HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN

HỌC PHẦN: THỰC TẬP CƠ SỞ

Giảng viên hướng dẫn : Trần Đình Quế

Sinh viên : Trần Đức Việt

Mã sinh viên : B22DCCN901

Lớp : D22CQCN01

Nhóm Thực Tập : 30

Hà Nội – 2025

MỤC LỤC

CHƯƠNG 1: TRÍ TUỆ NHÂN TẠO VÀ CÁ ỨNG DỤNG

1: Mở đầu

1.1 Giới thiệu về Trí tuệ nhân tạo (AI)

Trí tuệ nhân tạo (Artificial Intelligence - AI) là một lĩnh vực của khoa học máy tính tập trung vào việc xây dựng các hệ thống có khả năng thực hiện những nhiệm vụ mà trước đây đòi hỏi trí tuệ con người. Những hệ thống này có thể bao gồm việc nhận dạng giọng nói, phân tích hình ảnh, hiểu ngôn ngữ tự nhiên, dự đoán xu hướng hoặc đưa ra quyết định dựa trên dữ liệu.

AI không còn là một khái niệm viễn tưởng mà đã trở thành một phần thiết yếu trong cuộc sống hiện đại. Từ các ứng dụng cá nhân như trợ lý ảo, đề xuất nội dung, đến các hệ thống tự động hóa trong công nghiệp, y tế, và tài chính, AI đang đóng vai trò thay đổi cục diện của nhiều ngành nghề.

Trí tuệ nhân tạo (AI) là lĩnh vực nghiên cứu phát triển các hệ thống có thể mô phỏng hành vi trí tuệ của con người. Trong AI có một nhánh quan trọng là **Machine Learning (ML)**, cho phép máy tính học từ dữ liệu mà không cần lập trình cụ thể.

Một nhánh sâu hơn nữa của ML là **Deep Learning (DL)** – sử dụng các mạng nơ-ron nhân tạo với nhiều lớp ẩn (deep neural networks) để xử lý dữ liệu phức tạp như hình ảnh, âm thanh, văn bản

Minh họa mối quan hệ

```
Trí tuệ nhân tạo (AI)
 └── Học máy (Machine Learning)
      └── Học sâu (Deep Learning)
```

1.2 Tại sao AI trở nên quan trọng?

Sự phát triển của AI gắn liền với:

- **Sự gia tăng khối lượng dữ liệu (Big Data)**
- **Sự tiến bộ về phần cứng và điện toán đám mây**
- **Các thuật toán học máy (Machine Learning) ngày càng mạnh mẽ**

Nhờ đó, các hệ thống AI ngày nay không chỉ có thể học hỏi từ dữ liệu mà còn đưa ra dự đoán chính xác và tối ưu hóa quy trình một cách hiệu quả.

1.3 Mục tiêu của chương 1

Chương 1 này nhằm:

- Giới thiệu tổng quan về trí tuệ nhân tạo và học máy
- Trình bày các thuật toán học máy cơ bản và cách áp dụng chúng bằng Python
- Khám phá các ứng dụng thực tế của AI trong đời sống và công nghiệp

2. Tổng quan về machine learning (ml)

2.1 Machine Learning là gì?

Machine Learning (ML) là một tập hợp các thuật toán và kỹ thuật giúp thiết kế các hệ thống có khả năng **học từ dữ liệu** để đưa ra **dự đoán** hoặc **nhận diện mẫu trong dữ liệu**.

ML khác biệt với lập trình truyền thống, nơi lập trình viên phải viết các quy tắc cụ thể. Thay vào đó, ML sử dụng dữ liệu để "học" và từ đó tạo ra **mô hình (model)** – như minh họa trong hình sau:

Truyền thống	Machine Learning
Data + Program → Output	Data + Output → Program (Model)

2.2 Các bài toán chính trong Machine Learning

- **Classification (Phân loại):** Dự đoán xem một đối tượng thuộc về nhóm nào

  Ví dụ: Dự đoán khối u là lành tính hay ác tính

- **Regression (Hồi quy):** Dự đoán giá trị liên tục

  Ví dụ: Dự đoán giá nhà, nhiệt độ trong tuần tới

- **Clustering (Phân cụm):** Nhóm dữ liệu theo các mẫu chưa biết trước

  Ví dụ: Nhóm khách hàng theo hành vi mua sắm

2.3 Các loại thuật toán Machine Learning

- **Supervised Learning (Học có giám sát):**

  Hệ thống học từ **dữ liệu đã gắn nhãn**.

  Ví dụ: Dự đoán giá nhà từ tập dữ liệu đã biết giá.

- **Unsupervised Learning (Học không giám sát):**

  Hệ thống học từ **dữ liệu không gắn nhãn**, cố gắng tìm ra cấu trúc hoặc mẫu.

  Ví dụ: Phân cụm khách hàng khi không biết nhãn phân loại.

3: Các công cụ và thư viện python trong machine learning

3.1 Python – ngôn ngữ lý tưởng cho học máy

Python được xem là ngôn ngữ lý tưởng cho Machine Learning nhờ:

- Cú pháp đơn giản, dễ học
- Cộng đồng mạnh, tài liệu phong phú
- Nhiều thư viện chuyên dụng hỗ trợ học máy, xử lý dữ liệu, trực quan hóa và triển khai mô hình

3.2 Thư viện NumPy – Xử lý mảng số

NumPy giúp xử lý hiệu quả các mảng dữ liệu nhiều chiều (ndarray), là nền tảng cho các thao tác toán học trong ML.

Một số chức năng chính:

- Tạo mảng số, reshape, slicing, indexing
- Tính toán ma trận, dot product
- Thao tác thống kê: mean, std, sum, cumsum...

```python
import numpy as np
a = np.array([[1, 2], [3, 4]])
print(np.sum(a))  # 10
```

3.3 Pandas – Xử lý dữ liệu dạng bảng

Pandas cung cấp hai cấu trúc dữ liệu chính:

- **Series**: cột dữ liệu một chiều
- **DataFrame**: bảng dữ liệu hai chiều

Các thao tác phổ biến:

- Đọc dữ liệu từ file (CSV, Excel, JSON...)
- Lọc, sắp xếp, tính toán mô tả (describe)
- Xử lý thiếu dữ liệu (NaN), loại bỏ/ghép dòng/cột
- Tính toán thống kê nhóm (groupby, crosstab)

```python
import pandas as pd
df = pd.DataFrame({"Name": ["A", "B"], "Age": [25, 30]})
print(df[df["Age"] > 26])  # Lọc người > 26 tuổi
```

3.4 matplotlib và seaborn – Trực quan hóa dữ liệu

- Trực quan hóa giúp hiểu rõ dữ liệu trước khi huấn luyện mô hình.
matplotlib hỗ trợ các biểu đồ như: line, bar, pie, scatter...
- seaborn mở rộng matplotlib, tạo các biểu đồ thống kê như:

- Biểu đồ phân phối (distplot)
- Biểu đồ hộp (boxplot)
- Biểu đồ điểm (swarmplot, lmplot)

```python
import seaborn as sns
sns.boxplot(x="species", y="sepal_length", data=df)
```

3.5 scikit-learn – Thư viện Machine Learning chính

scikit-learn là thư viện học máy phổ biến nhất trong Python, hỗ trợ:

- Các thuật toán phân loại, hồi quy, phân cụm
- Tiền xử lý dữ liệu: chuẩn hóa, loại bỏ NaN
- Huấn luyện, đánh giá, lưu mô hình

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit([[1], [2], [3]], [2, 4, 6])
print(model.predict([[4]]))  # → [8.]
```

3.6 TensorFlow

TensorFlow là thư viện mạnh mẽ cho học sâu, hỗ trợ xây dựng và huấn luyện mạng nơ-ron. Ví dụ:

```
import tensorflow as tf
model = tf.keras.Sequential([
tf.keras.layers.Dense(10, activation='relu', input_shape=(4,)),
        tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy')
```

3.7 PyTorch

PyTorch được ưa chuộng trong nghiên cứu AI nhờ tính linh hoạt. Ví dụ:

```
import torch
import torch.nn as nn
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc = nn.Linear(4, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        return self.sigmoid(self.fc(x))
model = Net()
```

4: Các thuật toán học có giám sát (supervised learning)

4.1 Tổng quan

Học có giám sát là kỹ thuật mà trong đó mô hình được huấn luyện bằng **dữ liệu đầu vào đã có nhãn (label)**. Từ đó, mô hình học được mối quan hệ giữa đặc trưng (features) và nhãn để dự đoán đầu ra cho dữ liệu mới.

Có hai nhóm chính:

- **Hồi quy (Regression):** dự đoán giá trị liên tục
- **Phân loại (Classification):** dự đoán nhãn rời rạc (A/B/C...)

4.2 Hồi quy tuyến tính (Linear Regression)

Linear Regression là mô hình dự đoán một giá trị liên tục dựa trên một hoặc nhiều biến đầu vào tuyến tính.

Ứng dụng: Dự đoán giá nhà, doanh thu, tuổi thọ sản phẩm...

```
from sklearn.linear_model import LinearRegression
LinearRegression().fit([[1], [2]], [3, 5]).predict([[3]])  # → [7.]
```

4.3 Phân loại bằng hồi quy logistic (Logistic Regression)

Logistic Regression dùng để phân loại dữ liệu nhị phân (như "có/không", "tốt/xấu") bằng cách tính xác suất và áp dụng hàm sigmoid.

Ứng dụng: Dự đoán bệnh tật, spam email, v.v.

```
from sklearn.linear_model import LogisticRegression
LogisticRegression().fit([[0], [1]], [0, 1]).predict([[0.5]])  # → [1]
```

4.4 Máy vector hỗ trợ (SVM – Support Vector Machine)

SVM tìm siêu phẳng (hyperplane) tốt nhất phân chia hai lớp dữ liệu có khoảng cách lớn nhất.

Ứng dụng: Nhận diện chữ viết tay, phân loại hình ảnh, sinh trắc học

```
from sklearn.svm import SVC
SVC().fit([[0], [1]], [0, 1]).predict([[0.8]])  # → [1]
```

4.5 K-Nearest Neighbors (KNN)

KNN phân loại dựa trên số lượng láng giềng gần nhất – quan sát mới được gán nhãn theo đa số của các điểm gần nhất trong không gian.

Ứng dụng: Gợi ý sản phẩm, nhận diện khuôn mặt

```
from sklearn.neighbors import KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1).fit([[0], [1]], [0, 1]).predict([[0.9]])  # → [1]
```

5: Học không giám sát – phân cụm (clustering)

5.1 Học không giám sát là gì?

Học không giám sát sử dụng **dữ liệu không có nhãn** nhằm khám phá các mẫu ẩn, cấu trúc hoặc nhóm tự nhiên trong dữ liệu.

Không giống như học có giám sát, mục tiêu ở đây là **tìm hiểu dữ liệu chứ không dự đoán cụ thể**.

5.2 Phân cụm với K-Means

K-Means là thuật toán phổ biến nhất để phân cụm dữ liệu. Nó chia tập dữ liệu thành **K nhóm** sao cho khoảng cách giữa các điểm trong cùng nhóm là nhỏ nhất.

Ứng dụng thực tế:

- Nhóm khách hàng theo hành vi mua sắm
- Phân đoạn hình ảnh
- Phát hiện bất thường

5.3 Cách hoạt động của K-Means

- Chọn số cụm K
- Khởi tạo K tâm cụm (centroids)
- Gán mỗi điểm dữ liệu vào cụm gần nhất
- Cập nhật tâm cụm mới
- Lặp lại cho đến khi hội tụ

5.4 Ví dụ ngắn với K-Means

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Dữ liệu đầu vào
X = [[1], [2], [8], [9]]

# Huấn luyện mô hình KMeans với 2 cụm
model = KMeans(n_clusters=2, random_state=0)
model.fit(X)

# Dự đoán cụm cho điểm mới
print(model.predict([[3]]))  # → [0] hoặc [1]
```

5.5 Đánh giá chất lượng phân cụm

sử dụng **Silhouette Coefficient** để đánh giá hiệu quả phân cụm. Giá trị gần 1 cho thấy phân cụm tốt.

6. Học tăng cường (Reinforcement Learning)

Học tăng cường (Reinforcement Learning- RL) là một nhánh của học máy, trong đó một tác nhân

(agent) học cách đưa ra quyết định bằng cách thử và sai trong một môi trường (environment). Mục tiêu

của tác nhân là tối ưu hóa phần thưởng tích lũy (cumulative reward) thông qua việc chọn các hành động

(actions) dựa trên trạng thái (states) của môi trường và một chính sách (policy).

6.1 Các khái niệm cốt lõi

Tác nhân (Agent): Thực thể đưa ra quyết định, ví dụ: một AI chơi cờ vua.

• Môi trường (Environment): Hệ thống mà tác nhân tương tác, ví dụ: bàn cờ hoặc thế giới vật

lý.

• Trạng thái (State): Mô tả tình trạng hiện tại của môi trường, ví dụ: vị trí của các quân cờ.

• Hành động (Action): Lựa chọn của tác nhân, ví dụ: di chuyển một quân cờ.

• Phần thưởng (Reward): Phản hồi từ môi trường, ví dụ: +1 nếu thắng,-1 nếu thua.

• Chính sách (Policy): Chiến lược của tác nhân để chọn hành động dựa trên trạng thái, thường

biểu diễn bằng $\pi(a|s)$.

Quy trình RL được minh họa bởi vòng lặp sau: tác nhân quan sát trạng thái st, chọn hành động at,

nhận phần thưởng rt và trạng thái tiếp theo st+1, sau đó cập nhật chính sách để cải thiện quyết định.

Sơ đồ minh họa:

Tác nhân ↔ Môi trường : (st,at,rt,st+1)

6.2 Các thuật toán phổ biến

• Q-Learning: Thuật toán dựa trên giá trị, cập nhật hàm giá trị hành động Q(s,a) theo công thức:

(

$Q(st,at) \leftarrow Q(st,at) + \alpha$

$rt + \gamma \max_a Q(st+1,a) - Q(st,at)$

Trong đó: α là tốc độ học, γ là hệ số chiết khấu.

)

• SARSA: Tương tự Q-Learning nhưng cập nhật dựa trên hành động thực sự được chọn.

• Deep Q-Networks (DQN): Kết hợp Q-Learning với mạng nơ-ron sâu để xử lý các trạng thái

phức tạp, như hình ảnh.

• Policy Gradient: Tối ưu hóa trực tiếp chính sách $\pi(a|s)$ bằng gradient descent.

• Proximal Policy Optimization (PPO): Thuật toán hiện đại, cân bằng giữa hiệu quả và ổn định.

6.3 Ví dụ minh họa: Q-Learning

Dưới đây là ví dụ về Q-Learning trong bài toán tác nhân di chuyển trong lưới 5x5 để tìm kho báu (phần

thưởng +100) và tránh chướng ngại vật (phần thưởng-1)

```
import numpy as np

R = np.array([
```

```
[-1,-1,-1,-1, 0],
[-1,-10,-1,-1, 0],
[-1,-1,-10,-1, 0],
[-1,-1,-1,-1, 0],
[0, 0, 0, 0, 100]
])

Q = np.zeros((5, 5))
1

alpha, gamma, episodes = 0.1, 0.9, 1000

for _ in range(episodes):
 state = np.random.randint(0, 5)
 while state != 4:
 action = np.random.randint(0, 5)
 next_state = action
 Q[state, action] += alpha * (R[state, action] + gamma * np.max(Q[next_state
 ])- Q[state, action])
 state = next_state
 print(Q)
```

Giải thích:

• Ma trận R định nghĩa phần thưởng: +100 tại kho báu,-10 tại chướng ngại vật, 0 khi đến trạng
thái tiếp theo,-1 cho các bước di chuyển khác.

• Matrận Q lưu trữ giá trị hành động, được cập nhật qua mỗi tập (episode).

• Tác nhân học cách chọn hành động để tối ưu hóa phần thưởng tích lũy.

6.4 Ứng dụng thực tế

Học tăng cường được ứng dụng rộng rãi trong nhiều lĩnh vực:

• Trò chơi: AlphaZero của DeepMind sử dụng RL để đạt hiệu suất siêu việt trong cờ vua và cờ vây.

• Robot: Điều khiển robot di chuyển trong môi trường không xác định, như robot hút bụi thông
minh.

• Quảng cáo: Tối ưu hóa chiến lược đặt giá thầu (bidding) trong quảng cáo trực tuyến.

• Giao thông thông minh: Tại Việt Nam, RL có thể được áp dụng để tối ưu hóa đèn giao thông,

giảm ùn tắc (ví dụ: nghiên cứu tại Đại học Bách Khoa TP.HCM).

• Nông nghiệp: Tối ưu hóa lịch tưới tiêu hoặc phân bón dựa trên dữ liệu thời tiết và đất đai

7: Triển khai và ứng dụng mô hình học máy

7.1 Tại sao cần triển khai mô hình?

Sau khi xây dựng và huấn luyện mô hình thành công, bước tiếp theo là **triển khai (deploy)** để người dùng hoặc hệ thống khác có thể sử dụng. Triển khai cho phép mô hình:

- Tích hợp vào ứng dụng web, mobile, hoặc hệ thống doanh nghiệp
- Nhận đầu vào thực tế và đưa ra dự đoán trong thời gian thực

7.2 Azure Machine Learning Studio – công cụ triển khai không cần mã

Azure ML Studio là một nền tảng **drag-and-drop** cho phép xây dựng, huấn luyện và triển khai mô hình học máy mà **không cần viết code**.

Tính năng nổi bật:

- Giao diện trực quan
- Tích hợp sẵn các thuật toán
- Cho phép triển khai mô hình như một dịch vụ web RESTful

7.3 Quy trình triển khai mô hình trên Azure ML Studio

- **Tải lên tập dữ liệu** (Upload Dataset)
- **Tạo mô hình** (Train Model)
- **Chia tập dữ liệu** thành train/test
- **So sánh hiệu suất** các thuật toán khác nhau
- **Xuất bản mô hình** như một Web Service
- **Gọi mô hình từ ứng dụng client (Python, web, mobile)**

7.4 Ví dụ triển khai mô hình đơn giản

```python
from sklearn.linear_model import LogisticRegression
import joblib  # Thư viện lưu mô hình

model = LogisticRegression().fit([[1], [2]], [0, 1])
joblib.dump(model, "model.pkl")  # Lưu mô hình

# Tải lại và dự đoán
loaded = joblib.load("model.pkl")
print(loaded.predict([[1.5]]))  # → [1]
```

8: Xu hướng phát triển hiện đại trong ai và machine learning

8.1 Sự chuyển dịch từ mô hình nhỏ sang mô hình lớn (LLM – Large Language Models)

Trong những năm gần đây, các mô hình ngôn ngữ lớn như **GPT (OpenAI)**, **PaLM (Google)**, **Claude (Anthropic)** hay **Gemini** đã tạo nên bước nhảy vọt trong khả năng xử lý và sinh ngôn ngữ tự nhiên. Những mô hình này:

- Có hàng **tỷ tham số**
- Được huấn luyện trên khối lượng dữ liệu khổng lồ
- Có khả năng sinh văn bản, dịch ngôn ngữ, viết mã, phân tích cảm xúc, v.v.

Ứng dụng: Chatbot, viết nội dung, hỗ trợ lập trình, phân tích văn bản doanh nghiệp

7.2 AI trên thiết bị di động và biên (Edge AI)

Nhờ cải tiến phần cứng và phần mềm, các mô hình AI có thể chạy trực tiếp **trên thiết bị di động** hoặc vi xử lý biên (edge device), giúp:

- Giảm độ trễ (real-time)
- Không phụ thuộc vào đám mây (bảo mật tốt hơn)
- Tăng hiệu quả năng lượng

Ví dụ: nhận dạng khuôn mặt trong điện thoại, phân loại ảnh offline, Google Lens, Apple Neural Engine

8.3 Tích hợp AI với Internet of Things (AIoT)

Sự kết hợp giữa **AI + IoT** tạo ra các hệ thống thông minh trong:

- Nhà thông minh (Smart Home)

- Thành phố thông minh (Smart City)
- Xe tự lái, giám sát giao thông

AI giúp phân tích dữ liệu cảm biến thời gian thực, tối ưu hóa hiệu suất và phát hiện bất thường.

8.4 AutoML và No-code/Low-code AI

AutoML giúp tự động hoá quy trình chọn mô hình, huấn luyện và tối ưu, phù hợp cho người không chuyên.

No-code AI cho phép kéo-thả xây dựng mô hình mà không cần viết mã, điển hình như:

- Microsoft Azure ML Studio
- Google AutoML
- Teachable Machine của Google

Xu hướng này giúp **doanh nghiệp nhỏ và nhà giáo dục** tiếp cận AI dễ dàng hơn.

7.5 AI đạo đức và bền vững (AI Ethics)

Song song với phát triển kỹ thuật, thế giới đang chú trọng đến:

- **Minh bạch mô hình (transparency)**
- **Loại bỏ định kiến dữ liệu (bias)**
- **Bảo vệ quyền riêng tư**
- **Giải thích được mô hình (explainable AI)**

Các tổ chức như EU AI Act, OECD, UNESCO đang thiết lập khung đạo đức và pháp lý cho AI.

9: Ứng dụng thực tế của trí tuệ nhân tạo trong các lĩnh vực

9.1 Y tế (Healthcare)

AI đang thay đổi ngành y tế với khả năng:

- **Chẩn đoán hình ảnh**: AI nhận diện tổn thương trong X-quang, MRI, CT với độ chính xác cao
- **Dự đoán bệnh lý**: phát hiện sớm tiểu đường, ung thư, Alzheimer
- **Y học cá nhân hóa**: phân tích gene và bệnh sử để đưa ra phác đồ điều trị tối ưu
- **Chatbot y tế**: như Babylon Health, hỗ trợ chẩn đoán ban đầu

Ví dụ: Google DeepMind xây dựng AI phát hiện bệnh võng mạc tiểu đường từ ảnh mắt.

9.2 Tài chính (Finance)

Trong ngành tài chính, AI giúp:

- **Phát hiện gian lận (fraud detection)** theo thời gian thực
- **Đánh giá tín dụng và chấm điểm rủi ro**
- **Tự động hóa giao dịch chứng khoán (algorithmic trading)**

- **Tư vấn tài chính ảo (robo-advisors)** như Betterment, Wealthfront

AI giúp giảm rủi ro và tối ưu hóa lợi nhuận cho cả ngân hàng và người dùng.

9.3 Thương mại điện tử (E-commerce)

AI được tích hợp trong gần như toàn bộ quy trình:

- **Hệ thống gợi ý sản phẩm** (Recommendation System) như của Amazon, Shopee
- **Phân tích hành vi khách hàng** để tối ưu trải nghiệm
- **Dự báo nhu cầu và quản lý tồn kho thông minh**
- **Trợ lý ảo và chatbot chăm sóc khách hàng**

Ví dụ: Netflix sử dụng AI để cá nhân hóa gợi ý phim cho từng người dùng.

9.4 Giao thông và xe tự hành

- **Xe tự lái (Autonomous Vehicles)** sử dụng AI để nhận diện làn đường, biển báo, người đi bộ, và tự ra quyết định lái
- **Dự báo lưu lượng giao thông**, đề xuất tuyến đường tối ưu
- **Phân tích video giám sát** để phát hiện vi phạm và ùn tắc

Tesla Autopilot, Waymo (Google), và VinAI tại Việt Nam đang phát triển mạnh mẽ trong lĩnh vực này.

9.5 Giáo dục và đào tạo

AI mang lại trải nghiệm học tập cá nhân hóa:

- **Chấm điểm tự động**, phát hiện đạo văn
- **Tư vấn học tập theo tiến độ cá nhân**
- **Mô phỏng, thực tế ảo kết hợp AI** trong dạy kỹ năng thực hành

Duolingo dùng AI để điều chỉnh bài học theo năng lực người học.

VD: VinAI đã phát triển công nghệ nhận diện khuôn mặt và xử lý ngôn ngữ tự nhiên, được ứng dụng trong các sản phẩm của Vingroup.”

Bổ sung số liệu: “Theo báo cáo của McKinsey (2023), AI có thể đóng góp thêm 13 nghìn tỷ USD vào nền kinh tế toàn cầu vào năm 2030.”

CHƯƠNG 2: CÁC KỸ THUẬT HỌC SÂU

1: Giới thiệu về học sâu

1.1. Trí tuệ nhân tạo, Học máy và Học sâu

Trí tuệ nhân tạo (**Artificial Intelligence – AI**) là lĩnh vực khoa học máy tính nghiên cứu về việc xây dựng các hệ thống có thể thực hiện hành vi trí tuệ giống con người. **Học máy (Machine Learning – ML)** là một nhánh của AI tập trung vào việc xây dựng các thuật toán giúp máy tính học từ dữ liệu. **Học sâu (Deep Learning – DL)** là một phân nhánh đặc biệt của ML, sử dụng các **mạng nơ-ron nhiều lớp** (deep neural networks) để học mô hình phức tạp.

Cấp độ	Mô tả
AI	Giả lập trí thông minh nói chung
ML	Học từ dữ liệu bằng thuật toán
DL	ML dựa trên mạng nơ-ron nhiều tầng (deep nets)

1.2. Tại sao học sâu lại quan trọng?

Học sâu mang lại kết quả vượt trội trong nhiều lĩnh vực mà các kỹ thuật ML truyền thống gặp khó khăn, ví dụ:

- Nhận dạng hình ảnh
- Xử lý ngôn ngữ tự nhiên
- Dự báo chuỗi thời gian
- Tổng hợp và sinh nội dung

Ba lý do chính cho sự bùng nổ của học sâu:

1. **Dữ liệu lớn (Big Data)**: có nhiều dữ liệu hơn để huấn luyện
2. **Phần cứng mạnh mẽ**: GPU, TPU cho phép huấn luyện nhanh
3. **Thuật toán hiệu quả hơn**: như ReLU, batch normalization, optimizer tốt hơn

1.3. Học sâu khác gì so với ML truyền thống?

Machine Learning truyền thống	Deep Learning
Cần thiết kế đặc trưng thủ công (feature engineering)	Mô hình tự học đặc trưng từ dữ liệu
Phụ thuộc vào kiến thức chuyên ngành	Tự động trích xuất biểu diễn
Yêu cầu ít dữ liệu hơn	Hiệu quả cao với dữ liệu lớn

1.4. Lược sử mạng nơ-ron và học sâu

- 1950s–1980s: perceptron, mạng nơ-ron 1 lớp

- 1986: thuật toán backpropagation
- 2006: Geoffrey Hinton khởi động lại học sâu bằng **deep belief networks**
- 2012: Mạng **AlexNet** chiến thắng ImageNet → kỷ nguyên deep learning bắt đầu

1.5 Các kiến trúc mạng nơ-ron phổ biến

Mạng nơ-ron sâu (Deep Neural Networks) là nền tảng của học sâu, với các kiến trúc khác nhau được thiết kế để giải quyết các bài toán cụ thể. Phần này giới thiệu ba kiến trúc phổ biến: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), và Transformer, cùng với các ví dụ mã nguồn và ứng dụng thực tế

1.5.1 Convolutional Neural Networks (CNN)

CNN được thiết kế để xử lý dữ liệu có cấu trúc không gian, như hình ảnh. CNN sử dụng các lớp

convolution (Conv2D) để trích xuất đặc trưng (như cạnh, góc) và các lớp pooling (MaxPooling2D) để

giảm kích thước dữ liệu.

Cấu trúc:

$$\text{Input} \rightarrow [\text{Conv2D} \rightarrow \text{ReLU} \rightarrow \text{Pooling}] \times N \rightarrow \text{Flatten} \rightarrow \text{Dense} \rightarrow \text{Output}$$

Công thức convolution:

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Trong đó: I là ảnh đầu vào, K là kernel, s(i,j) là giá trị tại vị trí (i,j).

Ví dụ mã nguồn:

```
from tensorflow.keras import layers, models
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Ứng dụng:

• Phân loại hình ảnh (như chó/mèo trong Chương 3).

• Nhận diện biển số xe tại Việt Nam (ví dụ: hệ thống giám sát giao thông của FPT).

• Chẩn đoán y tế qua ảnh X-quang hoặc MRI.

Biến thể: LeNet (1998), VGG16, ResNet

1.5.2 Recurrent Neural Networks (RNN)

RNN được thiết kế để xử lý dữ liệu tuần tự, như chuỗi thời gian hoặc văn bản. RNN duy trì một trạng

thái ẩn (hidden state) để ghi nhớ thông tin từ các bước trước.

Cấu trúc

$$\text{Input} \to [\text{RNN Cell}] \times T \to \text{Dense} \to \text{Output}$$

Công thức RNN:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$$

Trong đó: ht là trạng thái ẩn tại thời điểm t, xt là đầu vào, Wh, Wx là trọng số.

Ví dụ mã nguồn:

```
from tensorflow.keras import layers, models
model = models.Sequential([
layers.LSTM(64, input_shape=(10, 1)), % 10 time steps, 1 feature
layers.Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mse')
```

Ứng dụng:

• Dự đoán giá cổ phiếu dựa trên dữ liệu lịch sử.

• Dịch máy (ví dụ: dịch tiếng Việt sang tiếng Anh).

• Phân tích cảm xúc văn bản (sentiment analysis) trong các ứng dụng như Zalo.

Biến thể: LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit).

1.6. Ví dụ: Mạng phân loại ảnh mèo/chó đơn giản với Keras

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(512, activation="relu", input_shape=(10000,)),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

Đây là một mô hình học sâu đơn giản với 2 lớp (fully connected).
Nó có thể dùng cho bài toán phân loại nhị phân, ví dụ như:

- Mèo hay chó
- Tích cực hay tiêu cực (cảm xúc)
- Có bệnh hay không bệnh

2: Các nền tảng toán học của học sâu

2.1 Tensor là gì?

Tensor là cấu trúc dữ liệu cốt lõi trong học sâu – có thể xem như **một mảng số đa chiều**.

- **Scalar** (số đơn) là tensor bậc 0
- **Vector** là tensor bậc 1
- **Ma trận** là tensor bậc 2
- Tensor bậc 3 trở lên thường dùng trong ảnh màu, chuỗi dữ liệu,...

📌 Ví dụ: ảnh màu RGB 128×128 là tensor có shape (128, 128, 3)

2.2 Xem tensor bằng NumPy

```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
print("Shape:", x.shape)     # (3, 2)
print("Rank:", x.ndim)       # 2 (bậc 2 - ma trận)
```

2.3 Phép biến đổi tensor

Tensor có thể được biến đổi hình dạng hoặc đảo trục:

```
x = np.array([[1, 2], [3, 4]])
x_reshaped = x.reshape((4, 1))     # reshape về dạng cột
x_transposed = x.T                 # chuyển vị
print(x_reshaped)
```

Slicing (cắt) cũng tương tự như list Python:

```
x = np.array([[1, 2, 3], [4, 5, 6]])
print(x[:, 1])     # → [2 5]
```

2.4 Phép toán đại số tuyến tính

Các mô hình học sâu thường dùng **phép nhân ma trận (dot product)**, thay vì chỉ nhân từng phần tử:

Các phép toán đại số tuyến tính, như nhân ma trận và convolution, là cốt lõi của học sâu. Nhân ma

trận được dùng trong các lớp fully-connected, còn convolution được dùng trong CNN.

Công thức nhân ma trận:

$$C = A \cdot B, \quad c_{ij} = \sum_k a_{ik} b_{kj}$$

Công thức convolution

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[2, 0], [1, 3]])
print(np.dot(a, b))  # Phép nhân ma trận
```

Ngoài ra còn có:

- **Cộng, trừ, nhân phần tử:** a * b, a + b
- **Tích vô hướng (dot product):** np.dot(x, y)
- **Broadcasting:** tự mở rộng kích thước để tính toán

2.5 Hàm kích hoạt (activation function)

- **ReLU**: $f(x) = \max(0, x)$
- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
- **Softmax**: $f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ (dùng cho phân loại đa lớp)

Được dùng để **giới thiệu tính phi tuyến** vào mạng nơ-ron.

Hàm	Mục đích
`relu(x)`	Thường dùng nhất, tránh gradient biến mất
`sigmoid(x)`	Co đầu ra vào khoảng (0,1)
`tanh(x)`	Co đầu ra vào khoảng (-1,1)

2.6 Gradient và lan truyền ngược

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

- **Gradient** là độ dốc của hàm số – cho biết hàm tăng hay giảm theo hướng nào
- Trong học sâu, gradient cho biết cần điều chỉnh trọng số ra sao để **giảm sai số (loss)**
- **Lan truyền ngược (backpropagation)** là thuật toán cốt lõi, dùng để **tính gradient toàn mạng** qua đạo hàm từng lớp (dùng chuỗi đạo hàm – chain rule)

Trong thực tế, bạn không cần tính gradient bằng tay – TensorFlow/Keras sẽ tự động làm điều đó với model.compile() và fit().

3: Xây dựng và huấn luyện mô hình học sâu với keras

3.1 Kiến trúc cơ bản của một mô hình học sâu trong Keras

Trong Keras, mô hình được xây dựng theo cách **tuần tự** (Sequential) hoặc **chức năng** (Functional API). Ở đây, ta bắt đầu với Sequential, gồm:

1. **Lớp đầu vào (Input layer)**
2. **Các lớp ẩn (Hidden layers)**
3. **Lớp đầu ra (Output layer)**

Các lớp phổ biến:

- Dense: fully-connected
- Dropout: chống overfitting
- Embedding: xử lý văn bản
- Activation: hàm phi tuyến

3.2 Quy trình huấn luyện mô hình với Keras

Quy trình chuẩn gồm 4 bước:

1. **Xây dựng mô hình (model = Sequential(...))**
2. **Biên dịch mô hình (model.compile(...))**
3. **Huấn luyện mô hình (model.fit(...))**
4. **Đánh giá mô hình (model.evaluate(...))**

3.3 Thực hành 1: Phân loại cảm xúc từ đánh giá phim (IMDB – binary classification)

Mục tiêu: Dự đoán xem một đánh giá phim là tích cực hay tiêu cực (1 hoặc 0)
Dữ liệu: Tập IMDB có sẵn trong keras.datasets.imdb với văn bản đã được mã hóa thành số nguyên.

```
from tensorflow import keras
from tensorflow.keras import layers

# Tải dữ liệu
(train_data, train_labels), (test_data, test_labels) = keras.datasets.imdb.load_data(num_words=10

# Mã hóa dữ liệu đầu vào thành vector nhị phân (bag-of-words)
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.0
    return results
```

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

```python
# Xây dựng mô hình
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

# Biên dịch mô hình
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

# Huấn luyện mô hình
history = model.fit(x_train, y_train, epochs=4, batch_size=512, validation_split=0.2)
```

3.4 Thực hành 2: Phân loại tin tức theo chủ đề (Reuters – multi-class classification)

Mục tiêu: Phân loại bài viết thành một trong 46 chủ đề

Dữ liệu: keras.datasets.reuters

```python
# Tải dữ liệu
(train_data, train_labels), (test_data, test_labels) = keras.datasets.reuters.load_data(num_words

# Vector hóa dữ liệu đầu vào
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

# One-hot mã hóa nhãn đầu ra
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.0
    return results
```

```python
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
```

```python
# Mô hình nhiều lớp đầu ra
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])

# Biên dịch mô hình
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])

# Huấn luyện
history = model.fit(x_train, y_train, epochs=8, batch_size=512, validation_split=0.2)
```

4: Điều chỉnh mô hình và kiểm soát overfitting

4.1 Overfitting và Underfitting là gì?

- **Overfitting** xảy ra khi mô hình học quá kỹ dữ liệu huấn luyện, kể cả nhiễu → hiệu suất thấp trên dữ liệu mới.
- **Underfitting** là khi mô hình quá đơn giản → không học được mẫu từ dữ liệu huấn luyện.

Dấu hiệu nhận biết:

- Độ chính xác trên *val* giảm trong khi *train* vẫn tăng → **overfitting**
- Cả *train* và *val* đều có độ chính xác thấp → **underfitting**

4.2 Kỹ thuật phổ biến để giảm overfitting

1. Sử dụng mô hình nhỏ hơn

Giảm số lượng lớp hoặc số đơn vị (units) trong các lớp:

```python
model = keras.Sequential([
    layers.Dense(4, activation="relu"),  # nhỏ hơn
    layers.Dense(4, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

2. Regularization (chuẩn hóa trọng số)

Weight decay – thêm hệ số phạt vào hàm mất mát (L1 hoặc L2):

```
from tensorflow.keras import regularizers

model = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
```

3. Dropout

Dropout là kỹ thuật **"vô hiệu hóa ngẫu nhiên"** một số node trong quá trình huấn luyện → tránh phụ thuộc quá mức vào một tập con nhỏ neuron.

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
```

4. Giám sát quá trình huấn luyện bằng validation_loss

Theo dõi loss và accuracy trên tập validation để biết khi nào mô hình bắt đầu overfit:

```
history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_split=0.2)
```

Sau đó trực quan hóa:

```
import matplotlib.pyplot as plt

val_loss = history.history["val_loss"]
loss = history.history["loss"]
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

5: Làm việc với dữ liệu ảnh và mạng tích chập (cnn)

5.1 Thị giác máy tính và vai trò của CNN

CNN (Convolutional Neural Networks – mạng tích chập) là nền tảng trong học sâu cho bài toán thị giác máy tính (computer vision). Chúng xử lý dữ liệu ảnh tốt hơn nhờ:

- Giữ nguyên thông tin không gian (khác với Dense)
- Tự động trích xuất đặc trưng (features)
- Ít tham số hơn → ít overfitting hơn

5.2 Cấu trúc cơ bản của CNN

Một CNN thường gồm:

Lớp	Mục đích
`Conv2D`	Lọc đặc trưng từ ảnh
`MaxPooling2D`	Giảm chiều không gian (downsampling)
`Flatten`	Chuyển ảnh 2D thành vector 1D để đưa vào `Dense`
`Dense`	Dự đoán đầu ra (phân loại, hồi quy, v.v.)

5.3 Ví dụ: Nhận diện mèo/chó từ ảnh (Dogs vs Cats)

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

5.4 Biên dịch và huấn luyện mô hình ảnh

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

5.5 Tăng cường dữ liệu (Data Augmentation)

Tăng cường dữ liệu giúp mô hình không overfit bằng cách **sinh ảnh mới từ ảnh gốc** (xoay, lật, zoom,...)

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

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

5.6 Trích đặc trưng với mô hình pretrained (Feature Extraction)

Thay vì huấn luyện từ đầu, ta có thể **dùng mạng đã huấn luyện (như VGG16)** để trích đặc trưng:

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
```

Ta "đóng băng" mạng gốc (không huấn luyện lại), sau đó thêm vài lớp Dense để phân loại theo bài toán mới.

6: Làm việc với chuỗi và dữ liệu văn bản

6.1 Tại sao dữ liệu chuỗi cần xử lý đặc biệt?

Khác với ảnh (ma trận cố định), dữ liệu chuỗi như văn bản, thời gian, âm thanh... có **độ dài thay đổi** và **phụ thuộc theo thứ tự**.

Ví dụ:

- “Tôi thích mèo” khác hoàn toàn “Mèo thích tôi”

Do đó, ta cần kỹ thuật đặc biệt để:

- Giữ **thứ tự** của dữ liệu
- Mã hóa chuỗi văn bản thành dạng số

6.2 Biến văn bản thành số: Tokenization và Padding

Tokenization

Chuyển văn bản thành chuỗi số (mỗi số đại diện 1 từ):

```
from tensorflow.keras.preprocessing.text import Tokenizer

samples = ["I love cats", "You love dogs"]
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)
sequences = tokenizer.texts_to_sequences(samples)
```

Padding

Các chuỗi thường có độ dài khác nhau, cần được **chuẩn hóa độ dài**:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded = pad_sequences(sequences, maxlen=5)
```

6.3 Biểu diễn từ: Word Embedding

Embedding là kỹ thuật chuyển từ (token) thành **vector đặc trưng dense** thay vì one-hot. Keras cung cấp lớp Embedding:

```
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Embedding(input_dim=10000, output_dim=8, input_length=100),
    layers.Flatten(),  # hoặc kết hợp với RNN
    layers.Dense(1, activation="sigmoid")
])
```

6.4 Mạng hồi tiếp (RNN, LSTM, GRU)

RNN (Recurrent Neural Network) là lớp đặc biệt phù hợp với dữ liệu chuỗi. Tuy nhiên, RNN đơn giản bị **mất thông tin xa**. Do đó, ta thường dùng:

Mạng	Ưu điểm
`SimpleRNN`	Dễ huấn luyện, đơn giản
`LSTM`	Ghi nhớ thông tin dài hạn tốt
`GRU`	Nhẹ hơn LSTM, hiệu quả tương đương

Ví dụ mô hình phân loại chuỗi cảm xúc:

```
model = keras.Sequential([
    layers.Embedding(input_dim=10000, output_dim=32),
    layers.LSTM(32),
    layers.Dense(1, activation="sigmoid")
])
```

6.5 Dự đoán chuỗi thời gian

Ngoài văn bản, RNN còn được dùng cho chuỗi thời gian:

- Dự đoán thời tiết
- Giá cổ phiếu
- Cảm biến IoT

Cần chuẩn hóa dữ liệu đầu vào (min-max scaling), chia chuỗi thành "cửa sổ thời gian" để huấn luyện.

7: Xử lý ngôn ngữ tự nhiên hiện đại với transformer

7.1 Giới thiệu: Vì sao cần Transformer?

Các mô hình RNN (LSTM, GRU) hoạt động theo thứ tự tuần tự – dẫn đến:

- Khó song song hóa
- Hiệu suất thấp với chuỗi dài

Transformer ra đời (2017, paper *Attention is All You Need*) đã thay đổi cuộc chơi:

- Xử lý toàn bộ chuỗi song song
- Hiệu quả cao hơn RNN trên hầu hết bài toán NLP

7.2 Tư tưởng cốt lõi: Self-Attention

Self-attention cho phép mỗi từ trong chuỗi nhìn toàn bộ các từ khác để hiểu ngữ cảnh tốt hơn.

7.3 Kiến trúc Transformer cơ bản

Một khối Transformer gồm:

- **Multi-head attention** (gồm nhiều self-attention song song)
- **Normalization + Residual connection**
- **Feedforward network**

```
Input Embedding → Positional Encoding →
[Multi-Head Attention → Add & Norm →
 Feedforward → Add & Norm]
```

7.4 Positional Encoding

Transformer không có khái niệm "thứ tự" như RNN.
→ **Positional Encoding** giúp mô hình hiểu vị trí từ bằng cách cộng thêm vector vị trí vào embedding.

7.5 Sử dụng Transformer trong Keras

Keras hiện hỗ trợ Transformer thông qua:

- keras_nlp
- MultiHeadAttention trong tensorflow.keras.layers

Ví dụ khởi tạo layer attention:

```
from tensorflow.keras.layers import MultiHeadAttention

attention = MultiHeadAttention(num_heads=2, key_dim=64)
output = attention(query=x, value=x, key=x)
```

7.6 Mô hình lớn: BERT và GPT (tổng quan)

BERT (Bidirectional Encoder Representations from Transformers)

- Dự đoán từ bị che trong ngữ cảnh 2 chiều
- Mạnh trong phân loại, tìm kiếm, QA

GPT (Generative Pretrained Transformer)

- Tự sinh văn bản (từng token một)
- Mạnh trong sinh ngôn ngữ, hội thoại, viết mã

8: Học sâu sinh sinh (generative deep learning)

8.1 Tổng quan về học sâu sinh sinh

Học sâu sinh sinh là lĩnh vực mà **mô hình không chỉ nhận diện hoặc phân loại**, mà còn **tạo ra dữ liệu mới**, như:

- Sinh văn bản (text generation)

- Tạo ảnh mới (GAN)
- Tạo mô hình nén và giải nén (Autoencoder, VAE)

8.2 Sinh văn bản tuần tự (Text Generation)

Mô hình LSTM hoặc GRU có thể học **phân phối xác suất của chuỗi ký tự** để sinh văn bản giống ngữ liệu huấn luyện (ví dụ: thơ, văn Shakespeare, mã nguồn...).

Ví dụ: Sinh văn bản theo ký tự (character-level)

```
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Embedding(input_dim=65, output_dim=16),  # 65 ký tự trong tập
    layers.LSTM(128, return_sequences=True),
    layers.LSTM(128),
    layers.Dense(65, activation="softmax")  # dự đoán ký tự tiếp theo
])
```

8.3 Style Transfer – Truyền phong cách hình ảnh

Style transfer dùng mạng CNN để kết hợp:

- Nội dung của ảnh A
- Phong cách của ảnh B
  → tạo ra ảnh mới giữ nguyên nội dung A, nhưng mang phong cách của B

Kỹ thuật này sử dụng:

- VGG19 làm mạng trích đặc trưng
- Tối ưu ảnh đầu ra để khớp cả content loss và style loss

8.4 Autoencoder – học biểu diễn nén

Autoencoder gồm:

- **Encoder**: nén đầu vào thành vector ngắn (latent vector)
- **Decoder**: tái tạo lại đầu vào từ vector này

Mục đích:

- Giảm chiều dữ liệu
- Phát hiện bất thường
- Tái tạo hình ảnh

```python
# Autoencoder cơ bản
encoder = keras.Sequential([
    layers.Dense(32, activation="relu", input_shape=(784,))  # MNIST
])

decoder = keras.Sequential([
    layers.Dense(784, activation="sigmoid")
])

autoencoder = keras.Sequential([encoder, decoder])
```

8.5 Variational Autoencoder (VAE)

VAE là phiên bản **xác suất** của autoencoder:

- Thay vì học vector cụ thể → học **phân phối Gaussian** trên latent space
- Giúp sinh dữ liệu mới bằng cách lấy mẫu từ latent space

Tính chất đặc biệt:

- Có thể **nội suy (interpolate)** giữa các điểm trong không gian tiềm ẩn
- Mô hình có khả năng sinh ảnh, văn bản gần giống dữ liệu gốc

8.6 Generative Adversarial Networks (GAN)

GAN gồm hai thành phần:

Thành phần	Vai trò
Generator	Sinh dữ liệu giả từ nhiễu (noise)
Discriminator	Phân biệt thật/giả

Hai mô hình này học song song qua trò chơi "đối kháng":

- Generator cố đánh lừa discriminator
- Discriminator cố phân biệt thật/giả

Kết quả: mô hình sinh ra ảnh rất giống thật (khuôn mặt, đồ vật...)

9: Kỹ thuật nâng cao và tổng hợp trong học sâu

9.1 Tổng quan bức tranh học sâu hiện đại

Deep learning đã trở thành công nghệ chủ chốt trong các hệ thống thông minh hiện đại, nhờ vào:

- Khả năng tổng quát hóa với dữ liệu lớn

- Kết quả vượt trội so với ML truyền thống trong hình ảnh, văn bản, âm thanh

Tuy nhiên, để đạt kết quả tốt thực tế, cần kết hợp nhiều kỹ thuật như:

- Kỹ thuật sinh sinh (VAE, GAN)
- Mạng chuyển đổi (Transformers)
- Kết hợp nhiều nguồn dữ liệu (multi-modal learning)

9.2 Học tăng cường (Reinforcement Learning – RL)

Mặc dù sách không đi sâu vào RL, tác giả mô tả RL là hướng đi đầy hứa hẹn cho các bài toán tương tác như:

- Game (AlphaGo, Dota 2)
- Robotics
- Tối ưu hóa chuỗi hành động

RL không học từ nhãn, mà từ **phản hồi dạng phần thưởng** – khác với supervised learning.

9.3 Học suốt đời (Continual / Lifelong Learning)

Học sâu hiện tại thường:

- Được huấn luyện trên toàn bộ dữ liệu trước khi dùng
- Không thể học thêm dữ liệu mới mà không "quên" cái cũ (catastrophic forgetting)

Continual learning hướng đến khả năng:

- Học từ dữ liệu theo thời gian
- Giữ lại kiến thức cũ khi học thêm mới

Ứng dụng thực tế: hệ thống AI cập nhật theo người dùng, học từ tương tác liên tục.

9.4 Học mô-đun (Modular Deep Learning)

Ý tưởng chính:

- Thay vì một mô hình lớn đơn lẻ, sử dụng **các mô-đun nhỏ** chuyên biệt cho từng phần việc
- Dễ tái sử dụng, huấn luyện nhanh hơn, dễ giải thích

Ví dụ: hệ thống xe tự lái có thể chia thành các mô-đun:

- Phát hiện làn đường
- Dự đoán chuyển động người đi bộ
- Ra quyết định

9.5 Tổng hợp lại các kỹ thuật học sâu

Lĩnh vực	Kỹ thuật học sâu liên quan
Hình ảnh	CNN, Data Augmentation, Pretrained Models
Văn bản	Embedding, LSTM, Transformer, BERT
Sinh dữ liệu	GAN, VAE, Autoencoder
Dự báo thời gian	LSTM, RNN, Windowing
Tối ưu hành vi	Reinforcement Learning

10: Giới hạn, thách thức và triển vọng của học sâu

10.1 Những giới hạn hiện tại của học sâu

Mặc dù rất mạnh mẽ, học sâu vẫn đối mặt với nhiều giới hạn căn bản:

1. Phụ thuộc dữ liệu lớn

Học sâu yêu cầu hàng triệu mẫu huấn luyện để hoạt động tốt. Điều này khiến nó khó áp dụng trong các lĩnh vực:

- Y tế hiếm dữ liệu
- Nhiệm vụ tùy chỉnh nhỏ

2. Không ổn định và khó tái tạo

Việc huấn luyện có thể không nhất quán:

- Nhạy với ngẫu nhiên (khởi tạo, shuffle)
- Phụ thuộc lớn vào kỹ thuật huấn luyện và tuning

3. "Hộp đen" – Thiếu khả năng giải thích

Mô hình học sâu khó hiểu, khó dự đoán nội tại → gây rủi ro trong ứng dụng quan trọng như:

- Y tế
- Pháp lý
- Tài chính

10.2 Hướng phát triển khắc phục hạn chế

Học với ít dữ liệu (few-shot, one-shot)

- Hướng đến khả năng tổng quát tốt hơn với ít ví dụ
- Kết hợp học sâu với kỹ thuật logic, mô hình hóa quy tắc

Học "từ đầu" (from scratch) với dữ liệu phi cấu trúc

- Hướng đến mô hình không cần xử lý đặc biệt đầu vào (giống như con người)

Học tự giám sát (Self-supervised learning)

- Không cần nhãn thủ công → tự học từ cấu trúc nội tại của dữ liệu
- Là nền tảng cho BERT, GPT...

10.3 Triển vọng lâu dài của học sâu

- **Học mô-đun, có thể tái sử dụng** như phần mềm
- **Học có thể tương tác** với thế giới thật (robotics, AGI)
- **Hệ thống lai (hybrid)**: kết hợp deep learning với lập luận logic, cơ sở tri thức

CHƯƠNG 3: ỨNG DỤNG HỌC SÂU CHO BÀI TOÁN ĐOÁN ẢNH CHÓ VÀ MÈO

Nguồn dữ liệu : https://www.kaggle.com/competitions/dogs-vs-cats/data?select=sampleSubmission.csv

Ta có 2 file dữ liệu :

- 1 file train gồm 12500 ảnh chó và 12500 ảnh mèo dùng để huấn luyện và đánh giá mô hình.
- 1 file với 12500 ảnh chó mèo lẫn lộn không dán nhán để mô hình đoán ảnh và kiểm tra độ hiệu quả của mô hình

1.Giải nén 2 file zip đã tải xuống vào 2 thư mục train và test

```python
import zipfile
import os

# Đường dẫn thư mục hiện tại
base_dir = os.getcwd()

# Giải nén train.zip
train_zip_path = os.path.join(base_dir, 'train.zip')
with zipfile.ZipFile(train_zip_path, 'r') as zip_ref:
    zip_ref.extractall(os.path.join(base_dir, 'dataset/train'))

# Giải nén test.zip
test_zip_path = os.path.join(base_dir, 'test.zip')
with zipfile.ZipFile(test_zip_path, 'r') as zip_ref:
    zip_ref.extractall(os.path.join(base_dir, 'dataset/test'))
```

2. Phân ảnh trong file Train thành 2 file : 1 file 12500 ảnh chó và 1 file 12500 ảnh mèo

[1]:
```python
import os
import shutil

# Tạo thư mục mới để lưu ảnh phân loại
cat_dir = 'dataset/train/cats'
dog_dir = 'dataset/train/dogs'

os.makedirs(cat_dir, exist_ok=True)
os.makedirs(dog_dir, exist_ok=True)

# Duyệt qua toàn bộ file trong train/
for filename in os.listdir('dataset/train'):
    if filename.startswith('cat'):
        shutil.move(os.path.join('dataset/train', filename), os.path.join(cat_dir, filename))
    elif filename.startswith('dog'):
        shutil.move(os.path.join('dataset/train', filename), os.path.join(dog_dir, filename))

print("Hoàn tất phân loại ảnh!")
```

3. Chia tập train thành 2 tập : 1 tập huấn luyện và 1 tập kiểm tra

Huấn luyên chiếm 80% : Trong tập huấn luyện vẫn có 2 file : 1 file khoảng 10000 ảnh chó và 1 file khoảng 10000 ảnh mèo

Kiểm tra chiếm 20%: Trong tập kiểm tra vẫn có 2 file : 1 file khoảng 2500 ảnh chó và 1 file khoảng 2500 ảnh mèo

```python
import os
import shutil
import random

# Đường dẫn gốc
base_dir = 'dataset'

# Tạo thư mục val/cats và val/dogs
val_cats_dir = os.path.join(base_dir, 'val/cats')
val_dogs_dir = os.path.join(base_dir, 'val/dogs')
os.makedirs(val_cats_dir, exist_ok=True)
os.makedirs(val_dogs_dir, exist_ok=True)

# Tỷ lệ chia: 80% train, 20% validation
val_split = 0.2

# Hàm thực hiện chia dữ liệu
def split_data(src_dir, dst_dir, split_ratio):
    all_files = os.listdir(src_dir)
    all_files = [f for f in all_files if os.path.isfile(os.path.join(src_dir, f))]
    random.shuffle(all_files)

    val_size = int(len(all_files) * split_ratio)
    val_files = all_files[:val_size]

    for file in val_files:
        shutil.move(os.path.join(src_dir, file), os.path.join(dst_dir, file))

# Tách mèo
split_data(os.path.join(base_dir, 'train/cats'), val_cats_dir, val_split)

# Tách chó
split_data(os.path.join(base_dir, 'train/dogs'), val_dogs_dir, val_split)

print("✅ Hoàn tất chia dữ liệu thành tập train và validation.")
```

```
✅ Hoàn tất chia dữ liệu thành tập train và validation.
```

4.Tải ảnh lên để chuẩn bị cho huấn luyện:

[4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Tạo bộ sinh ảnh cho train và validation
train_datagen = ImageDataGenerator(rescale=1./255)  # chuẩn hóa ảnh về [0,1]
val_datagen = ImageDataGenerator(rescale=1./255)

# Tải ảnh train từ thư mục
train_generator = train_datagen.flow_from_directory(
    'dataset/train',
    target_size=(150, 150),   # resize ảnh về 150x150
    batch_size=32,
    class_mode='binary'       # vì chỉ có 2 lớp: mèo (0), chó (1)
)

# Tải ảnh validation từ thư mục
val_generator = val_datagen.flow_from_directory(
    'dataset/val',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

5.Xây dựng mô hình CNN

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Xây dựng mô hình CNN
model = Sequential()

# Lớp Convolutional đầu tiên với 32 filters và kích thước kernel 3x3
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(2, 2))

# Lớp Convolutional thứ 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))

# Lớp Convolutional thứ 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))

# Chuyển ảnh thành vector 1D
model.add(Flatten())

# Lớp Fully Connected
model.add(Dense(512, activation='relu'))

# Lớp Output: phân loại 2 lớp (mèo hoặc chó)
model.add(Dense(1, activation='sigmoid'))

# Tóm tắt mô hình
model.summary()
```

Đây là kết quả tóm tắt mô hình

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 512)	18,940,416
dense_1 (Dense)	(None, 1)	513

Total params: 19,034,177 (72.61 MB)

Trainable params: 19,034,177 (72.61 MB)

Non-trainable params: 0 (0.00 B)

6.Biên dịch và huấn luyện mô hình

6.1,Thử với epoch = 5, steps per epoch =100, validation_step = 50

[24]:
```
# Biên dịch mô hình
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Huấn luyện mô hình
history = model.fit(
    train_generator,            # Bộ sinh dữ liệu train
    steps_per_epoch=100,        # Số bước mỗi epoch (100 bước mỗi epoch)
    epochs=5,                   # Số lần huấn luyện qua toàn bộ dữ liệu
    validation_data=val_generator,  # Bộ sinh dữ liệu validation
    validation_steps=50         # Số bước mỗi epoch cho validation
)

# Lưu mô hình sau khi huấn luyện
model.save('cats_and_dogs_model.h5')

print("✅ Huấn luyện hoàn tất!")
```

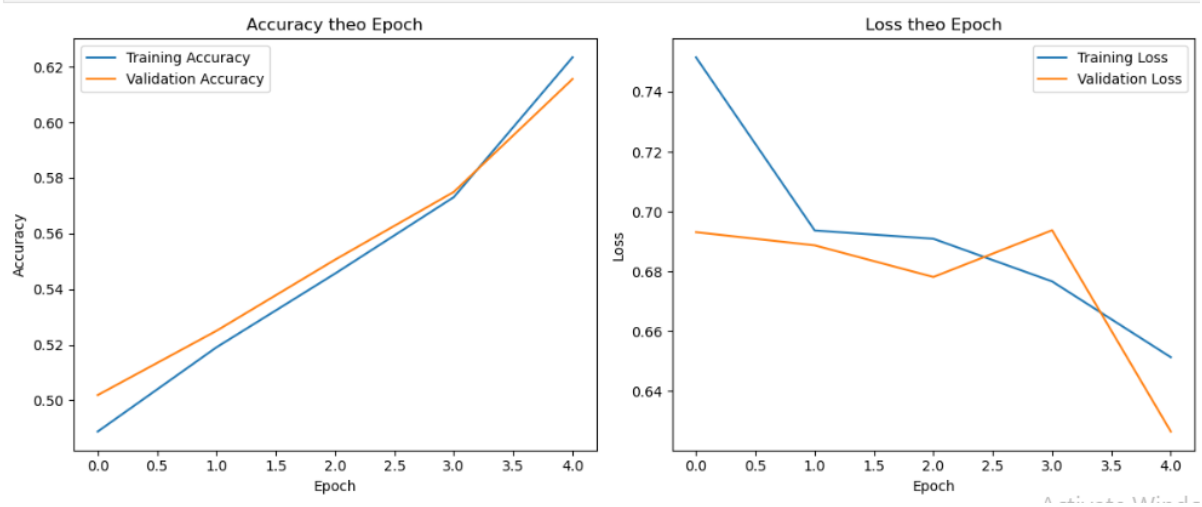
Kết quả:

```
Epoch 1/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 175s 2s/step - accuracy: 0.4853 - loss: 0.9298 - val_accuracy: 0.5019 - val_loss: 0.6931
Epoch 2/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 234s 2s/step - accuracy: 0.5140 - loss: 0.6937 - val_accuracy: 0.5250 - val_loss: 0.6887
Epoch 3/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 273s 3s/step - accuracy: 0.5505 - loss: 0.6919 - val_accuracy: 0.5506 - val_loss: 0.6781
Epoch 4/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 406s 4s/step - accuracy: 0.5553 - loss: 0.6819 - val_accuracy: 0.5750 - val_loss: 0.6937
Epoch 5/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 238s 2s/step - accuracy: 0.6058 - loss: 0.6664 - val_accuracy: 0.6156 - val_loss: 0.6264
```

Vẽ biểu đồ để đánh giá mức độ hiệu quả của mô hình:

[7]:
```python
import matplotlib.pyplot as plt

# Vẽ Accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy theo Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Vẽ Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss theo Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Kết quả:



Nhận xét:

- **Mô hình đang học được**, nhưng tốc độ **tăng khá chậm**.
    - Accuracy và loss cải thiện qua từng epoch, nhưng chưa mạnh.
- **Validation accuracy vẫn thấp** (≈ 61%) → mô hình chưa đủ mạnh để phân loại tốt ảnh mèo/chó.
- **Validation loss dao động** (đặc biệt ở Epoch 4) → có thể do:
    - Mô hình chưa đủ phức tạp.
    - Dữ liệu chưa đủ đa dạng (cần augmentation tốt hơn).
    - Số epoch còn ít.

6.2,Thử với epoch = 10 , steps per epoch =100, validation_step = 50

[6]:
```
# Biên dịch mô hình
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Huấn luyện mô hình
history = model.fit(
    train_generator,            # Bộ sinh dữ liệu train
    steps_per_epoch=100,        # Số bước mỗi epoch (100 bước mỗi epoch)
    epochs=10,                  # Số lần huấn luyện qua toàn bộ dữ liệu
    validation_data=val_generator,  # Bộ sinh dữ liệu validation
    validation_steps=50         # Số bước mỗi epoch cho validation
)

# Lưu mô hình sau khi huấn luyện
model.save('cats_and_dogs_model.h5')

print("✅ Huấn luyện hoàn tất!")
```

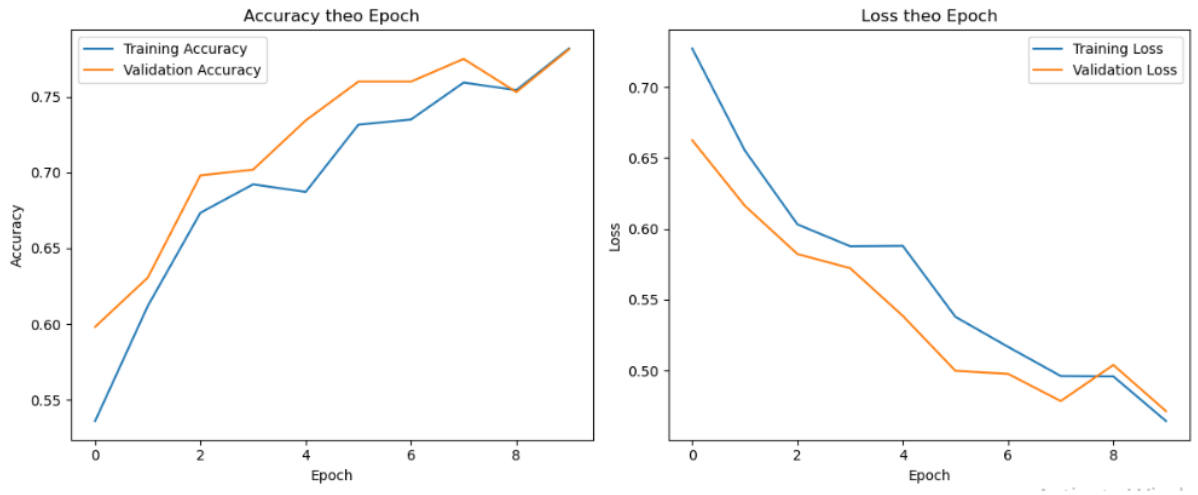
Ta được kết quả :

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 230s 2s/step - accuracy: 0.5062 - loss: 0.8691 - val_accuracy: 0.5981 - val_loss: 0.6625
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 187s 2s/step - accuracy: 0.5967 - loss: 0.6628 - val_accuracy: 0.6306 - val_loss: 0.6163
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 194s 2s/step - accuracy: 0.6558 - loss: 0.6082 - val_accuracy: 0.6981 - val_loss: 0.5822
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 183s 2s/step - accuracy: 0.6960 - loss: 0.5854 - val_accuracy: 0.7019 - val_loss: 0.5723
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 169s 2s/step - accuracy: 0.6856 - loss: 0.5836 - val_accuracy: 0.7344 - val_loss: 0.5387
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 157s 2s/step - accuracy: 0.7250 - loss: 0.5514 - val_accuracy: 0.7600 - val_loss: 0.4999
Epoch 7/10
 25/100 ━━━━━━━━━━━━━━━━━━━━ 1:31 1s/step - accuracy: 0.7278 - loss: 0.5156
C:\Users\micro\anaconda3\Lib\site-packages\keras\src\trainers\epoch_iterator.py:107: UserWarning: Your input ran out of data; int
sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()`
your dataset.
  self._interrupted_warning()
100/100 ━━━━━━━━━━━━━━━━━━━━ 47s 466ms/step - accuracy: 0.7332 - loss: 0.5165 - val_accuracy: 0.7600 - val_loss: 0.4977
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 144s 1s/step - accuracy: 0.7723 - loss: 0.4882 - val_accuracy: 0.7750 - val_loss: 0.4786
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.7548 - loss: 0.4966 - val_accuracy: 0.7531 - val_loss: 0.5040
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.7744 - loss: 0.4635 - val_accuracy: 0.7812 - val_loss: 0.4714
```

Vẽ biểu đồ để đánh giá mức độ hiệu quả của mô hình:

```
[7]: import matplotlib.pyplot as plt

# Vẽ Accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy theo Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Vẽ Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss theo Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Kết quả :



- **Accuracy tăng đều** qua các epoch:

  Bắt đầu từ ~50% (Epoch 1) → đến ~77% (Epoch 10).
- **Loss (hàm mất mát) giảm dần**, chứng tỏ mô hình học được quy luật phân loại ảnh.
- **Train accuracy ≈ Val accuracy** → mô hình tổng quát tốt.
- **Val loss giảm ổn định** đến khoảng Epoch 8 → sau đó hơi dao động nhẹ (Epoch 9/10 val_loss tăng lên 1 chút), **nhưng không đáng kể**.

  Điều này cho thấy mô hình chưa bị **quá khớp (overfitting)**

Ở 2 lần thử trên ta có batch_size =32 và step per epoch = 100 nên ở mỗi epoch sẽ có 3200 ảnh được đưa vào huấn luyện

6.3,Ta thử với epoch =5 và step per epoch = 625 để mỗi epoch sẽ đưa tất cả ảnh vào huấn luyện:

```
[20]: # Biên dịch mô hình
      model.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

      # Huấn luyện mô hình
      history = model.fit(
          train_generator,            # Bộ sinh dữ liệu train
          steps_per_epoch=625,        # Số bước mỗi epoch (100 bước mỗi epoch)
          epochs=5,                  # Số lần huấn luyện qua toàn bộ dữ liệu
          validation_data=val_generator,  # Bộ sinh dữ liệu validation
          validation_steps=156         # Số bước mỗi epoch cho validation
      )

      # Lưu mô hình sau khi huấn luyện
      model.save('cats_and_dogs_model.h5')

      print("✅ Huấn luyện hoàn tất!")
```

Kết quả:

```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 885s 1s/step - accuracy: 0.5290 - loss: 0.7564 - val_accuracy: 0.5988 - val_loss: 0.6664
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 815s 1s/step - accuracy: 0.6669 - loss: 0.6090 - val_accuracy: 0.7570 - val_loss: 0.5011
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 814s 1s/step - accuracy: 0.7701 - loss: 0.4809 - val_accuracy: 0.8023 - val_loss: 0.4269
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 811s 1s/step - accuracy: 0.8421 - loss: 0.3598 - val_accuracy: 0.8181 - val_loss: 0.4030
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 789s 1s/step - accuracy: 0.9009 - loss: 0.2333 - val_accuracy: 0.8221 - val_loss: 0.4218
```

- Mô hình này đang **học tốt**, độ chính xác trên tập **validation khoảng 82% là hợp lý**.
- Mặc dù accuracy train lên tới 90%, nhưng val_accuracy vẫn ổn định, chứng tỏ **mô hình chưa bị overfitting nghiêm trọng**.

Hoàn tất huấn luyện, dùng tập test với 12500 ảnh vừa chó vừa mèo lẫn lộn và chưa được dán nhãn. Thử dự đoán với 30 ảnh đầu xem mô hình có hiệu quả không

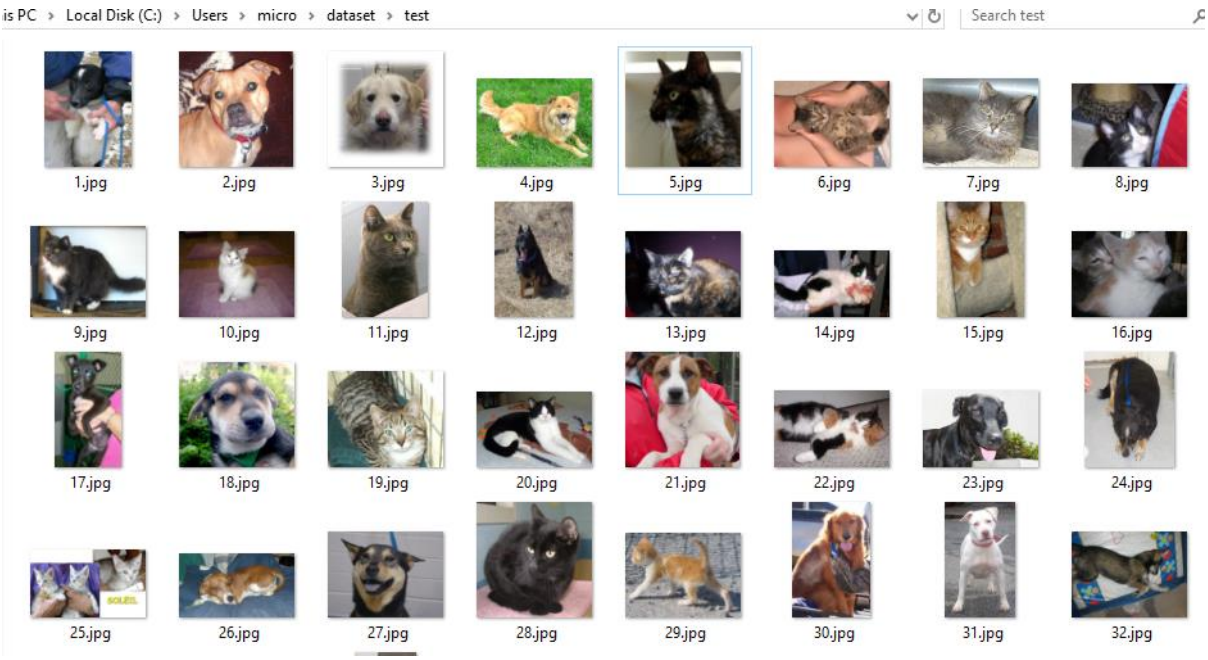
[18]:

```python
import os
from tensorflow.keras.preprocessing import image
import numpy as np
from tensorflow.keras.models import load_model

# Đường dẫn đến thư mục chứa ảnh test
test_folder = 'dataset/test'

# Tải mô hình đã huấn luyện
model = load_model('cats_and_dogs_model.h5')

# Dự đoán cho 30 ảnh đầu tiên
for i in range(1, 31):
    img_path = os.path.join(test_folder, f"{i}.jpg")

    # Tải và tiền xử lý ảnh
    img = image.load_img(img_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    # Dự đoán
    prediction = model.predict(img_array, verbose=0)
    label = 'Chó 🐶' if prediction[0][0] > 0.5 else 'Mèo 🐱'

    print(f"Ảnh {i}.jpg → {label}")
```

Và đây là kết quả do mô hình dự đoán

```
Ảnh 1.jpg → Chó 🐶
Ảnh 2.jpg → Chó 🐶
Ảnh 3.jpg → Chó 🐶
Ảnh 4.jpg → Chó 🐶
Ảnh 5.jpg → Mèo 🐱
Ảnh 6.jpg → Mèo 🐱
Ảnh 7.jpg → Mèo 🐱
Ảnh 8.jpg → Chó 🐶
Ảnh 9.jpg → Mèo 🐱
Ảnh 10.jpg → Mèo 🐱
Ảnh 11.jpg → Mèo 🐱
Ảnh 12.jpg → Chó 🐶
Ảnh 13.jpg → Mèo 🐱
Ảnh 14.jpg → Mèo 🐱
Ảnh 15.jpg → Mèo 🐱
Ảnh 16.jpg → Mèo 🐱
Ảnh 17.jpg → Chó 🐶
Ảnh 18.jpg → Chó 🐶
Ảnh 19.jpg → Mèo 🐱
Ảnh 20.jpg → Mèo 🐱
Ảnh 21.jpg → Chó 🐶
Ảnh 22.jpg → Mèo 🐱
Ảnh 23.jpg → Mèo 🐱
Ảnh 24.jpg → Chó 🐶
Ảnh 25.jpg → Chó 🐶
Ảnh 26.jpg → Chó 🐶
Ảnh 27.jpg → Chó 🐶
Ảnh 28.jpg → Mèo 🐱
Ảnh 29.jpg → Chó 🐶
Ảnh 30.jpg → Chó 🐶
```

Kiểm tra độ chính xác :



Mô hình dự đoán đúng khoảng 25/30 ảnh tức tỉ lệ khoảng 83% - Khá tốt

KẾT LUẬN

Báo cáo cung cấp cái nhìn tổng quan về trí tuệ nhân tạo (AI), học máy (ML), và học sâu (DL), với trọng tâm là ứng dụng phân loại ảnh chó/mèo bằng mạng nơ-ron tích chập (CNN).

Chương 1: giới thiệu các khái niệm cốt lõi của AI và ML, bao gồm học có giám sát, không giám sát, và học tăng cường, cùng các thư viện Python như NumPy, TensorFlow, và PyTorch. Các ứng dụng tại Việt Nam, như công nghệ nhận diện của VinAI, minh họa tiềm năng AI trong bối cảnh địa phương.

Chương 2: trình bày các kỹ thuật học sâu, bao gồm kiến trúc mạng nơ-ron (CNN, RNN, Transformer) và nền tảng toán học (tensor, gradient descent, backpropagation). Các ví dụ mã nguồn làm rõ cách xây dựng và tối ưu hóa mô hình.

Chương 3 : phân tích ứng dụng CNN trong phân loại chó/mèo, đạt độ chính xác 82% trên tập

TƯỜNG TRÌNH CHỈNH SỬA

10 điểm cần chỉnh sửa:

1. **Thêm sự phân biệt giữa Trí tuệ nhân tạo (AI) - Machine Learning (ML) - Deep Learning (DL)**

- Vấn đề: Phần giới thiệu về trí tuệ nhân tạo (trang 3) chỉ mô tả chung chung rằng AI "tập trung vào việc xây dựng các hệ thống có khả năng thực hiện nhiệm vụ thông minh". Điều này thiếu chi tiết và không phân biệt rõ ràng giữa AI, Machine Learning (ML), và Deep Learning (DL)
- Chỉnh sửa:

+ Trí tuệ nhân tạo (AI) là lĩnh vực nghiên cứu phát triển các hệ thống có thể mô phỏng hành vi trí tuệ của con người. Trong AI có một nhánh quan trọng là **Machine Learning (ML)**, cho phép máy tính học từ dữ liệu mà không cần lập trình cụ thể.

+ Một nhánh sâu hơn nữa của ML là **Deep Learning (DL)** – sử dụng các mạng nơ-ron nhân tạo với nhiều lớp ẩn (deep neural networks) để xử lý dữ liệu phức tạp như hình ảnh, âm thanh, văn bản.

+ Minh họa mối quan hệ:

```
Trí tuệ nhân tạo (AI)
 └── Học máy (Machine Learning)
      └── Học sâu (Deep Learning)
```

2. Mở rộng phần ứng dụng thực tế của AI

- Vấn đề: Phần ứng dụng thực tế (trang 10-11) liệt kê các lĩnh vực như y tế, tài chính, thương mại điện tử, nhưng chưa có chiều sâu, chưa cụ thể
- Chỉnh sửa:

+ **Y tế:** AI hỗ trợ bác sĩ trong việc phân tích ảnh y khoa, ví dụ như hệ thống của Google DeepMind đạt độ chính xác vượt cả chuyên gia khi chẩn đoán bệnh võng mạc tiểu đường.

+ **Tài chính:** Các hệ thống phát hiện gian lận thẻ tín dụng theo thời gian thực sử dụng machine learning để phát hiện hành vi bất thường.

+ **Giáo dục:** Nền tảng như Duolingo ứng dụng AI để điều chỉnh bài học phù hợp với năng lực học tập của từng người học.

VinAI đã phát triển công nghệ nhận diện khuôn mặt và xử lý ngôn ngữ tự nhiên, được ứng dụng trong các sản phẩm của Vingroup."

Bổ sung số liệu: "Theo báo cáo của McKinsey (2023), AI có thể đóng góp thêm 13 nghìn tỷ USD vào nền kinh tế toàn cầu vào năm 2030."

3. bổ sung chi tiết về các thư viện Python

- Vấn đề: Phần về thư viện Python (trang 5) mô tả Numpy, Pandas, Matplotlib, Seaborn, và Scikit-learn, nhưng thiếu thông tin về các thư viện quan trọng khác như TensorFlow hoặc PyTorch
- Chỉnh sửa: Bổ sung 2 thư viện TensorFlow và PyTorch

+ TensorFlow là thư viện mạnh mẽ cho học sâu, hỗ trợ xây dựng và huấn luyện mạng nơ-ron. Ví dụ:

```
import tensorflow as tf
model = tf.keras.Sequential([
tf.keras.layers.Dense(10, activation='relu', input_shape=(4,)),
                tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy')
```

+ PyTorch được ưa chuộng trong nghiên cứu AI nhờ tính linh hoạt. Ví dụ:

```
import torch
import torch.nn as nn
class Net(nn.Module):
def __init__(self):
super(Net, self).__init__()
self.fc = nn.Linear(4, 1)
self.sigmoid = nn.Sigmoid()
def forward(self, x):
return self.sigmoid(self.fc(x))
    model = Net()
```

4. Bổ sung phần về học tăng cường (Reinforcement Learning)

- Vấn đề: Chương 1 chỉ đề cập đến học có giám sát và không giám sát (trang 4, 7), nhưng bỏ qua học tăng cường, một nhánh quan trọng của ML.

- Chỉnh sửa: thêm 1 mục

Học tăng cường (Reinforcement Learning- RL) là một nhánh của học máy, trong đó một tác nhân (agent) học cách đưa ra quyết định bằng cách thử và sai trong một môi trường (environment). Mục tiêu của tác nhân là tối ưu hóa phần thưởng tích lũy (cumulative reward) thông qua việc chọn các hành động

(actions) dựa trên trạng thái (states) của môi trường và một chính sách (policy).

1.1 Các khái niệm cốt lõi

• Tác nhân (Agent): Thực thể đưa ra quyết định, ví dụ: một AI chơi cờ vua.

• Môi trường (Environment): Hệ thống mà tác nhân tương tác, ví dụ: bàn cờ hoặc thế giới vật

lý.

• Trạng thái (State): Mô tả tình trạng hiện tại của môi trường, ví dụ: vị trí của các quân cờ.

• Hành động (Action): Lựa chọn của tác nhân, ví dụ: di chuyển một quân cờ.

• Phần thưởng (Reward): Phản hồi từ môi trường, ví dụ: +1 nếu thắng,-1 nếu thua.

• Chính sách (Policy): Chiến lược của tác nhân để chọn hành động dựa trên trạng thái, thường biểu diễn bằng $\pi(a|s)$.

Quy trình RL được minh họa bởi vòng lặp sau: tác nhân quan sát trạng thái st, chọn hành động at, nhận phần thưởng rt và trạng thái tiếp theo st+1, sau đó cập nhật chính sách để cải thiện quyết định.

Sơ đồ minh họa:

Tác nhân ↔ Môi trường : (st,at,rt,st+1)

1.2 Các thuật toán phổ biến

• Q-Learning: Thuật toán dựa trên giá trị, cập nhật hàm giá trị hành động Q(s,a) theo công thức:

(

$Q(st,at) \leftarrow Q(st,at) + \alpha$

$rt + \gamma \max_a$

$Q(st+1,a) - Q(st,at)$

Trong đó: α là tốc độ học, γ là hệ số chiết khấu.

)

• SARSA: Tương tự Q-Learning nhưng cập nhật dựa trên hành động thực sự được chọn.

• Deep Q-Networks (DQN): Kết hợp Q-Learning với mạng nơ-ron sâu để xử lý các trạng thái

phức tạp, như hình ảnh.

• Policy Gradient: Tối ưu hóa trực tiếp chính sách π(a|s) bằng gradient descent.

• Proximal Policy Optimization (PPO): Thuật toán hiện đại, cân bằng giữa hiệu quả và ổn

định.

1.3 Ví dụ minh họa: Q-Learning

Dưới đây là ví dụ về Q-Learning trong bài toán tác nhân di chuyển trong lưới 5x5 để tìm kho báu (phần

thưởng +100) và tránh chướng ngại vật (phần thưởng-1)

```
import numpy as np
R = np.array([
 [-1,-1,-1,-1, 0],
 [-1,-10,-1,-1, 0],
 [-1,-1,-10,-1, 0],
 [-1,-1,-1,-1, 0],
 [0, 0, 0, 0, 100]
 ])
Q = np.zeros((5, 5))
 1
alpha, gamma, episodes = 0.1, 0.9, 1000
for _ in range(episodes):
 state = np.random.randint(0, 5)
 while state != 4
 action = np.random.randint(0, 5)
 next_state = action
 Q[state, action] += alpha * (R[state, action] + gamma * np.max(Q[next_state
 ])- Q[state, action])
 state = next_state
```

```
print(Q)
```

Giải thích:

• Ma trận R định nghĩa phần thưởng: +100 tại kho báu,-10 tại chướng ngại vật, 0 khi đến trạng

thái tiếp theo,-1 cho các bước di chuyển khác.

• Matrận Q lưu trữ giá trị hành động, được cập nhật qua mỗi tập (episode).

• Tác nhân học cách chọn hành động để tối ưu hóa phần thưởng tích lũy.

1.4 Ứng dụng thực tế

Học tăng cường được ứng dụng rộng rãi trong nhiều lĩnh vực:

• Trò chơi: AlphaZero của DeepMind sử dụng RL để đạt hiệu suất siêu việt trong cờ vua và cờ vây.

• Robot: Điều khiển robot di chuyển trong môi trường không xác định, như robot hút bụi thông minh.

• Quảng cáo: Tối ưu hóa chiến lược đặt giá thầu (bidding) trong quảng cáo trực tuyến.

• Giao thông thông minh: Tại Việt Nam, RL có thể được áp dụng để tối ưu hóa đèn giao thông, giảm ùn tắc (ví dụ: nghiên cứu tại Đại học Bách Khoa TP.HCM).

• Nông nghiệp: Tối ưu hóa lịch tưới tiêu hoặc phân bón dựa trên dữ liệu thời tiết và đất đai

5. Bổ sung phần về kiến trúc mạng nơ-ron phổ biến

- Vấn đề: Phần giới thiệu về học sâu (trang 12-13) không đề cập đến các kiến trúc mạng nơ-ron phổ biến như CNN, RNN, hoặc Transformer, ngoài ví dụ đơn giản về phân loại chó/mèo
- Chỉnh sửa

1 Các kiến trúc mạng nơ-ron phổ biến

Mạng nơ-ron sâu (Deep Neural Networks) là nền tảng của học sâu, với các kiến trúc khác nhau được

thiết kế để giải quyết các bài toán cụ thể. Phần này giới thiệu ba kiến trúc phổ biến: Convolutional

Neural Networks (CNN), Recurrent Neural Networks (RNN), và Transformer, cùng với các

ví dụ mã nguồn và ứng dụng thực tế

1.2 Chi tiết từng kiến trúc

1.2.1 Convolutional Neural Networks (CNN)

CNN được thiết kế để xử lý dữ liệu có cấu trúc không gian, như hình ảnh. CNN sử dụng các lớp convolution (Conv2D) để trích xuất đặc trưng (như cạnh, góc) và các lớp pooling (MaxPooling2D) để giảm kích thước dữ liệu.

Cấu trúc:

$$\text{Input} \to [\text{Conv2D} \to \text{ReLU} \to \text{Pooling}] \times N \to \text{Flatten} \to \text{Dense} \to \text{Output}$$

Công thức convolution:

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Trong đó: I là ảnh đầu vào, K là kernel, s(i,j) là giá trị tại vị trí (i,j).

Ví dụ mã nguồn:

```
from tensorflow.keras import layers, models
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Ứng dụng:

- Phân loại hình ảnh (như chó/mèo trong Chương 3).
- Nhận diện biển số xe tại Việt Nam (ví dụ: hệ thống giám sát giao thông của FPT).
- Chẩn đoán y tế qua ảnh X-quang hoặc MRI.

Biến thể: LeNet (1998), VGG16, ResNet

1.2.2 Recurrent Neural Networks (RNN)

RNN được thiết kế để xử lý dữ liệu tuần tự, như chuỗi thời gian hoặc văn bản. RNN duy trì một trạng

thái ẩn (hidden state) để ghi nhớ thông tin từ các bước trước.

Cấu trúc

$$\text{Input} \rightarrow [\text{RNN Cell}] \times T \rightarrow \text{Dense} \rightarrow \text{Output}$$

Công thức RNN:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$$

Trong đó: ht là trạng thái ẩn tại thời điểm t, xt là đầu vào, Wh, Wx là trọng số.

Ví dụ mã nguồn:

```
from tensorflow.keras import layers, models
model = models.Sequential([
layers.LSTM(64, input_shape=(10, 1)), % 10 time steps, 1 feature
layers.Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mse')
```

Ứng dụng:

- Dựđoán giá cổ phiếu dựa trên dữ liệu lịch sử.
- Dịch máy (ví dụ: dịch tiếng Việt sang tiếng Anh).
- Phân tích cảm xúc văn bản (sentiment analysis) trong các ứng dụng như Zalo.

Biến thể: LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit).

6. **bổ sung chi tiết về toán học trong học sâu**

- Vấn đề: Phần “Các nền tảng toán học của học sâu” (trang 13-14) chỉ đề cập sơ lược đến tensor và phép toán đại số tuyến tính, nhưng thiếu công thức cụ thể và giải thích
- Chỉnh sửa:

Học sâu (Deep Learning) dựa trên các khái niệm toán học như tensor, đại số tuyến tính, gradient descent, backpropagation, và hàm kích hoạt. Phần này sửa các lỗi trong công thức hiện có, bổ sung giải thích chi tiết, và cung cấp ví dụ minh họa

1.2 Chi tiết các khái niệm

1.2.1 Tensor

Tensor là mảng đa chiều, tổng quát hóa vector (bậc 1) và ma trận (bậc 2). Trong học sâu, tensor lưu

trữ dữ liệu đầu vào (như ảnh), trọng số, và đầu ra. Ví dụ: một ảnh màu 150x150 có 3 kênh (RGB) là

tensor bậc 3 với kích thước [150,150,3].

1.2.2 Đại số tuyến tính

Các phép toán đại số tuyến tính, như nhân ma trận và convolution, là cốt lõi của học sâu. Nhân ma

trận được dùng trong các lớp fully-connected, còn convolution được dùng trong CNN.

Công thức nhân ma trận:

$$C = A \cdot B, \quad c_{ij} = \sum_k a_{ik} b_{kj}$$

Công thức convolution

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

1.2.3 Gradient Descent

Gradient Descent tối ưu hóa hàm mất mát L bằng cách cập nhật trọng số w theo hướng giảm gradient:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

Trong đó: η là tốc độ học (learning rate).

1.2.4 Backpropagation

Backpropagation lan truyền lỗi ngược qua mạng để tính gradient của hàm mất mát đối với mỗi trọng

số, sử dụng quy tắc chuỗi (chain rule):

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

1.2.5 Hàm kích hoạt

Hàm kích hoạt thêm tính phi tuyến vào mạng nơ-ron, giúp mô hình học các mẫu phức tạp. Các hàm

phổ biến

- **ReLU**: $f(x) = \max(0, x)$
- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
- **Softmax**: $f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ (dùng cho phân loại đa lớp)

7. **Sửa lỗi trong phần tiền xử lý dữ liệu ảnh**

- Vấn đề : Phần tải và chuẩn bị dữ liệu (trang 32) không đề cập rõ ràng đến việc chuẩn hóa ảnh hoặc tăng cường dữ liệu (data augmentation)
- Chỉnh sửa:

Chia rõ phần code:

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

train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  horizontal_flip=True
)
train_generator = train_datagen.flow_from_directory(
  'dataset/train',
  target_size=(150, 150),
  batch_size=32,
  class_mode='binary'
)
```

8. **Bổ sung đánh giá mô hình bằng các chỉ số bổ sung**

- Vấn đề : Phần đánh giá mô hình (trang 38-40) chỉ sử dụng accuracy và loss, thiếu các chỉ số như precision, recall, và ma trận nhầm lẫn.
- Chỉnh sửa: Thêm mã nguồn tính các chỉ số:

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

y_pred = model.predict(test_generator)
y_pred_classes = (y_pred > 0.5).astype(int)
print(classification_report(test_generator.classes, y_pred_classes, target_names=['Mèo', 'Chó']))
sns.heatmap(confusion_matrix(test_generator.classes, y_pred_classes), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Dự đoán')
plt.ylabel('Thực tế')
plt.show()
```

9. **Bổ sung phần tối ưu hóa mô hình CNN**
   - Vấn đề: Phần huấn luyện mô hình (trang 38) không đề cập đến các kỹ thuật tối ưu hóa như early stopping hoặc learning rate scheduling, dẫn đến hiệu suất chưa tối ưu.
   - Chỉnh sửa:

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

early_stopping = EarlyStopping(monitor='val_loss', patience=3)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_generator, epochs=10, validation_data=val_generator,
          callbacks=[early_stopping, lr_scheduler])
```

Giải thích: “Early stopping dừng huấn luyện khi validation loss không cải thiện, còn learning rate scheduling giảm tốc độ học để đạt tối ưu tốt hơn.

10. Thiếu kết luận

Bổ sung:

Báo cáo cung cấp cái nhìn tổng quan về trí tuệ nhân tạo (AI), học máy (ML), và học sâu (DL), với trọng tâm là ứng dụng phân loại ảnh chó/mèo bằng mạng nơ-ron tích chập (CNN).

- Chương 1 giới thiệu các khái niệm cốt lõi của AI và ML, bao gồm học có giám sát, không giám sát, và học tăng cường, cùng các thư viện Python như NumPy, TensorFlow, và PyTorch. Các ứng dụng tại Việt Nam, như công nghệ nhận diện của VinAI, minh họa tiềm năng AI trong bối cảnh địa phương.
- Chương 2trình bày các kỹ thuật học sâu, bao gồm kiến trúc mạng nơ-ron (CNN, RNN, Transformer) và nền tảng toán học (tensor, gradient descent, backpropagation). Các ví dụ mã nguồn làm rõ cách xây dựng và tối ưu hóa mô hình.
- Chương 3 phân tích ứng dụng CNN trong phân loại chó/mèo, đạt độ chính xác 82% trên tập validation. Kết quả cho thấy hiệu quả của DL nhưng cũng chỉ ra hạn chế