(rescale=1./255)`
`val_generator = val_datagen.flow_from_directory(`
`'dataset/val',`
`target_size=IMAGE_SIZE,`
`batch_size=BATCH_SIZE,`
`class_mode='categorical'`
`)`
 - Huấn luyện bước 1: chỉ train lớp mới, base model đóng băng
`model.fit(`
`train_generator,`
`epochs=5,`
`validation_data=val_generator`
`)`
 - Mở khóa 20 lớp cuối của base model để fine-tune
`for layer in base_model.layers[-20:]:`
`layer.trainable = True`

Cho phép tinh chỉnh một phần model gốc giúp tăng khả năng thích ứng với dữ liệu mới.

- Compile lại model với learning rate nhỏ hơn cho fine-tuning
`model.compile(optimizer=Adam(learning_rate=1e-5),
loss='categorical_crossentropy',
metrics=['accuracy'])`
- Thiết lập callback hỗ trợ huấn luyện
`early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3)`
 - EarlyStopping: dừng huấn luyện khi validation loss không cải thiện 5 epoch.
 - ReduceLROnPlateau: giảm learning rate khi val_loss không giảm sau 3 epoch.
- Huấn luyện bước 2: fine-tuning toàn bộ model mở khóa
`model.fit(
train_generator,
epochs=EPOCHS,
validation_data=val_generator,
callbacks=[early_stop, reduce_lr]
)`
- Lưu model huấn luyện hoàn chỉnh
`model.save("image_classifier_finetuned.h5")`

Huấn luyện thành công

```
Found 59 images belonging to 3 classes.
Found 16 images belonging to 3 classes.
Epoch 1/5
2/2 ----- 0s 616ms/step - accuracy: 0.2945 - loss: 1.4714
c:\Users\lenovo\Downloads\AI\duan_2_chatbot\venv\lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning:
self._warn_if_super_not_called()
2/2 ----- 9s 3s/step - accuracy: 0.3093 - loss: 1.4541 - val_accuracy: 0.3750 - val_loss: 1.3256
Epoch 2/5
2/2 ----- 2s 1s/step - accuracy: 0.3484 - loss: 1.2865 - val_accuracy: 0.3125 - val_loss: 1.2395
Epoch 3/5
2/2 ----- 2s 1s/step - accuracy: 0.4070 - loss: 1.1514 - val_accuracy: 0.3750 - val_loss: 1.1817
Epoch 4/5
2/2 ----- 2s 1s/step - accuracy: 0.3597 - loss: 1.2060 - val_accuracy: 0.3125 - val_loss: 1.1362
Epoch 5/5
2/2 ----- 2s 924ms/step - accuracy: 0.3510 - loss: 1.1449 - val_accuracy: 0.3750 - val_loss: 1.0968
Epoch 1/20
2/2 ----- 11s 3s/step - accuracy: 0.3284 - loss: 1.1946 - val_accuracy: 0.3750 - val_loss: 1.0774 - learning_rate: 1.0000e-05
Epoch 2/20
2/2 ----- 2s 1s/step - accuracy: 0.4283 - loss: 1.0751 - val_accuracy: 0.3750 - val_loss: 1.0589 - learning_rate: 1.0000e-05
Epoch 3/20
2/2 ----- 2s 1s/step - accuracy: 0.5365 - loss: 1.0018 - val_accuracy: 0.4375 - val_loss: 1.0404 - learning_rate: 1.0000e-05
Epoch 4/20
2/2 ----- 2s 1s/step - accuracy: 0.3084 - loss: 1.0706 - val_accuracy: 0.5000 - val_loss: 1.0226 - learning_rate: 1.0000e-05
Epoch 5/20
2/2 ----- 2s 1s/step - accuracy: 0.5161 - loss: 1.0078 - val_accuracy: 0.5000 - val_loss: 1.0051 - learning_rate: 1.0000e-05
Epoch 6/20
2/2 ----- 2s 1s/step - accuracy: 0.4518 - loss: 1.0795 - val_accuracy: 0.5000 - val_loss: 0.9889 - learning_rate: 1.0000e-05
Epoch 7/20
2/2 ----- 2s 1s/step - accuracy: 0.4057 - loss: 1.0192 - val_accuracy: 0.5000 - val_loss: 0.9734 - learning_rate: 1.0000e-05
Epoch 8/20
2/2 ----- 2s 1s/step - accuracy: 0.5786 - loss: 0.9709 - val_accuracy: 0.5000 - val_loss: 0.9585 - learning_rate: 1.0000e-05
...
```

Xây dựng API FastAPI:

- Import thư viện
`import nest_asyncio`

```

import uvicorn
import threading
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np
import io
import json

```

- nest_asyncio: vá lỗi cho event loop của Python trong môi trường Jupyter, giúp chạy uvicorn không bị block.
- uvicorn: server ASGI chạy app FastAPI.
- threading: chạy server trong một luồng riêng biệt.
- fastapi: framework để xây dựng API.
- tensorflow.keras.models.load_model: tải mô hình deep learning đã lưu.
- PIL.Image: xử lý ảnh.
- numpy: xử lý mảng dữ liệu.
- io: làm việc với luồng byte.
- json: đọc file JSON.

▪ Vá lỗi Jupyter bằng nest_asyncio.apply()

```
nest_asyncio.apply()
```

Jupyter sử dụng event loop riêng, nest_asyncio.apply() giúp uvicorn chạy bình thường bên trong notebook mà không gây lỗi.

▪ Tải mô hình và file mapping nhãn

```

model = load_model("image_classifier_finetuned.h5")
with open("ethnic_class_indices.json", "r", encoding="utf-8") as f:
    class_indices = json.load(f)
    index_to_class = {v: k for k, v in class_indices.items()}

```

- Mô hình Keras được load từ file .h5.
- File JSON chứa mapping class_name -> index, được đảo lại thành index -> class_name để dễ lấy nhãn khi dự đoán.

▪ Khởi tạo FastAPI và cấu hình CORS

```

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Cho phép mọi domain frontend gọi API
    allow_methods=["*"],
    allow_headers=["*"],
)

```

- Tạo app FastAPI.
- Thêm middleware CORS để cho phép frontend từ bất kỳ domain nào gọi API (phòng trường hợp bạn gọi API từ localhost, Jupyter, hoặc domain khác).

▪ Hàm tiền xử lý ảnh

```
def preprocess_image(image_bytes: bytes) -> np.ndarray:
    try:
        img = Image.open(io.BytesIO(image_bytes)).convert("RGB")
    except Exception:
        raise HTTPException(status_code=400, detail="File không phải ảnh hợp lệ")
    img = img.resize((224, 224))
    img_array = np.array(img) / 255.0
    return np.expand_dims(img_array, axis=0)
```

- Nhận ảnh dưới dạng bytes.
- Mở ảnh, chuyển sang RGB.
- Resize về 224x224 (định dạng đầu vào của model).
- Chuẩn hóa ảnh về khoảng [0,1].
- Thêm chiều batch (dạng [1, 224, 224, 3]) để model nhận.

Nếu file không phải ảnh, trả về lỗi HTTP 400.

▪ API route dự đoán ảnh

```
@app.post("/predict/")
async def predict_image(file: UploadFile = File(...)):
    image_bytes = await file.read()
    input_tensor = preprocess_image(image_bytes)
    prediction = model.predict(input_tensor)
    label_index = int(np.argmax(prediction[0]))
    confidence = float(np.max(prediction[0]))
    class_name = index_to_class.get(label_index, "Unknown")
    return {
        "label": label_index,
        "class_name": class_name,
        "confidence": confidence,
    }
```

- Nhận file ảnh upload POST tới /predict/.
- Đọc nội dung file ảnh.
- Tiền xử lý ảnh.
- Dự đoán với model.
- Lấy nhãn có xác suất cao nhất, cùng với độ tin cậy (confidence).

- Trả về JSON gồm label (số nhãn), class_name (tên nhãn), và confidence (độ tin cậy).
- **Hàm chạy uvicorn server**
`def run():`
`uvicorn.run(app, host="0.0.0.0", port=8002)`
 Chạy server trên tất cả IP (0.0.0.0), port 8002.
- **Chạy uvicorn trong luồng riêng để không block notebook**
`thread = threading.Thread(target=run, daemon=True)`
`thread.start()`
 - Tạo một thread chạy server uvicorn.
 - `daemon=True` để luồng sẽ tự động dừng khi notebook tắt.
 - Giúp bạn vẫn dùng notebook bình thường mà backend chạy song song.

Khởi chạy API thành công:

```

🚀 Server đang chạy tại http://localhost:8002
Gửi ảnh POST tới /predict/ để nhận dự đoán.
INFO: Started server process [14292]
INFO: waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8002 (Press CTRL+C to quit)

```

🌈 Frontend (HTML/JS hoặc React):

- Phần HTML & CSS:
`<!DOCTYPE html>`
`<html lang="vi">`
`<head>`
`<meta charset="UTF-8" />`
`<meta name="viewport" content="width=device-width, initial-scale=1" />`
`<title>Nhận diện Địa danh</title>`
`<style>`
`/* CSS tạo bố cục và kiểu dáng cho trang */`
`body { font-family: Arial, sans-serif; background: #f0f2f5; padding: 20px;`
`}`
`.container { max-width: 600px; margin: auto; background: #fff; padding:`
`20px; border-radius: 12px;`
`box-shadow: 0 0 15px rgba(0,0,0,0.2); text-align: center; }`
`#imageInput { margin: 10px auto; display: block; }`
`#preview { display: none; max-width: 100%; margin: 10px auto; border-`
`radius: 10px;`
`box-shadow: 0 0 6px rgba(0,0,0,0.2); }`

```

    #uploadBtn { display: none; background: #007bff; color: white; padding:
10px 24px;
        border: none; border-radius: 5px; cursor: pointer; margin: 10px
auto; display: block; }
    #result { font-size: 18px; font-weight: bold; color: green; margin-top:
15px; }
    #conclusion { font-style: italic; color: #444; margin-top: 5px; }
</style>
</head>
<body>
    <div class="container">
        <h2 style="color:#004080;">Nhận diện Địa danh</h2>
        <input type="file" id="imageInput" accept="image/*" />
        <img id="preview" src="" alt="Ảnh xem trước" />
        <button id="uploadBtn">Gửi ảnh</button>
        <p id="result"></p>
        <p id="conclusion"></p>
    </div>

```

- Tạo một vùng chứa trung tâm (container) để chứa các thành phần giao diện:
 - Tiêu đề
 - Input chọn file ảnh
 - Ảnh xem trước (ẩn lúc đầu)
 - Nút gửi ảnh (ẩn lúc đầu)
 - 2 đoạn văn để hiển thị kết quả dự đoán.

■ Phần JavaScript

```

document.addEventListener('DOMContentLoaded', () => {
    const imgInput = document.getElementById('imageInput');
    const imgPreview = document.getElementById('preview');
    const uploadBtn = document.getElementById('uploadBtn');
    const result = document.getElementById('result');
    const conclusion = document.getElementById('conclusion');

```

// Khi chọn file ảnh

```

imgInput.addEventListener('change', () => {
    const file = imgInput.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = e => {
            imgPreview.src = e.target.result;    // Hiển thị ảnh xem trước

```

```

    imgPreview.style.display = 'block';    // Hiện ảnh
    uploadBtn.style.display = 'block';    // Hiện nút gửi
    result.textContent = "";              // Xóa kết quả cũ
    conclusion.textContent = "";
  };
  reader.readAsDataURL(file);            // Đọc file thành URL ảnh
} else {
  // Nếu bỏ chọn ảnh thì ẩn ảnh xem trước và nút gửi
  imgPreview.style.display = 'none';
  uploadBtn.style.display = 'none';
  result.textContent = "";
  conclusion.textContent = "";
}
});

// Khi nhấn nút gửi ảnh
uploadBtn.onclick = async () => {
  const file = imgInput.files[0];
  if (!file) {
    alert('Vui lòng chọn ảnh!');
    return;
  }
  result.textContent = 'Đang xử lý...';
  conclusion.textContent = "";

  // Tạo đối tượng FormData để gửi file
  const formData = new FormData();
  formData.append('file', file);

  try {
    // Gửi POST ảnh lên backend ở địa chỉ http://localhost:8002/predict/
    const response = await fetch('http://localhost:8002/predict/', {
      method: 'POST',
      body: formData
    });

    if (!response.ok) {
      const errText = await response.text();
      console.error('Server lỗi:', errText);
    }
  }

```

```

        throw new Error('Server trả về lỗi');
    }

    // Đọc dữ liệu JSON trả về
    const data = await response.json();

    // Hiển thị độ chính xác và tên lớp dự đoán
    result.textContent = `Độ chính xác: ${((data.confidence * 100).toFixed(2))}%`;
    conclusion.textContent = `Ảnh này có khả năng cao là ${data.class_name.toUpperCase()}.`;
    } catch (err) {
        // Nếu lỗi kết nối hoặc lỗi server
        result.textContent = 'Lỗi kết nối tới server.';
        console.error(err);
    }
};
});

```

- Khi trang tải xong, JS lấy các phần tử trong DOM để thao tác.
- Khi người dùng chọn ảnh:
 - Đọc ảnh và hiển thị xem trước.
 - Hiện nút gửi ảnh.
 - Xóa các thông tin kết quả cũ.
- Khi nhấn nút gửi:
 - Kiểm tra có ảnh được chọn hay không, nếu không báo lỗi.
 - Gửi ảnh dưới dạng POST request lên API backend.
 - Đợi phản hồi JSON, hiển thị kết quả dự đoán (độ chính xác và tên lớp).
 - Xử lý lỗi nếu không kết nối được hoặc server trả lỗi.
- Phần Python ghi file ra đĩa


```
with open("upload_image.html", "w", encoding="utf-8") as f:
    f.write(html_code)
print("Đã tạo file upload_image.html thành công.")
```

 - Ghi chuỗi HTML + JS trong biến html_code ra file tên upload_image.html trong thư mục hiện hành.
 - Mở file này bằng trình duyệt để sử dụng giao diện upload ảnh.

Mô phỏng:

- Bước 1: Chuẩn bị ảnh để mô phỏng
Dưới đây là 1 bức hình về **Hội An**



- Bước 2: Khởi chạy API

Trước khi chạy thử ứng dụng thì cần phải khởi chạy API trước, khi thấy xuất hiện như hình dưới là đã khởi chạy thành công

```
🚀 Server đang chạy tại http://localhost:8002  
Gửi ảnh POST tới /predict/ để nhận dự đoán.  
INFO: Started server process [14292]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
INFO: Uvicorn running on http://0.0.0.0:8002 (Press CTRL+C to quit)
```

- Bước 3: Upload file ảnh lên giao diện để kiểm thử

Giao diện ban đầu:

Nhận diện Địa danh

Choose File No file chosen

Giao diện khi Upload file ảnh lên:

Nhận diện Địa danh

Choose File hoi_an_1.jfif



Gửi ảnh

- Bước 4: Nhận kết quả dự đoán
Khi nhấn nút gửi thì sẽ hiển thị kết quả dự đoán như sau:

Nhận diện Địa danh

Choose File hoi_an_1.jfif



Gửi ảnh

Độ chính xác: 46.49%

Ảnh này có khả năng cao là HOIAN.

3.3.2. Hệ thống chatbot hỏi đáp trong du lịch

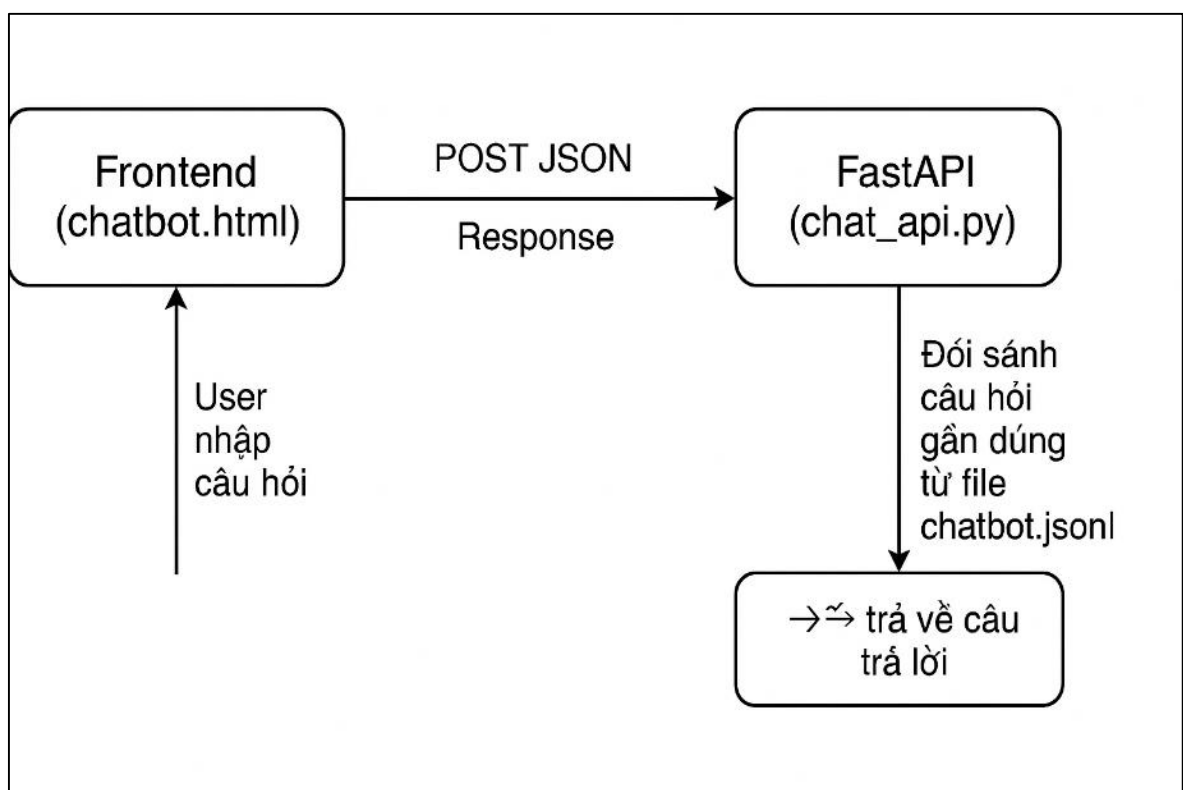
1. Mục đích

Mục tiêu của hệ thống là xây dựng một chatbot đơn giản có khả năng trả lời các câu hỏi phổ biến trong lĩnh vực du lịch. Người dùng có thể đặt câu hỏi (ví dụ: “Đà Lạt mùa nào đẹp?”, “Thời điểm nên đi Sapa là khi nào?”...), chatbot sẽ phản hồi bằng nội dung đã được huấn luyện trước.

Việc xây dựng chatbot này giúp:

- Tự động hóa việc trả lời câu hỏi cho du khách.
- Nâng cao trải nghiệm người dùng trên nền tảng du lịch trực tuyến.
- Tạo nền tảng để mở rộng thành chatbot thông minh tích hợp AI như GPT hoặc RAG.

2. Sơ đồ



3. Triển khai

🔧 Bước 1. Tạo dữ liệu huấn luyện

- Giao diện Jupyter để nhập câu hỏi – câu trả lời và lưu vào file chatbot.jsonl.
- Dữ liệu được lưu theo từng dòng JSON:

```
{"prompt": "Sapa có tuyết không?", "completion": "Sapa thỉnh thoảng có tuyết vào mùa"
```

✓ Code & Giải thích

1. Import các thư viện

```
import json  
import ipywidgets as widgets
```

from IPython.display import display, Markdown

- json: để ghi dữ liệu vào file dưới dạng JSON.
- ipywidgets: tạo các ô nhập liệu tương tác trong Jupyter Notebook.
- display, Markdown: để hiển thị giao diện người dùng.

2. Tạo 2 ô nhập liệu cho người dùng

question_input = widgets.Textarea(...)

answer_input = widgets.Textarea(...)

- question_input: nơi nhập câu hỏi người dùng muốn chatbot nhận diện.
- answer_input: nơi nhập câu trả lời tương ứng.
- Cả 2 đều là Textarea với độ cao 80px và độ rộng 100%.

3. Tạo nút Lưu

save_button = widgets.Button(...)

- Nút này sẽ gọi hàm lưu dữ liệu khi được nhấn.
- button_style='success': tô màu xanh cho nút (hiệu ứng đẹp hơn).

4. Tạo khu vực hiển thị kết quả

output = widgets.Output()

- Dùng để hiển thị thông báo như “Đã lưu” hoặc “Lỗi...”.

5. Hàm xử lý khi nhấn nút Lưu

def save_to_jsonl(b):

...

- Lấy giá trị từ 2 ô question_input và answer_input.
- Nếu một trong hai ô trống → cảnh báo và không lưu.
- Nếu hợp lệ → tạo một dictionary:
data = {"prompt": ..., "completion": ...}
- Ghi vào file chatbot.jsonl dưới dạng 1 dòng JSON (dạng JSON Lines).
- Sau khi ghi thành công:
 - Hiển thị thông báo "✓ Đã lưu..."
 - Xóa nội dung hai ô nhập.

6. Gắn sự kiện click vào nút

save_button.on_click(save_to_jsonl)

- Khi nhấn nút, sẽ gọi hàm save_to_jsonl.

7. Hiển thị toàn bộ giao diện trên Notebook

display(Markdown(...))

display(question_input, answer_input, save_button, output)

- Hiển thị tiêu đề bằng Markdown.
- Hiển thị lần lượt: ô nhập câu hỏi, ô nhập trả lời, nút lưu và vùng kết quả.

Bước 2: Xây dựng API với FastAPI

- Tạo file chat_api.py gồm:
 - Đọc dữ liệu từ chatbot.jsonl
 - Tìm câu hỏi gần giống nhất bằng difflib.get_close_matches
 - Trả về câu trả lời tương ứng hoặc phản hồi mặc định nếu không tìm thấy

✓ Code & Giải thích

1. Nội dung file API – Chatbot dùng FastAPI

```
backend_code = '''
```

```
...
'''
```

a. Import các thư viện cần thiết

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import json
import difflib
```

- FastAPI: framework xây dựng REST API rất nhanh.
- CORSMiddleware: cho phép các domain khác truy cập API (frontend ↔ backend).
- BaseModel: dùng để định nghĩa schema đầu vào (đối tượng JSON).
- json: đọc file chatbot.jsonl.
- difflib: dùng get_close_matches để tìm câu hỏi gần giống nhất.

b. Khởi tạo ứng dụng FastAPI và bật CORS

```
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Cho phép gọi từ mọi nguồn (localhost, web khác)
    allow_methods=["*"],
    allow_headers=["*"],
)
```

c. Đọc dữ liệu từ file chatbot.jsonl

```
with open("chatbot.jsonl", "r", encoding="utf-8") as f:
```

```
    chat_data = [json.loads(line) for line in f]
```

- Mỗi dòng trong file là một JSON có dạng:


```
{ "prompt": "sapa có tuyết không", "completion": "Sapa có tuyết vào mùa đông." }
```
- Tạo ra danh sách chat_data chứa tất cả các cặp câu hỏi – trả lời.

d. Định nghĩa schema cho yêu cầu POST

```
class Message(BaseModel):
    prompt: str
```


- Dùng khi người dùng gửi dữ liệu kiểu:
`{ "prompt": "đà lạt có gì chơi?" }`
- e. Định nghĩa endpoint chính của chatbot
- ```
@app.post("/chat")
def chat_with_bot(message: Message):
 ...
 Nhận prompt từ người dùng và chuẩn hóa:
 user_input = message.prompt.strip().lower()
 Lấy danh sách tất cả các prompt đã lưu:
 prompts = [item["prompt"].lower() for item in chat_data]
 Tìm câu hỏi gần đúng nhất với input người dùng:
 matches = difflib.get_close_matches(user_input, prompts, n=1,
 cutoff=0.6)
 o n=1: chỉ lấy 1 câu gần giống nhất.
 o cutoff=0.6: ngưỡng giống nhau tối thiểu (từ 0–1).
 Nếu tìm thấy câu gần giống → trả lời tương ứng:
 if matches:
 for item in chat_data:
 if item["prompt"].lower() == matches[0]:
 return {"completion": item["completion"]}
 Nếu không tìm thấy → trả lời mặc định:
 return {"completion": "Xin lỗi. Tôi không hiểu câu hỏi của bạn."}
```
2. Ghi chuỗi backend\_code thành file thật
- ```
with open("chat_api.py", "w", encoding="utf-8") as f:
    f.write(backend_code)
```
- Ghi nội dung chuỗi backend_code vào file chat_api.py trên máy bạn.
3. In ra thông báo sau khi ghi xong
- ```
print("Đã tạo file chat_api.py")
```

### **Bước 3: Giao diện frontend (chatbot.html)**

- HTML đơn giản với ô nhập tin nhắn và khung hiển thị hội thoại
- Gửi câu hỏi đến API qua fetch("/chat", POST)
- Nhận kết quả phản hồi và hiển thị ra màn hình theo kiểu chat

#### ✓ **Code & Giải thích**

1. Cấu trúc tổng thể

```
<div id="container">
 <h3>  Chatbot </h3>
 <div id="chat"></div> <!-- Nơi hiển thị hội thoại -->
```

```
<input id="msg" ... /> <!-- Ô nhập câu hỏi -->
<button onclick="send()">Gửi</button> <!-- Nút gửi -->
</div>
```

## 2. Phần style (CSS)

```
body, #container, #chat, .message, .user, .bot, #msg, button { ... }
```

- #container: giao diện hộp trắng bo góc.
- #chat: vùng hiển thị tin nhắn (cuộn được).
- .user: tin nhắn người dùng (bên phải, nền xanh).
- .bot: tin nhắn chatbot (bên trái, nền xám).
- #msg: ô nhập tin nhắn.
- button: nút "Gửi" có màu xanh lá.

## 3. Hàm send() – xử lý khi bấm nút Gửi

```
async function send() {
 let msg = document.getElementById("msg").value.trim();
 if (!msg) return;
```

- Lấy nội dung tin nhắn người dùng.
- Nếu rỗng thì không gửi gì cả.

## 4. Hiển thị tin nhắn người dùng

```
chat.innerHTML += "<div class='message user'>Bạn: " + msg +
"</div>";
```

- Gắn nội dung người dùng vào vùng chat, căn phải, màu xanh.

## 5. Gửi tin nhắn tới API /chat

```
const res = await fetch("http://localhost:8000/chat", {
 method: "POST",
 headers: { "Content-Type": "application/json" },
 body: JSON.stringify({ prompt: msg })
});
```

- Gửi HTTP POST tới FastAPI backend tại localhost:8000/chat kèm nội dung JSON: { "prompt": "..." }.

## 6. Hiển thị câu trả lời của bot

```
const data = await res.json();
chat.innerHTML += "<div class='message bot'>Bot: " +
data.completion + "</div>";
```

- Hiển thị phần "completion" mà backend trả về như phản hồi từ chatbot.

## 7. Tự động cuộn xuống cuối khung chat

```
chat.scrollTop = chat.scrollHeight;
```

- Giúp người dùng luôn nhìn thấy tin nhắn mới nhất.

## Bước 4: Khởi chạy server

## ✓ Code & Giải thích

1. `import nest_asyncio`

`import nest_asyncio`

- Thư viện này vá lỗi cho các môi trường như Jupyter Notebook, vốn không cho phép chạy nhiều vòng lặp async cùng lúc.

2. Áp dụng vá lỗi vòng lặp async

`nest_asyncio.apply()`

- Cho phép uvicorn khởi động bên trong Notebook mà không gặp lỗi như: `RuntimeError: This event loop is already running`

3. `import uvicorn` và tạo hàm `run()`

`import uvicorn`

`def run():`

`uvicorn.run("chat_api:app", host="0.0.0.0", port=8000)`

- uvicorn là ASGI server dùng để chạy ứng dụng FastAPI.
- "chat\_api:app": chỉ đến file chat\_api.py và đối tượng app trong đó.
- host="0.0.0.0": cho phép truy cập từ mọi địa chỉ IP (cần khi dùng Colab hoặc máy ảo).
- port=8000: chạy ở cổng 8000 (mặc định của FastAPI).

4. Chạy server trong luồng song song

`import threading`

`thread = threading.Thread(target=run)`

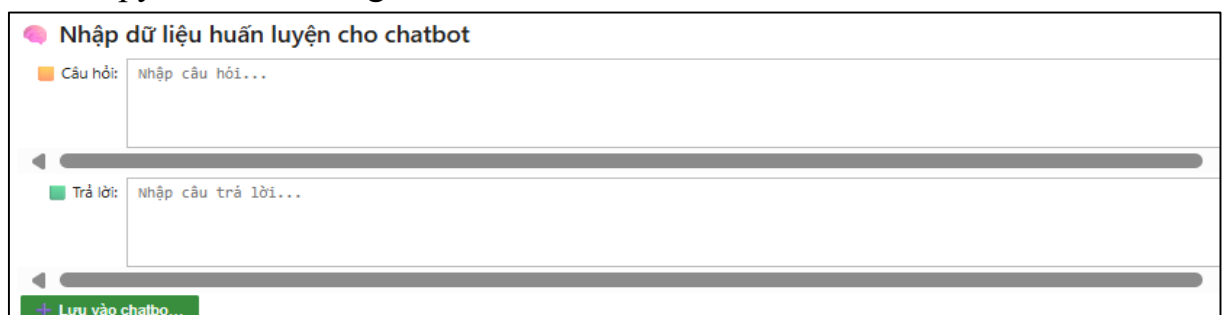
`thread.start()`

- Tạo một luồng mới (thread) để chạy server.
- Giúp không chặn kernel chính của Jupyter – bạn vẫn có thể nhập thêm lệnh khác.

## 🚦 Mô phỏng

✓ Bước 1: Tạo dữ liệu huấn luyện

Trên Jupyter sẽ hiển thị giao diện sau:



The screenshot shows a Jupyter Notebook cell with a title "Nhập dữ liệu huấn luyện cho chatbot". Below the title, there are two input fields: "Câu hỏi: Nhập câu hỏi..." and "Trả lời: Nhập câu trả lời...". At the bottom, there is a green button labeled "+ Lưu vào chatbo...".

Nhập câu hỏi và câu trả lời

**Nhập dữ liệu huấn luyện cho chatbot**

**Câu hỏi:** Sapa có đẹp không?

**Trả lời:** Sapa rất đẹp đặc biệt là vào mùa xuân và mùa thu, bạn muốn biết thêm gì không?

[Lưu vào chatbot...](#)

Sau khi nhập xong dữ liệu sẽ tự động nạp vào file *chatbot.jsonl* dưới dạng sau:

```
{'prompt': 'Sapa có đẹp không?', 'completion': 'Sapa rất đẹp đặc biệt là vào mùa xuân và mùa thu, bạn muốn biết thêm gì không?'}
```

### ✓ Bước 2: Khởi chạy Server

Sau khi nhìn thấy như hình dưới đây thì đồng nghĩa với việc đã khởi chạy thành công.

```
INFO: Started server process [17440]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

### ✓ Bước 3: Nhập câu hỏi cho chatbot

**Chatbot**

Sapa ở đâu?

[Gửi](#)

### ✓ Bước 4: Nhận phản hồi từ chatbot

Nếu câu hỏi liên quan đến lĩnh vực huấn đã huấn luyện thì chatbot sẽ phản hồi theo những gì nó được học

Chatbot

Bạn: Sapa ở đâu?

**Bot:** Sapa là một thị trấn thuộc tỉnh Lào Cai, nằm ở phía Tây Bắc Việt Nam, gần biên giới Trung Quốc. Nơi đây nổi tiếng với cảnh quan núi non hùng vĩ và khí hậu mát mẻ quanh năm.

Nhập tin nhắn...

Gửi

Nếu câu hỏi không liên quan đến lĩnh vực chatbox được huấn luyện thì nó sẽ phản hồi về câu trả lời mặc định.

Chatbot

Bạn: Bóng đá là gì?

**Bot:** Xin lỗi. Tôi không hiểu câu hỏi của bạn.

Nhập tin nhắn...

Gửi

## KẾT LUẬN

Qua quá trình thực hiện báo cáo thực tập cơ sở với chủ đề “Trí tuệ nhân tạo và các ứng dụng học sâu”, em đã có cơ hội tiếp cận và tìm hiểu sâu hơn về một lĩnh vực công nghệ tiên tiến, có tầm ảnh hưởng rộng lớn trong thời đại số hóa hiện nay. Từ những khái niệm nền tảng của trí tuệ nhân tạo, các lĩnh vực liên quan như học máy, xử lý ngôn ngữ tự nhiên, thị giác máy tính đến những kỹ thuật học sâu hiện đại như mạng nơ-ron, CNN, RNN, Transformer hay GAN – tất cả đều đã được hệ thống hóa và trình bày một cách chi tiết trong báo cáo này.

Điểm nổi bật trong quá trình thực tập là việc kết hợp lý thuyết với thực hành. Em đã tự tay xây dựng hai mô hình ứng dụng thực tiễn: một hệ thống dự đoán địa danh từ ảnh sử dụng CNN triển khai qua FastAPI và React, và một chatbot hỏi đáp trong lĩnh vực du lịch dựa trên dữ liệu huấn luyện thủ công. Thông qua các dự án nhỏ này, em không chỉ hiểu rõ hơn về quy trình phát triển mô hình AI mà còn nắm được cách triển khai chúng trong môi trường ứng dụng thực tế thông qua việc xây dựng giao diện, tạo API và xử lý dữ liệu đầu vào/đầu ra.

Bên cạnh kiến thức chuyên môn, báo cáo cũng giúp em rèn luyện các kỹ năng như làm việc độc lập, tư duy logic, tìm kiếm và xử lý thông tin, khả năng lập trình cũng như trình bày nội dung kỹ thuật một cách rõ ràng, có hệ thống.

Tuy vẫn còn nhiều hạn chế về phạm vi nghiên cứu, độ phức tạp mô hình và trải nghiệm thực tế, nhưng đây là một bước khởi đầu quan trọng giúp em định hướng rõ hơn về sở thích và con đường chuyên môn trong tương lai – đặc biệt là trong lĩnh vực trí tuệ nhân tạo và học sâu. Em hy vọng với nền tảng kiến thức đã tích lũy được, em có thể tiếp tục học tập và nghiên cứu sâu hơn, từ đó đóng góp vào các ứng dụng AI phục vụ xã hội một cách thiết thực và hiệu quả.

## TÀI LIỆU THAM KHẢO

*Deep Learning – Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016*

*Deep Learning with Python – François Chollet, 2018 (1st edition), 2021 (2nd edition)*

*Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow – Aurélien Géron, 2022 (3rd edition)*

*Attention is All You Need – Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al., 2017*

*Python Machine Learning – Sebastian Raschka, Vahid Mirjalili, 2020 (3rd edition)*



## TƯỜNG TRÌNH CHỈNH SỬA BÁO CÁO

Sau khi hoàn thành bản báo cáo ban đầu, em đã nhận được nhiều nhận xét, góp ý từ các bạn trong nhóm. Qua đó, em đã tiến hành chỉnh sửa, bổ sung nhằm nâng cao chất lượng báo cáo, khắc phục các điểm chưa hoàn thiện. Tường trình này sẽ trình bày chi tiết quá trình chỉnh sửa dựa trên các nhận xét và phản hồi mà em đã nhận.

### 1. Các điểm tồn tại ban đầu và nhận xét từ các bạn

Trong quá trình đánh giá báo cáo ban đầu, các bạn đã giúp em chỉ ra một số điểm hạn chế như sau:

- Thiếu phần Mở đầu: Báo cáo chưa có phần giới thiệu tổng quan về nội dung, mục tiêu và phạm vi nghiên cứu. Điều này khiến người đọc khó nắm bắt được bức tranh tổng thể và định hướng của báo cáo.
- Không có phần Kết luận chung: Báo cáo thiếu phần tổng kết cuối cùng để nhấn mạnh kết quả đạt được, các bài học kinh nghiệm cũng như đề xuất hướng phát triển.
- Thiếu mục Tài liệu tham khảo: Việc thiếu phần trích dẫn các tài liệu, sách, bài báo khoa học khiến báo cáo thiếu tính học thuật và không minh bạch nguồn thông tin tham khảo.
- Định dạng và trình bày không đồng nhất: Một số chỗ trong báo cáo còn chưa tuân thủ quy chuẩn về font chữ, cỡ chữ, khoảng cách dòng và cách trình bày mục lục, tiêu đề.
- Thiếu đánh giá, so sánh các mô hình học sâu: Phần giới thiệu các mô hình ANN, CNN, RNN, Transformer chỉ mô tả về cấu trúc và nguyên lý, thiếu phần so sánh, đánh giá ưu nhược điểm để xác định mô hình phù hợp cho từng bài toán.
- Thiếu sơ đồ kiến trúc tổng quan hệ thống: Báo cáo chưa trình bày sơ đồ kiến trúc backend – frontend – mô hình AI, làm giảm sự trực quan và khó hình dung được luồng xử lý dữ liệu.
- Phần triển khai ứng dụng tập trung nhiều vào mã nguồn, thiếu phân tích: Việc trình bày code chi tiết nhưng chưa có giải thích rõ về cách thức hoạt động, lý do chọn mô hình và công nghệ.
- Mô hình dự đoán địa danh chưa đạt độ chính xác cao: Kết quả thử nghiệm còn nhiều ảnh trả về sai lệch, chưa được tối ưu.
- Thiếu mô phỏng chi tiết quá trình hoạt động chatbot: Chưa minh họa được các bước huấn luyện, lưu trữ dữ liệu và phản hồi câu hỏi của chatbot.
- Chưa làm rõ mục đích và lợi ích ứng dụng của từng phần triển khai: Báo cáo chưa nêu cụ thể ứng dụng phục vụ lĩnh vực nào và lợi ích thực tế mang lại cho người dùng.

## 2. Quá trình chỉnh sửa và hoàn thiện báo cáo

### + Bổ sung phần Mở đầu

- Em đã xây dựng phần Mở đầu để giới thiệu tổng quan về trí tuệ nhân tạo, vai trò của học sâu trong thời đại hiện nay, các lĩnh vực ứng dụng chính và mục tiêu của báo cáo. Phần này giúp người đọc nắm bắt nhanh ý tưởng và nội dung trọng tâm mà báo cáo hướng đến.

### + Thêm phần Kết luận tổng hợp

- Phần Kết luận được viết rõ ràng, hệ thống, tóm tắt những kết quả đã đạt được trong quá trình thực tập, nhấn mạnh các kỹ thuật học sâu được nghiên cứu, ứng dụng và triển khai thành công.
- Em cũng trình bày các bài học kinh nghiệm, những khó khăn trong thực hành và phương án khắc phục.

### + Hoàn thiện mục Tài liệu tham khảo

- Em bổ sung phần trích dẫn các tài liệu khoa học, sách giáo trình, website uy tín đã tham khảo trong quá trình nghiên cứu và triển khai báo cáo.
- Phần này được trình bày theo chuẩn APA để đảm bảo tính học thuật và giúp người đọc dễ dàng tìm kiếm nguồn tài liệu.

### + Đồng nhất định dạng và trình bày văn bản

- Em đã rà soát toàn bộ báo cáo, chỉnh sửa lại font chữ về Times New Roman, cỡ chữ 14, giãn dòng 1.15, căn lề: phải 3.5 cm, trái 1.5 cm, trên và dưới 2 cm.
- Các tiêu đề chương, mục được đánh số đúng quy chuẩn, giữ khoảng cách hợp lý và nhất quán. Các bảng biểu, hình ảnh được căn chỉnh để báo cáo đẹp mắt và dễ theo dõi hơn.

### + Bổ sung bảng so sánh và đánh giá các mô hình học sâu

- Tại Chương 2, em đã thêm bảng tổng hợp so sánh các mô hình Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Generative Adversarial Network (GAN) và Transformer.
- Bảng so sánh tập trung vào: ưu điểm, nhược điểm, ứng dụng thực tế, yêu cầu tài nguyên và hiệu suất cho từng mô hình.
- Điều này giúp người đọc hiểu sâu hơn về từng mô hình, từ đó lựa chọn mô hình phù hợp với bài toán cụ thể.

### + Vẽ sơ đồ kiến trúc tổng quan hệ thống

- Em đã thiết kế sơ đồ kiến trúc minh họa toàn bộ luồng dữ liệu và tương tác trong ứng dụng:
  - + Người dùng (User) thao tác trên giao diện frontend (React/HTML).
  - + Giao diện gọi API backend (FastAPI).
  - + Backend xử lý ảnh đầu vào, chuyển sang mô hình AI dự đoán.

- + Mô hình trả kết quả dự đoán cho backend.
  - + Backend gửi kết quả cho frontend hiển thị lại cho người dùng.
  - Sơ đồ được đặt tại phần triển khai ứng dụng để làm rõ quy trình vận hành, giúp báo cáo trực quan hơn.
  - ✚ Giải thích và phân tích chi tiết đoạn code triển khai
    - Phần trình bày code được bổ sung giải thích từng bước:
      - + Lý do lựa chọn mô hình MobileNetV2 (nhẹ, hiệu quả với dữ liệu ảnh).
      - + Mô tả quá trình tiền xử lý ảnh (resize, normalize).
      - + Cách tạo API bằng FastAPI nhận ảnh và trả kết quả.
      - + Cách kết nối frontend - backend qua fetch API.
    - Mỗi đoạn code đều có chú thích chi tiết, giúp người đọc hiểu rõ hoạt động và cách thức triển khai.
  - ✚ Cải thiện độ chính xác mô hình
    - Em đã mở rộng và tối ưu lại bộ dữ liệu huấn luyện, điều chỉnh tham số huấn luyện, tăng số epoch, cải thiện độ chính xác của mô hình dự đoán địa danh.
    - Kết quả thử nghiệm mới được bổ sung vào báo cáo, thể hiện sự tiến bộ rõ rệt so với bản đầu.
  - ✚ Mô phỏng chi tiết quá trình hoạt động chatbot
    - Em mô phỏng và ghi lại quy trình huấn luyện chatbot, từ nhập dữ liệu huấn luyện bằng câu hỏi - trả lời vào file JSONL, xây dựng API phản hồi câu hỏi, đến giao diện chat tương tác với người dùng.
    - Các bước được minh họa bằng ảnh chụp màn hình và mô tả cụ thể từng bước xử lý.
  - ✚ Nêu rõ mục đích và lợi ích ứng dụng
    - Trong phần mô tả ứng dụng, em đã nêu rõ mục đích:
      - + Ứng dụng dự đoán địa danh giúp người dùng xác định địa điểm từ hình ảnh.
      - + Chatbot hỏi đáp hỗ trợ cung cấp thông tin du lịch nhanh chóng và tiện lợi.
    - Em cũng trình bày lợi ích cụ thể, giúp báo cáo có chiều sâu và giá trị thực tiễn hơn.
- ### 3. Ý nghĩa và kết quả đạt được
- Việc chỉnh sửa và bổ sung đã giúp báo cáo trở nên hoàn thiện hơn về mọi mặt:
    - + Nội dung đầy đủ, mạch lạc, dễ hiểu: Các phần mở đầu, kết luận, tài liệu tham khảo được hoàn thiện tạo nên bố cục chuẩn mực.
    - + Tăng tính học thuật và thuyết phục: Bảng so sánh mô hình và phân tích kỹ thuật giúp tăng giá trị nghiên cứu khoa học.

- + Tăng tính trực quan: Sơ đồ kiến trúc và mô phỏng chatbot giúp người đọc hình dung rõ ràng quy trình vận hành hệ thống.
- + Ứng dụng có kết quả thực tiễn tốt hơn: Mô hình được cải tiến, cho độ chính xác dự đoán cao, chatbot hoạt động hiệu quả.
- + Báo cáo đạt chuẩn về trình bày: Văn bản được chuẩn hóa định dạng, trình bày chuyên nghiệp, dễ theo dõi.

Những cải tiến này đã giúp báo cáo của em hoàn chỉnh và logic hơn, đồng thời cũng từ những đóng góp đó của các bạn em cũng nhận thấy được những thiếu sót của mình trong quá trình học tập và nghiên cứu, để từ đó em cũng có cơ hội để hoàn chỉnh hơn những thiếu sót của mình trong bài báo cáo.