

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	: PGS.TS. Trần Đình Quế
Họ và tên sinh viên	: Nguyễn Xuân Việt
Mã sinh viên	: B22DCCN898
Lớp	: D22CQCN10

Hà Nội – 2025

LỜI MỞ ĐẦU.....	6
CHƯƠNG 1: TRÍ TUỆ NHÂN TẠO VÀ CÁC ỨNG DỤNG.....	7
1.1. Giới thiệu	7
1.2. Lịch sử hình thành và phát triển của AI.....	7
1.3. Các khái niệm cơ bản	9
1.3.1. Các loại AI:.....	9
1.3.2. Học máy (Machine Learning - ML):	9
1.3.3. Học sâu (Deep Learning - DL):.....	10
1.3.4. Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP):.....	10
1.3.5. Thị giác máy tính (Computer Vision):	11
1.3.6. Robotics và Hệ thống tự hành:	11
1.4. Ứng dụng của trí tuệ nhân tạo	11
1.4.1. Y tế và Chăm sóc sức khỏe:	11
1.4.2. Kinh doanh, Tài chính và Tiếp thị:	11
1.4.3. Giao thông vận tải và Logistics:	12
1.4.4. Sản xuất và Công nghiệp:	12
1.4.5. Giáo dục và Đào tạo:	12
1.4.6. Giải trí và Truyền thông:	13
1.4.7. Nông nghiệp thông minh (Precision Agriculture):	13
1.4.8. Khoa học và Nghiên cứu:.....	13
1.4.9. An ninh và Quốc phòng:	13
1.4.10. Đời sống hàng ngày và Nhà thông minh:.....	14
1.5. Thách thức và Rủi ro đi kèm với AI	14
1.5.1. Thách thức về kỹ thuật và dữ liệu:	14
1.5.2. Các vấn đề đạo đức và xã hội:.....	15
1.5.3. Nhu cầu về quản lý và pháp lý:	15
1.6. Tương lai của Trí tuệ nhân tạo	15
1.6.1. Xu hướng phát triển công nghệ:	15
1.6.2. Hướng tới các dạng trí tuệ nhân tạo cao hơn:	16
1.6.3. Tích hợp AI sâu rộng vào xã hội:	16
1.6.4. Vai trò của đạo đức và quản trị AI:	16
1.6.5. AI và việc giải quyết các thách thức toàn cầu:	16
CHƯƠNG 2: CÁC KỸ THUẬT HỌC SÂU	17

2.1. Nền tảng Cốt lõi và Cơ chế Huấn luyện	17
2.1.1. Các Thành phần Cơ bản	17
2.1.1.1. Các Loại Lớp (Layer Types).....	17
2.1.1.2. Hàm Kích hoạt (Activation Functions)	18
2.1.2. Cơ chế Huấn luyện Cơ bản	18
2.1.2.1. Hàm Mất mát (Loss Functions)	18
2.1.2.2. Tối ưu hóa dựa trên Gradient.....	19
2.1.2.3. Backpropagation (Tính toán Gradient)	19
2.1.2.4. Stochastic Gradient Descent và các Thuật toán Tối ưu hóa	20
2.2. Xây dựng và Đánh giá Mô hình Cơ bản	21
2.2.1. Mô hình Kết nối Dày đặc (Densely Connected Networks)	21
2.2.2. Quy trình Đánh giá Mô hình	21
2.2.2.1. Chia Tập Dữ liệu (Train/Validation/Test Split).....	21
2.2.2.2. K-Fold Cross-Validation.....	22
2.2.2.3. Số liệu Đánh giá (Metrics)	22
2.3. Xử lý Dữ liệu và Kỹ thuật Điều chuẩn Phổ biến.....	23
2.3.1. Xử lý Dữ liệu.....	23
2.3.1.1. Vector hóa (Vectorization)	23
2.3.1.2. Chuẩn hóa Dữ liệu (Data Normalization)	24
2.3.1.3. Padding và Masking.....	24
2.3.2. Các Kỹ thuật Điều chuẩn Cơ bản	25
2.3.2.1. Giảm Kích thước Mạng.....	25
2.3.2.2. Dropout.....	25
2.3.2.3. Weight Regularization.....	26
2.3.2.4. Early Stopping	26
2.3.2.5. Data Augmentation	27
2.4. Các Kiến trúc Nâng cao cho Dữ liệu Có cấu trúc	28
2.4.1. Mạng Nơ-ron Tích chập (Convolutional Neural Networks - CNNs)	28
2.4.1.1. Các Lớp Chính.....	28
2.4.1.2. Ứng dụng của CNN.....	29
2.4.2. Mạng Nơ-ron Hồi quy (Recurrent Neural Networks - RNNs).....	29
2.4.2.1. Các Lớp Chính.....	29
2.4.2.2. Xếp chồng RNN (Stacking RNNs)	29

2.4.2.3. RNN Hai chiều (Bidirectional RNNs)	30
2.4.2.4. Recurrent Dropout	30
2.4.2.5. Ứng dụng của RNN.....	30
2.5. Các Kỹ thuật Kiến trúc và Chuẩn hóa Nâng cao.....	31
2.5.1. Kết nối Thặng dư (Residual Connections)	31
2.5.2. Chuẩn hóa Lô (Batch Normalization).....	31
2.5.3. Chuẩn hóa Lớp (Layer Normalization).....	32
2.5.4. Tích chập Tách biệt theo Chiều sâu (Depthwise Separable Convolutions)	33
2.6. Kỹ thuật Chuyên sâu cho Xử lý Ngôn ngữ Tự nhiên	33
2.6.1. Word Embeddings.....	33
2.6.2. Attention và Self-Attention	34
2.6.3. Kiến trúc Transformer	35
2.6.4. Học Sequence-to-Sequence.....	36
CHƯƠNG 3: ỨNG DỤNG HỌC SÂU TRONG PHÂN LOẠI HÌNH ẢNH	38
3.1. Huấn luyện mô hình	38
3.1.1. Mô hình phân loại chim (Fine-tuning ResNet50).....	38
3.1.1.1. Mục tiêu bài toán	38
3.1.1.2. Tập dữ liệu.....	38
3.1.1.3. Thực hiện tiền xử lý và chia tập huấn luyện/kiểm tra.....	38
3.1.1.4. Xây dựng mô hình CNN (Fine-tuning ResNet50).....	40
3.1.1.5. Huấn luyện mô hình	41
3.1.1.6. Đánh giá và thử nghiệm mô hình.....	43
3.1.2. Mô hình phân biệt cảm xúc	44
3.1.2.1. Mục tiêu bài toán	44
3.1.2.2. Tập dữ liệu.....	44
3.1.2.3. Thực hiện tiền xử lý và chia tập huấn luyện kiểm tra.....	44
3.1.2.4. Xây dựng mô hình CNN	45
3.1.2.5. Huấn luyện mô hình	47
3.1.2.6. Đánh giá và thử nghiệm mô hình.....	49
3.1.3. Mô hình phân loại rác (Fine-tuning VGG16).....	50
3.1.3.1. Mục tiêu bài toán	50
3.1.3.2. Tập dữ liệu.....	50
3.1.3.3. Thực hiện tiền xử lý và chia tập huấn luyện/kiểm tra.....	51

3.1.3.4. Xây dựng mô hình CNN (Fine-tuning VGG16).....	52
3.1.3.5. Huấn luyện mô hình.....	54
3.1.3.6. Đánh giá và thử nghiệm mô hình.....	56
3.2. Triển khai ứng dụng.....	58
3.2.1. Giới thiệu.....	58
3.2.2. Kiến trúc hệ thống	58
3.2.3. Các mô hình phân loại	58
3.2.3.1 Mô hình phân loại loài chim (BirdClassifier).....	58
3.2.3.2 Mô hình phân loại rác thải (TrashClassifier).....	59
3.2.3.3 Mô hình phân loại cảm xúc (EmotionClassifier).....	59
3.2.4. Backend (API)	59
3.2.5. Frontend (Web Application)	60
BẢN TƯỜNG TRÌNH NHỮNG ĐIỂM ĐÃ CHỈNH SỬA.....	61
KẾT LUẬN	67

LỜI MỞ ĐẦU

Trí tuệ nhân tạo (AI), đặc biệt là lĩnh vực học sâu (Deep Learning), đã và đang tạo ra những bước tiến vượt bậc, mang lại những thay đổi sâu sắc trong nhiều khía cạnh của khoa học, công nghệ và đời sống xã hội. Khả năng của máy tính trong việc "học" từ dữ liệu và thực hiện các tác vụ phức tạp như nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, hay ra quyết định đã mở ra vô số ứng dụng tiềm năng. Trong bối cảnh đó, việc tìm hiểu và ứng dụng các kỹ thuật học sâu vào giải quyết các bài toán thực tiễn trở thành một yêu cầu quan trọng đối với sinh viên ngành Công nghệ Thông tin.

Bài tiểu luận này được thực hiện trong khuôn khổ học phần Thực tập Cơ sở, nhằm mục tiêu tổng hợp kiến thức nền tảng về Trí tuệ nhân tạo và đi sâu nghiên cứu các kỹ thuật học sâu phổ biến. Đồng thời, tiểu luận trình bày quá trình thực nghiệm xây dựng và đánh giá một số mô hình học sâu cụ thể cho bài toán phân loại hình ảnh – một trong những ứng dụng tiêu biểu và có tác động rộng rãi của học sâu.

Nội dung của bài tiểu luận gồm ba chương chính. **Chương 1** sẽ giới thiệu tổng quan về Trí tuệ nhân tạo, bao gồm lịch sử hình thành và phát triển, các khái niệm cơ bản, những ứng dụng đa dạng trong thực tiễn, cũng như các thách thức và tiềm năng phát triển trong tương lai. **Chương 2** tập trung đi sâu vào các kỹ thuật học sâu, từ những nền tảng cốt lõi, cơ chế huấn luyện, các kiến trúc mạng nơ-ron phổ biến như Mạng Nơ-ron Tích chập (CNN) và Mạng Nơ-ron Hồi quy (RNN), đến các kỹ thuật kiến trúc và chuẩn hóa nâng cao, cũng như các kỹ thuật chuyên sâu cho xử lý ngôn ngữ tự nhiên. **Chương 3** là phần trọng tâm thực nghiệm, trình bày chi tiết quá trình xây dựng, huấn luyện và đánh giá ba mô hình học sâu ứng dụng trong phân loại hình ảnh: mô hình phân loại các loài chim (sử dụng fine-tuning ResNet50), mô hình phân biệt cảm xúc khuôn mặt (sử dụng CNN tùy chỉnh), và mô hình phân loại rác thải (sử dụng fine-tuning VGG16). Cuối cùng, chương này cũng mô tả quá trình triển khai một ứng dụng web đơn giản để minh họa khả năng ứng dụng của các mô hình đã huấn luyện.

CHƯƠNG 1: TRÍ TUỆ NHÂN TẠO VÀ CÁC ỨNG DỤNG

1.1. Giới thiệu

Trong những năm gần đây trí tuệ nhân tạo (Artificial Intelligence - AI) đã trở thành một trong những công nghệ đột phá lớn thay đổi cách thế giới vận hành. AI xuất hiện trong mọi ngóc ngách của đời sống hiện đại, từ hệ thống gợi ý sản phẩm, cho đến những robot tự hành và trợ lý ảo thông minh. Nó không còn là khái niệm trong các bộ phim khoa học viễn tưởng mà đã trở thành một động lực cốt lõi thúc đẩy sự đổi mới và tiến bộ trên toàn cầu.

Định nghĩa Trí tuệ nhân tạo (AI): Trí tuệ nhân tạo là lĩnh vực khoa học máy tính tập trung vào việc tạo ra các hệ thống máy móc có khả năng thực hiện những nhiệm vụ thường đòi hỏi trí thông minh của con người. Những khả năng này bao gồm học hỏi (learning), lập luận (reasoning), giải quyết vấn đề (problem-solving), nhận thức (perception), hiểu ngôn ngữ (language understanding) và đưa ra quyết định (decision-making). Mục tiêu của AI không nhất thiết là tạo ra một bản sao hoàn hảo của bộ não con người, mà là xây dựng các công cụ thông minh có thể hỗ trợ và nâng cao năng lực của chúng ta.

Sự phát triển vượt bậc của AI trong những năm gần đây được thúc đẩy bởi **ba** yếu tố chính:

- **Dữ liệu lớn (Big Data):** Sự bùng nổ của internet và các thiết bị kết nối trong thời gian giãn cách covid-19 đã tạo ra một lượng dữ liệu khổng lồ, cung cấp "nguyên liệu" cần thiết để huấn luyện các mô hình AI phức tạp.
- **Sức mạnh tính toán:** Sự tiến bộ trong công nghệ phần cứng, đặc biệt là các bộ xử lý đồ họa (GPU) và các chip chuyên dụng cho AI, đã cho phép xử lý các thuật toán AI đòi hỏi tài nguyên lớn một cách hiệu quả.
- **Thuật toán tiên tiến:** Các thuật toán ngày càng mạnh mẽ, đặc biệt trong lĩnh vực Học máy (Machine Learning) và Học sâu (Deep Learning), giúp máy tính học hỏi từ dữ liệu với độ chính xác ngày càng cao.

1.2. Lịch sử hình thành và phát triển của AI

Những ý tưởng sơ khai và nền tảng lý thuyết: Mặc dù thuật ngữ "Trí tuệ nhân tạo" chỉ mới xuất hiện vào giữa thế kỷ 20, nhưng khát vọng tạo ra những cỗ máy thông minh đã có từ lâu trong lịch sử loài người, thể hiện qua các huyền thoại và tác phẩm văn học. Về mặt lý thuyết, nền tảng cho AI hiện đại được đặt bởi các nhà toán học và logic học như George Boole (đại số Boole), Gottlob Frege (logic vị từ), và đặc biệt là Alan Turing.

Phép thử Turing (Turing Test): Vào năm 1950, Alan Turing đã đề xuất một phép thử nổi tiếng để đánh giá xem một cỗ máy có thể thể hiện hành vi thông minh tương đương hoặc không thể phân biệt được với con người hay không. Phép thử này đã đặt ra một câu hỏi nền tảng và định hướng cho nghiên cứu AI sau này.

Lý thuyết Tính toán: Công trình của Turing về máy Turing phổ quát đã chứng minh về mặt lý thuyết khả năng của một cỗ máy thực hiện bất kỳ quy trình tính toán

nào có thể mô tả bằng thuật toán, mở đường cho ý tưởng về máy tính có thể "suy nghĩ".

Sự ra đời chính thức và những thập kỷ đầu tiên (1956 - 1974): Thuật ngữ "Artificial Intelligence" được đặt ra bởi John McCarthy tại Hội thảo Dartmouth vào năm 1956, sự kiện này chính là khai sinh của AI. Các chương trình như Logic Theorist và General Problem Solver là những bước đầu tiên trong việc mô phỏng khả năng suy luận của con người:

- ♦ **Logic Theorist:** Chương trình do Allen Newell và Herbert A. Simon phát triển, có khả năng chứng minh các định lý toán học.
- ♦ **General Problem Solver (GPS):** Cũng do Newell và Simon tạo ra, cố gắng mô phỏng các bước giải quyết vấn đề của con người.
- ♦ **Ngôn ngữ lập trình LISP:** Được phát triển bởi John McCarthy, trở thành ngôn ngữ ưa thích cho nghiên cứu AI trong nhiều thập kỷ.
- ♦ Các chương trình chơi cờ vua, giải toán đại số, và các hệ thống xử lý ngôn ngữ tự nhiên sơ khai cũng ra đời trong giai đoạn này. Tuy nhiên, tại thời điểm này AI vẫn chưa thể phát triển mạnh mẽ do các hệ thống phần cứng chưa đủ mạnh.

"Mùa đông AI" và sự hồi sinh (1974 - 1980s và cuối 1980s - đầu 1990s): Khi các lời hứa hẹn về các hệ thống AI không thể thành hiện thực, nguồn tài trợ cho các nhà nghiên cứu bị cắt giảm đáng kể, dẫn đến giai đoạn được gọi là "Mùa đông AI" lần thứ nhất.

- **Sự trỗi dậy của Hệ chuyên gia (Expert Systems):** Vào những năm 1980, AI có sự hồi sinh ngắn ngủi với sự thành công của các hệ chuyên gia – chương trình mô phỏng kiến thức và khả năng ra quyết định như 1 chuyên gia. Tuy nhiên, việc xây dựng và bảo trì các hệ chuyên gia này tỏ ra tốn kém và khó mở rộng.
- **Mùa đông AI thứ hai:** Sự sụp đổ của thị trường máy LISP và những hạn chế của hệ chuyên gia đã dẫn đến một giai đoạn suy thoái khác vào cuối những năm 1980 và đầu 1990s.

Cuộc cách mạng Học máy và Học sâu (Từ giữa 1990s đến nay): Sự hồi sinh mạnh mẽ và bền vững của AI bắt đầu từ giữa những năm 1990 và tăng tốc vào thế kỷ 21, được thúc đẩy bởi các yếu tố như lượng dữ liệu khổng lồ, sức mạnh của các hệ thống máy tính hiện đại và đặc biệt là sự phát triển của các kỹ thuật Học máy và Học sâu.

- ♦ **Học máy (Machine Learning):** Thay vì lập trình rõ ràng mọi quy tắc, các nhà khoa học tập trung vào việc tạo ra các thuật toán cho phép máy tính tự học hỏi từ dữ liệu. Các phương pháp như Máy Vector Hỗ trợ (SVM), Cây quyết định, và các thuật toán dựa trên xác suất đã đạt được nhiều thành công.
- ♦ **Deep Blue đánh bại Kasparov (1997):** Sự kiện máy tính Deep Blue của IBM đánh bại nhà vô địch cờ vua thế giới Garry Kasparov là một cột mốc quan trọng, cho thấy khả năng xử lý và tính toán vượt trội của máy móc.

- ♦ **Học sâu (Deep Learning):** Sự hồi sinh của mạng nơ-ron nhân tạo dưới dạng các mô hình học sâu với nhiều lớp (deep neural networks) đã tạo ra những đột phá ngoạn mục trong các lĩnh vực như nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên và nhận dạng giọng nói từ khoảng năm 2010 trở đi. Các thành công như AlphaGo của Google DeepMind đánh bại nhà vô địch cờ vây thế giới Lee Sedol (2016) đã khẳng định sức mạnh của học sâu.

1.3. Các khái niệm cơ bản

1.3.1. Các loại AI:

Dựa trên năng lực và mục tiêu, AI thường được phân loại thành:

Trí tuệ nhân tạo hẹp (Artificial Narrow Intelligence - ANI): Còn gọi là AI yếu (Weak AI). Đây là loại AI được thiết kế và huấn luyện cho một nhiệm vụ cụ thể hoặc một tập hợp các nhiệm vụ hạn chế. Hầu hết các ứng dụng AI chúng ta thấy ngày nay (nhận dạng khuôn mặt, trợ lý ảo, xe tự lái cấp độ thấp, hệ thống gợi ý) đều thuộc loại ANI. Chúng có thể thực hiện nhiệm vụ của mình rất tốt, thậm chí vượt trội con người, nhưng không có ý thức hay khả năng hiểu biết thực sự như con người.

Trí tuệ nhân tạo tổng quát (Artificial General Intelligence - AGI): Còn gọi là AI mạnh (Strong AI). Đây là loại AI giả định có khả năng hiểu, học và áp dụng kiến thức trên nhiều lĩnh vực khác nhau, tương đương với trí tuệ của con người. AGI có thể tự suy luận, lập kế hoạch, giải quyết các vấn đề phức tạp và học hỏi từ kinh nghiệm một cách linh hoạt. Hiện nay, AGI vẫn là mục tiêu nghiên cứu dài hạn và chưa tồn tại.

Siêu trí tuệ nhân tạo (Artificial Superintelligence - ASI): Đây là loại AI giả định vượt xa trí tuệ của những bộ óc thông minh nhất của con người trong hầu hết mọi lĩnh vực, bao gồm cả sáng tạo khoa học, trí tuệ xã hội và kỹ năng thực tiễn. ASI là một khái niệm mang tính lý thuyết và tiềm ẩn nhiều hệ lụy khó lường.

1.3.2. Học máy (Machine Learning - ML):

ML là một nhánh quan trọng của AI, tập trung vào việc phát triển các thuật toán cho phép máy tính "học" từ dữ liệu mà không cần được lập trình rõ ràng cho từng trường hợp. Thay vì viết mã để giải quyết một vấn đề, các nhà phát triển cung cấp dữ liệu cho thuật toán ML, và thuật toán đó sẽ tự xây dựng một mô hình để đưa ra dự đoán hoặc quyết định. Có ba loại hình học máy chính:

Học có giám sát (Supervised Learning): Thuật toán được huấn luyện trên một tập dữ liệu đã được gán nhãn (labeled data), nghĩa là mỗi điểm dữ liệu đầu vào đều có một kết quả đầu ra "đúng" tương ứng. Mục tiêu là học ra một hàm ánh xạ từ đầu vào đến đầu ra. Ví dụ: huấn luyện mô hình nhận dạng email rác bằng cách cung cấp các email đã được đánh dấu là "rác" hoặc "không phải rác"; huấn luyện mô hình dự đoán giá nhà dựa trên dữ liệu lịch sử về giá và đặc điểm nhà. Các nhiệm vụ phổ biến là phân loại (classification) và hồi quy (regression).

Học không giám sát (Unsupervised Learning): Thuật toán làm việc với dữ liệu không được gán nhãn. Mục tiêu là tự khám phá các cấu trúc, mẫu hoặc mối quan hệ tiềm ẩn trong dữ liệu. Ví dụ: phân cụm khách hàng dựa trên hành vi mua

sắc (clustering); giảm chiều dữ liệu để trực quan hóa hoặc đơn giản hóa (dimensionality reduction); phát hiện các giao dịch bất thường (anomaly detection).

Học tăng cường (Reinforcement Learning - RL): Thuật toán học cách đưa ra một chuỗi các quyết định tối ưu để đạt được mục tiêu trong một môi trường động. Nó học thông qua cơ chế thử-và-sai (trial-and-error), nhận "phần thưởng" (reward) cho các hành động tốt và "hình phạt" (penalty) cho các hành động xấu. Ví dụ: huấn luyện AI chơi game (như AlphaGo), điều khiển robot, tối ưu hóa hệ thống điều khiển công nghiệp.

1.3.3. Học sâu (Deep Learning - DL):

DL là một nhánh con của ML, sử dụng các mạng nơ-ron nhân tạo (ANN) với nhiều lớp (layers) để học các biểu diễn dữ liệu phức tạp và trừu tượng. Các lớp sâu này cho phép mô hình tự động trích xuất các đặc trưng (features) từ dữ liệu thô ở các cấp độ khác nhau, từ đơn giản đến phức tạp. DL đã tạo ra những đột phá lớn trong nhiều nhiệm vụ AI, đặc biệt là với dữ liệu phi cấu trúc như hình ảnh, âm thanh và văn bản. Có các mạng nơ-ron phổ biến như:

Mạng nơ-ron nhân tạo (Artificial Neural Networks - ANN): Lấy cảm hứng từ cấu trúc nơ-ron sinh học trong não người, ANN bao gồm các nút (nơ-ron) được kết nối với nhau theo các lớp. Mỗi kết nối có một trọng số (weight) thể hiện tầm quan trọng của tín hiệu. Mạng học bằng cách điều chỉnh các trọng số này dựa trên dữ liệu huấn luyện.

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN): Là một loại mạng nơ-ron sâu đặc biệt hiệu quả cho việc xử lý dữ liệu dạng lưới, như hình ảnh. CNN sử dụng các bộ lọc tích chập (convolutional filters) để tự động học các đặc trưng không gian phân cấp từ hình ảnh (ví dụ: cạnh, góc, hình dạng, đối tượng). Chúng là nền tảng cho hầu hết các ứng dụng thị giác máy tính hiện đại.

Mạng nơ-ron hồi quy (Recurrent Neural Networks - RNN): Được thiết kế để xử lý dữ liệu dạng chuỗi (sequential data), nơi thứ tự thông tin là quan trọng, chẳng hạn như văn bản hoặc chuỗi thời gian. RNN có các kết nối vòng lặp cho phép thông tin từ các bước thời gian trước đó được lưu giữ và ảnh hưởng đến các bước sau. Các biến thể như LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) giải quyết được vấn đề nhớ thông tin dài hạn, rất hữu ích trong dịch máy, tạo văn bản, nhận dạng giọng nói.

1.3.4. Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP):

NLP là lĩnh vực giao thoa giữa AI, khoa học máy tính và ngôn ngữ học, tập trung vào việc giúp máy tính hiểu, diễn giải và tạo ra ngôn ngữ của con người (cả văn bản nói và viết). Các nhiệm vụ NLP bao gồm: phân tích cú pháp, nhận dạng thực thể tên (Named Entity Recognition - NER), phân tích cảm xúc, dịch máy, trả lời câu hỏi, tóm tắt văn bản, tạo văn bản, nhận dạng giọng nói và tổng hợp giọng nói. Sự phát triển của

các mô hình ngôn ngữ lớn (Large Language Models - LLM) dựa trên kiến trúc Transformer (như BERT, GPT) đã cách mạng hóa lĩnh vực này.

1.3.5. Thị giác máy tính (Computer Vision):

Thị giác máy tính là lĩnh vực giúp máy tính "nhìn" và hiểu được nội dung của hình ảnh và video. Nó bao gồm các nhiệm vụ như: nhận dạng đối tượng, phân loại hình ảnh, dò tìm đối tượng (object detection), phân vùng ảnh (image segmentation), nhận dạng khuôn mặt, theo dõi đối tượng, tái tạo 3D từ ảnh 2D. CNN là công nghệ chủ đạo trong hầu hết các ứng dụng thị giác máy tính hiện đại.

1.3.6. Robotics và Hệ thống tự hành:

AI đóng vai trò trung tâm trong việc tạo ra các robot và hệ thống tự hành thông minh hơn. AI cung cấp khả năng nhận thức (thông qua thị giác máy tính, cảm biến), lập kế hoạch (path planning), ra quyết định và điều khiển chuyển động cho robot. Các ứng dụng bao gồm robot công nghiệp, robot dịch vụ, xe tự lái, máy bay không người lái (drones). Học tăng cường thường được sử dụng để huấn luyện robot thực hiện các nhiệm vụ phức tạp trong môi trường thực tế.

1.4. Ứng dụng của trí tuệ nhân tạo

Ngày nay, AI đã được ứng dụng trong rất nhiều khía cạnh của đời sống.

1.4.1. Y tế và Chăm sóc sức khỏe:

Chẩn đoán hình ảnh: AI (đặc biệt là CNN) có thể phân tích hình ảnh y tế (X-quang, CT, MRI) để phát hiện sớm các dấu hiệu bệnh lý như ung thư, bệnh võng mạc tiểu đường, bệnh tim mạch với độ chính xác cao, đôi khi vượt cả bác sĩ X-quang.

Phát hiện và dự đoán bệnh: Phân tích dữ liệu bệnh án điện tử, dữ liệu gen để dự đoán nguy cơ mắc bệnh, phát hiện sớm các đợt bùng phát dịch bệnh.

Phát triển thuốc: AI tăng tốc quá trình khám phá và phát triển thuốc mới bằng cách phân tích dữ liệu sinh học phức tạp, dự đoán hiệu quả và độc tính của các hợp chất.

Phẫu thuật bằng robot: Các hệ thống phẫu thuật được hỗ trợ bởi AI giúp tăng độ chính xác, giảm xâm lấn và cải thiện kết quả phẫu thuật.

Trợ lý ảo y tế và chatbot: Cung cấp thông tin sức khỏe, nhắc nhở uống thuốc, hỗ trợ bệnh nhân quản lý bệnh mãn tính.

Y học cá nhân hóa: Phân tích dữ liệu cá nhân để đưa ra phác đồ điều trị và phòng ngừa phù hợp nhất cho từng bệnh nhân.

1.4.2. Kinh doanh, Tài chính và Tiếp thị:

Dịch vụ khách hàng: Chatbot và trợ lý ảo hỗ trợ khách hàng 24/7, trả lời câu hỏi, xử lý yêu cầu, giảm thời gian chờ đợi và chi phí vận hành.

Tiếp thị cá nhân hóa: Phân tích hành vi người dùng để đưa ra các đề xuất sản phẩm, nội dung và quảng cáo phù hợp, tăng tỷ lệ chuyển đổi.

Phát hiện gian lận: AI phân tích các mẫu giao dịch để phát hiện các hoạt động đáng ngờ và gian lận trong tài chính, ngân hàng, bảo hiểm.

Giao dịch thuật toán (Algorithmic Trading): Các mô hình AI tự động thực hiện các giao dịch tài chính dựa trên phân tích thị trường thời gian thực để tối đa hóa lợi nhuận.

Quản lý rủi ro: Đánh giá rủi ro tín dụng, rủi ro thị trường và các loại rủi ro khác một cách chính xác hơn.

Phân tích kinh doanh thông minh (BI): Khai thác dữ liệu kinh doanh để tìm ra các xu hướng, dự báo doanh số, tối ưu hóa hoạt động.

1.4.3. Giao thông vận tải và Logistics:

Xe tự lái (Autonomous Vehicles): AI là công nghệ cốt lõi cho phép xe tự lái nhận thức môi trường (qua camera, lidar, radar), đưa ra quyết định lái xe và điều khiển phương tiện.

Tối ưu hóa lộ trình: Các ứng dụng như Google Maps sử dụng AI để phân tích tình hình giao thông và đề xuất lộ trình nhanh nhất. Các công ty logistics dùng AI để tối ưu hóa tuyến đường giao hàng, tiết kiệm thời gian và nhiên liệu.

Quản lý giao thông thông minh: Phân tích luồng giao thông để điều khiển đèn tín hiệu, dự đoán tắc nghẽn và điều phối phương tiện hiệu quả.

Quản lý chuỗi cung ứng: Dự báo nhu cầu, tối ưu hóa tồn kho, theo dõi và quản lý hàng hóa trong chuỗi cung ứng.

1.4.4. Sản xuất và Công nghiệp:

Bảo trì dự đoán (Predictive Maintenance): AI phân tích dữ liệu từ cảm biến trên máy móc để dự đoán thời điểm hỏng hóc, cho phép lên kế hoạch bảo trì trước khi sự cố xảy ra, giảm thời gian ngừng hoạt động.

Kiểm soát chất lượng: Hệ thống thị giác máy tính tự động kiểm tra sản phẩm trên dây chuyền sản xuất để phát hiện lỗi với tốc độ và độ chính xác cao.

Robot công nghiệp thông minh: Robot được trang bị AI có thể thực hiện các nhiệm vụ phức tạp hơn, linh hoạt hơn và hợp tác an toàn với con người (cobots).

Tối ưu hóa quy trình sản xuất: Phân tích dữ liệu sản xuất để tìm ra cách cải thiện hiệu quả, giảm lãng phí năng lượng và nguyên vật liệu.

1.4.5. Giáo dục và Đào tạo:

Hệ thống học tập cá nhân hóa: AI điều chỉnh nội dung và tốc độ học tập phù hợp với khả năng và nhu cầu của từng học sinh, sinh viên.

Gia sư thông minh (Intelligent Tutoring Systems): Cung cấp phản hồi tức thì, giải đáp thắc mắc và hướng dẫn học sinh giải quyết vấn đề.

Chấm điểm tự động: AI có thể hỗ trợ chấm điểm các bài kiểm tra trắc nghiệm và cả bài luận (ở một mức độ nhất định), giải phóng thời gian cho giáo viên.

Phân tích dữ liệu học tập: Giúp nhà trường và giáo viên hiểu rõ hơn về tiến trình học tập của học sinh, xác định các khó khăn và can thiệp kịp thời.

Quản lý và hành chính: Tự động hóa các công việc hành chính, xếp lịch, quản lý tài nguyên.

1.4.6. Giải trí và Truyền thông:

Hệ thống gợi ý (Recommendation Systems): Các nền tảng như Netflix, Spotify, YouTube sử dụng AI để phân tích lịch sử xem/nghe và sở thích của người dùng để đề xuất phim, nhạc, video phù hợp.

AI trong trò chơi điện tử: Tạo ra các nhân vật không phải người chơi (NPC) có hành vi thông minh và thực tế hơn, điều chỉnh độ khó của game, tạo nội dung game tự động.

Tạo nội dung tự động: AI có thể viết tin tức, tạo nhạc, vẽ tranh, tạo video ở mức độ cơ bản.

Phụ đề và lồng tiếng tự động: NLP và công nghệ giọng nói giúp tạo phụ đề và lồng tiếng cho video một cách nhanh chóng.

Kiểm duyệt nội dung: Tự động phát hiện và lọc các nội dung không phù hợp trên các nền tảng trực tuyến.

1.4.7. Nông nghiệp thông minh (Precision Agriculture):

Phân tích đất và cây trồng: AI phân tích hình ảnh từ drone hoặc vệ tinh, dữ liệu từ cảm biến để theo dõi sức khỏe cây trồng, phát hiện sâu bệnh, đánh giá nhu cầu nước và dinh dưỡng.

Tưới tiêu và bón phân chính xác: Tự động điều chỉnh lượng nước và phân bón cho từng khu vực nhỏ trong cánh đồng dựa trên dữ liệu phân tích, tiết kiệm tài nguyên và tăng năng suất.

Robot nông nghiệp: Robot tự động gieo hạt, làm cỏ, thu hoạch.

Dự báo năng suất: Phân tích dữ liệu thời tiết, đất đai, cây trồng để dự báo sản lượng thu hoạch.

1.4.8. Khoa học và Nghiên cứu:

Phân tích dữ liệu lớn: AI giúp các nhà khoa học xử lý và phân tích khối lượng dữ liệu khổng lồ từ các thí nghiệm, kính thiên văn, máy gia tốc hạt, giải mã gen... để tìm ra các mẫu và khám phá mới.

Mô phỏng khoa học: Tạo ra các mô hình mô phỏng phức tạp trong vật lý, hóa học, sinh học, khí hậu học.

Khám phá vật liệu mới: Dự đoán tính chất của các vật liệu chưa được tổng hợp.

Tự động hóa thí nghiệm: Robot được điều khiển bởi AI thực hiện các thí nghiệm lặp đi lặp lại.

1.4.9. An ninh và Quốc phòng:

Phân tích tình báo: Xử lý lượng lớn thông tin (văn bản, hình ảnh, video) để phát hiện các mối đe dọa, phân tích xu hướng.

An ninh mạng (Cybersecurity): Phát hiện và phản ứng với các cuộc tấn công mạng, phần mềm độc hại một cách tự động và nhanh chóng.

Giám sát và nhận dạng: Hệ thống camera thông minh sử dụng nhận dạng khuôn mặt, nhận dạng hành vi bất thường để giám sát an ninh.

Phương tiện tự hành quân sự: Máy bay không người lái, phương tiện mặt đất tự hành cho các nhiệm vụ trinh sát, hậu cần, và tiềm ẩn cả nhiệm vụ chiến đấu (gây tranh cãi).

1.4.10. Đời sống hàng ngày và Nhà thông minh:

Trợ lý ảo cá nhân: Siri, Google Assistant, Alexa giúp thực hiện các tác vụ bằng giọng nói như đặt báo thức, kiểm tra thời tiết, phát nhạc, điều khiển thiết bị thông minh.

Công cụ tìm kiếm: Google Search và các công cụ khác sử dụng AI để hiểu truy vấn của người dùng và trả về kết quả liên quan nhất.

Dịch thuật tự động: Google Translate và các dịch vụ tương tự phá vỡ rào cản ngôn ngữ.

Bộ lọc thư rác (Spam Filters): Tự động phân loại và loại bỏ email không mong muốn.

Nhà thông minh (Smart Homes): AI điều khiển hệ thống chiếu sáng, nhiệt độ, an ninh, thiết bị gia dụng để tối ưu hóa sự tiện nghi và tiết kiệm năng lượng.

Mạng xã hội: AI sắp xếp bảng tin (news feed), gợi ý kết bạn, gắn thẻ ảnh tự động.

Những ứng dụng kể trên chỉ là một phần nhỏ trong bức tranh lớn về tác động của AI. Khi công nghệ không ngừng tiến bộ, chúng ta sẽ chứng kiến AI dần được ứng dụng sâu rộng hơn vào nhiều lĩnh vực khác, mở ra những khả năng và cơ hội mà trước đây tưởng chừng chỉ có trong trí tưởng tượng.

1.5. Thách thức và Rủi ro đi kèm với AI

Bên cạnh những lợi ích to lớn, sự phát triển và ứng dụng AI cũng đặt ra nhiều thách thức và rủi ro đáng kể cần được quan tâm.

1.5.1. Thách thức về kỹ thuật và dữ liệu:

Nhu cầu dữ liệu lớn và chất lượng: Nhiều mô hình AI, đặc biệt là học sâu, đòi hỏi lượng lớn dữ liệu huấn luyện chất lượng cao và được gán nhãn cẩn thận, việc thu thập và chuẩn bị dữ liệu này có thể tốn kém và mất thời gian.

Sức mạnh tính toán: Huấn luyện các mô hình AI phức tạp đòi hỏi tài nguyên tính toán khổng lồ, gây tốn kém và tiêu thụ nhiều năng lượng.

Tính giải thích được (Explainability / Interpretability): Nhiều mô hình AI, đặc biệt là các mạng nơ-ron sâu, hoạt động như những "hộp đen" (black boxes). Rất khó để hiểu tại sao chúng lại đưa ra một quyết định hay dự đoán cụ thể. Điều này gây khó khăn trong các lĩnh vực nhạy cảm như y tế, tài chính, pháp lý, nơi cần sự minh bạch và giải trình.

Tính mạnh mẽ và Tổng quát hóa: Các mô hình AI có thể hoạt động tốt trên dữ liệu huấn luyện nhưng lại thất bại khi gặp phải dữ liệu mới hoặc hơi khác biệt trong thế giới thực (vấn đề tổng quát hóa). Chúng cũng có thể dễ bị tấn công bởi các dữ liệu đầu vào được tạo ra một cách cố ý (adversarial attacks).

1.5.2. Các vấn đề đạo đức và xã hội:

Việc làm và Bất bình đẳng kinh tế: Tự động hóa do AI có thể dẫn đến mất việc làm trong một số ngành nghề, đặc biệt là các công việc có tính lặp lại. Điều này có thể làm gia tăng bất bình đẳng kinh tế nếu không có các chính sách hỗ trợ chuyển đổi nghề nghiệp và tái đào tạo phù hợp. Khoảng cách về kỹ năng AI cũng có thể tạo ra sự phân hóa trong lực lượng lao động.

Thiên vị và Công bằng: Các hệ thống AI được huấn luyện trên dữ liệu lịch sử có thể học và khuếch đại những định kiến, thành kiến vô thức tồn tại trong xã hội (ví dụ: thiên vị về giới tính, chủng tộc). Điều này có thể dẫn đến các quyết định không công bằng trong tuyển dụng, cho vay tín dụng, tư pháp hình sự, và nhiều lĩnh vực khác. Đảm bảo sự công bằng và không thiên vị trong AI là một thách thức lớn.

Quyền riêng tư và Giám sát: Việc thu thập lượng lớn dữ liệu cá nhân để huấn luyện AI làm dấy lên lo ngại nghiêm trọng về quyền riêng tư. Các công nghệ như nhận dạng khuôn mặt và phân tích hành vi có thể bị lạm dụng cho mục đích giám sát hàng loạt, xâm phạm quyền tự do cá nhân.

An toàn và An ninh: Các hệ thống AI, đặc biệt là những hệ thống điều khiển cơ sở hạ tầng quan trọng (năng lượng, giao thông) hoặc vũ khí tự hành, phải đảm bảo hoạt động an toàn và không gây hại ngoài ý muốn. Nguy cơ AI bị tấn công mạng hoặc bị sử dụng cho mục đích xấu là rất hiện hữu.

Trách nhiệm pháp lý và Giải trình: Khi một hệ thống AI gây ra lỗi hoặc thiệt hại, việc xác định ai là người chịu trách nhiệm (nhà phát triển, người dùng, hay chính hệ thống AI?) trở nên phức tạp. Thiếu tính giải thích được của AI càng làm vấn đề này thêm khó khăn.

1.5.3. Nhu cầu về quản lý và pháp lý:

Sự phát triển nhanh chóng của AI đòi hỏi phải có các khung pháp lý, quy định và tiêu chuẩn đạo đức phù hợp để định hướng sự phát triển và ứng dụng AI một cách có trách nhiệm, đảm bảo lợi ích cho xã hội đồng thời giảm thiểu các rủi ro tiềm ẩn. Việc xây dựng các quy định này cần sự hợp tác quốc tế và sự tham gia của nhiều bên liên quan (chính phủ, ngành công nghiệp, học viện, xã hội dân sự).

1.6. Tương lai của Trí tuệ nhân tạo

Tương lai của Trí tuệ nhân tạo hứa hẹn sẽ tiếp tục mang đến những thay đổi sâu sắc và khó lường. Mặc dù không thể dự đoán chính xác mọi thứ, một số xu hướng và chủ đề chính đang định hình con đường phía trước.

1.6.1. Xu hướng phát triển công nghệ:

Tiếp tục tiến bộ trong ML và DL: Các mô hình sẽ trở nên mạnh mẽ hơn, hiệu quả hơn về dữ liệu và năng lượng. Các kỹ thuật mới như học tự giám sát (self-supervised learning), học chuyển giao (transfer learning), và học liên kết (federated learning) sẽ ngày càng phổ biến.

AI đa phương thức (Multimodal AI): Các hệ thống AI có khả năng hiểu và xử lý thông tin từ nhiều nguồn khác nhau cùng lúc (văn bản, hình ảnh, âm thanh, video) sẽ trở nên tinh vi hơn, gần với cách con người nhận thức thế giới.

AI giải thích được (Explainable AI - XAI): Nghiên cứu về XAI sẽ được đẩy mạnh để làm cho các quyết định của AI trở nên minh bạch và dễ hiểu hơn, đặc biệt trong các ứng dụng quan trọng.

AI trên thiết bị biên (Edge AI): Thay vì xử lý trên đám mây, các tác vụ AI sẽ ngày càng được thực hiện trực tiếp trên các thiết bị cục bộ (điện thoại, cảm biến, xe cộ), giúp tăng tốc độ, giảm độ trễ và bảo vệ quyền riêng tư tốt hơn.

1.6.2. Hướng tới các dạng trí tuệ nhân tạo cao hơn:

Việc đạt được AGI – AI có trí tuệ tương đương con người trên mọi lĩnh vực – vẫn là một mục tiêu xa vời và gây tranh cãi. Nhiều nhà nghiên cứu tin rằng chúng ta vẫn còn cách rất xa AGI, trong khi một số khác lạc quan hơn về những đột phá trong tương lai gần. Con đường đến AGI (nếu có) có thể sẽ không tuyến tính và đòi hỏi những bước nhảy vọt về mặt lý thuyết và kiến trúc mô hình. Nếu AGI trở thành hiện thực, nó sẽ mang lại những tác động biến đổi chưa từng có, cả tích cực lẫn tiêu cực.

1.6.3. Tích hợp AI sâu rộng vào xã hội:

AI sẽ ngày càng trở nên vô hình và được tích hợp liền mạch vào các sản phẩm, dịch vụ và quy trình hàng ngày. Từ trợ lý cá nhân thông minh hơn, hệ thống giao thông tự hành hoàn toàn, đến các công cụ chẩn đoán y tế tại nhà và môi trường làm việc cộng tác giữa người và AI. Sự tương tác giữa người và máy sẽ trở nên tự nhiên và hiệu quả hơn.

1.6.4. Vai trò của đạo đức và quản trị AI:

Khi AI trở nên mạnh mẽ và phổ biến hơn, các vấn đề về đạo đức, công bằng, minh bạch và trách nhiệm giải trình sẽ ngày càng trở nên quan trọng. Sẽ có nhu cầu cấp thiết về việc xây dựng các nguyên tắc đạo đức vững chắc, các khung pháp lý linh hoạt và các cơ chế quản trị hiệu quả ở cấp độ quốc gia và quốc tế để định hướng sự phát triển AI một cách có trách nhiệm. Cuộc đối thoại toàn cầu về tương lai của AI và các quy tắc ứng xử sẽ tiếp tục diễn ra sôi nổi.

1.6.5. AI và việc giải quyết các thách thức toàn cầu:

AI có tiềm năng đóng góp đáng kể vào việc giải quyết những thách thức lớn nhất mà nhân loại đang đối mặt, như biến đổi khí hậu, bệnh tật, nghèo đói, và quản lý tài nguyên bền vững. Việc khai thác tiềm năng này đòi hỏi sự đầu tư chiến lược, hợp tác quốc tế và cách tiếp cận lấy con người làm trung tâm.

CHƯƠNG 2: CÁC KỸ THUẬT HỌC SÂU

Chương này đi sâu vào các kỹ thuật và kiến trúc cốt lõi của Học sâu (Deep Learning - DL), một nhánh mạnh mẽ của Học máy (Machine Learning - ML) đã tạo ra những đột phá trong nhiều lĩnh vực, từ nhận dạng hình ảnh đến xử lý ngôn ngữ tự nhiên. Học sâu mô phỏng cách bộ não con người xử lý thông tin thông qua các mạng nơ-ron nhân tạo nhiều lớp.

2.1. Nền tảng Cốt lõi và Cơ chế Huấn luyện

2.1.1. Các Thành phần Cơ bản

2.1.1.1. Các Loại Lớp (Layer Types)

Neural network (mạng nơ-ron) là một mô hình học máy lấy cảm hứng từ cấu trúc não người, được tạo thành từ các đơn vị tính toán được gọi là nơ-ron, tổ chức thành các lớp kết nối với nhau. Các loại lớp phổ biến bao gồm:

- **Dense (Fully Connected):** Lớp cơ bản nhất, trong đó mỗi nơ-ron (nút tính toán) nhận đầu vào từ tất cả các nơ-ron ở lớp trước và kết nối với tất cả các nơ-ron ở lớp sau. Lớp này học các mẫu tổng quát trên toàn bộ đầu vào. Thường được sử dụng làm các lớp cuối cùng trong mạng phân loại hoặc hồi quy để đưa ra dự đoán cuối cùng dựa trên các đặc trưng đã được trích xuất bởi các lớp trước đó.
- **Convolutional:** Chuyên xử lý dữ liệu có cấu trúc dạng lưới (grid-like data) như hình ảnh (2D) hoặc tín hiệu âm thanh (1D). Lớp này sử dụng các bộ lọc (kernels hoặc filters) nhỏ trượt qua đầu vào để phát hiện các mẫu cục bộ (local patterns) như cạnh, góc, kết cấu. Ưu điểm lớn là khả năng chia sẻ tham số (parameter sharing), giúp giảm đáng kể số lượng trọng số cần học và làm cho mô hình bất biến với vị trí của đặc trưng (translation invariance)..
- **Recurrent:** Thiết kế đặc biệt cho dữ liệu tuần tự (sequential data) như văn bản, chuỗi thời gian, hoặc giọng nói, nơi thứ tự thông tin là quan trọng. Các nơ-ron trong lớp hồi quy có kết nối vòng lặp lại chính nó, tạo thành một "bộ nhớ" cho phép thông tin từ các bước thời gian trước đó ảnh hưởng đến tính toán ở bước hiện tại và tương lai..
- **Embedding:** Chuyển đổi dữ liệu hạng mục (categorical data), đặc biệt là các từ trong văn bản, thành các vector số dày đặc (dense vectors) có chiều thấp hơn. Các vector này, gọi là embeddings, nắm bắt được mối quan hệ ngữ nghĩa giữa các hạng mục (ví dụ: các từ đồng nghĩa sẽ có vector embedding gần nhau trong không gian). Điều này hiệu quả hơn nhiều so với các biểu diễn thưa như one-hot encoding.
- **Normalization:** Các lớp này điều chỉnh và chuẩn hóa đầu ra của lớp trước đó để giúp quá trình huấn luyện ổn định hơn và nhanh hơn. Ví dụ phổ biến là Batch Normalization (chuẩn hóa theo lô) và Layer Normalization (chuẩn hóa theo lớp). Chúng giúp giải quyết vấn đề thay đổi phân phối đầu vào của các lớp (internal covariate shift).
- **Pooling:** Thường đi kèm sau lớp tích chập trong CNN, lớp pooling giảm kích thước không gian (chiều rộng và chiều cao) của các bản đồ đặc trưng (feature maps). Điều

này giúp giảm số lượng tham số và tính toán trong mạng, đồng thời làm cho các đặc trưng học được trở nên mạnh mẽ hơn đối với các biến đổi nhỏ về vị trí. Các loại phổ biến là MaxPooling (lấy giá trị lớn nhất trong một vùng) và AveragePooling (lấy giá trị trung bình).

2.1.1.2. Hàm Kích hoạt (Activation Functions)

Được áp dụng cho đầu ra của mỗi nơ-ron (trừ lớp đầu vào), hàm kích hoạt đưa tính phi tuyến (non-linearity) vào mạng. Nếu không có hàm kích hoạt phi tuyến, một mạng nơ-ron sâu chỉ tương đương với một mô hình tuyến tính duy nhất, hạn chế khả năng học các mối quan hệ phức tạp trong dữ liệu.

- **ReLU (Rectified Linear Unit):** $f(x) = \max(0, x)$
 - ♦ Phổ biến nhất hiện nay, tính toán nhanh, giảm thiểu vấn đề gradient biến mất
 - ♦ Tuy nhiên, nó có thể gặp vấn đề "nơ-ron chết" (dying ReLU) khi một nơ-ron luôn có đầu vào âm, khiến gradient của nó luôn bằng 0 và không thể cập nhật trọng số.
- **Sigmoid:** $f(x) = \frac{1}{1+e^{-x}}$
 - ♦ Đầu ra vào khoảng (0, 1). Từng rất phổ biến, đặc biệt hữu ích cho lớp đầu ra của bài toán phân loại nhị phân để diễn giải kết quả như một xác suất.
 - ♦ Tuy nhiên, nó dễ bị bão hòa ở hai đầu (gradient gần bằng 0), làm chậm quá trình huấn luyện, và đầu ra không có tâm là 0.
- **Tanh:** $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
 - ♦ Đầu ra vào khoảng (-1, 1), với tâm là 0 (đây là một ưu điểm so với Sigmoid). Vẫn gặp vấn đề bão hòa gradient tương tự Sigmoid.
- **Softmax:** $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n (e^{x_j})}$
 - ♦ Thường chỉ được sử dụng ở lớp đầu ra cho bài toán phân loại nhiều lớp (multi-class classification). Nó biến đổi một vector điểm số (logits) thành một vector phân phối xác suất, trong đó tổng các phần tử bằng 1.
- **Leaky ReLU:** $f(x) = \max(\alpha x, x)$
 - ♦ với α là một hằng số nhỏ (ví dụ: 0.01 hoặc 0.2). Là một cải tiến của ReLU, nó cho phép một gradient nhỏ khác 0 khi đầu vào âm, giúp giải quyết vấn đề "nơ-ron chết". Các biến thể khác bao gồm PReLU (tham số α được học) và ELU.

2.1.2. Cơ chế Huấn luyện Cơ bản

Quá trình "học" của mạng nơ-ron là điều chỉnh các trọng số (weights) để giảm thiểu sự khác biệt giữa dự đoán và kết quả thực tế.

2.1.2.1. Hàm Mất mát (Loss Functions)

Hàm mất mát (loss function) đo lường mức độ khác biệt giữa đầu ra dự đoán của mô hình và giá trị thực tế, làm cơ sở cho quá trình tối ưu hóa. Mục tiêu của quá trình huấn luyện là tối thiểu hóa hàm mất mát này. Dưới đây là 1 số hàm mất mát phổ biến.

- **Mean Squared Error (MSE):**

- **Công thức:** $MSE = \left(\frac{1}{n}\right) * \sum (y_{true} - y_{pred})^2$
- Thường dùng cho bài toán hồi quy (ví dụ: dự đoán giá trị liên tục)
- **Đặc điểm:** Nhạy cảm với giá trị ngoại lai(outliers) do sử dụng bình phương sai số
- **Ứng dụng:** Dự đoán giá nhà, dự báo nhiệt độ, các bài toán dự đoán giá trị liên tục
- **Mean Absolute Error (MAE):**
 - **Công thức:** $MAE = \left(\frac{1}{n}\right) * \sum |y_{true} - y_{pred}|$
 - Thường dùng cho bài toán hồi quy
 - **Đặc điểm:** Ít nhạy cảm với outliers hơn MSE
 - **Ứng dụng:** Các bài toán hồi quy khi dữ liệu có nhiều nhiễu
- **Binary Cross-Entropy:**
 - **Công thức:** $BCE = -\sum (y_{true} * \log(y_{pred}) + (1 - y_{true}) * \log(1 - y_{pred}))$
 - Thường dùng cho phân loại nhị phân
 - Phù hợp với đầu ra *sigmoid*
 - **Ứng dụng:** Phát hiện scam, chẩn đoán y tế, phân loại hình ảnh nhị phân
- **Categorical Cross-Entropy:**
 - **Công thức:** $CCE = -\sum (y_{true} * \log(y_{pred}))$
 - Thường dùng cho phân loại nhiều lớp với one-hot encoding
 - **Đặc điểm:** Sử dụng kết hợp với hàm kích hoạt softmax
 - **Ứng dụng:** Nhận dạng chữ viết tay, phân loại hình ảnh nhiều lớp, phân loại văn bản.

2.1.2.2. Tối ưu hóa dựa trên Gradient

Quá trình huấn luyện neural network là tiến trình tối ưu hóa lặp đi lặp lại để tìm trọng số tốt nhất bằng cách tối thiểu hóa hàm mất mát:

1. **Forward Pass:** Truyền dữ liệu đầu vào qua mạng để nhận dự đoán
2. **Tính Loss:** So sánh dự đoán với giá trị thực
3. **Backward Pass:** Tính toán gradient của loss theo các tham số
4. **Cập nhật Tham số:** Điều chỉnh trọng số theo gradient

2.1.2.3. Backpropagation (Tính toán Gradient)

Là phương pháp cốt lõi để huấn luyện mạng nơ-ron. Ý tưởng là tính toán gradient (đạo hàm riêng) của hàm mất mát theo từng tham số (trọng số và độ lệch) của mạng. Gradient chỉ ra hướng mà hàm mất mát tăng nhanh nhất; do đó, để giảm mất mát, ta cập nhật tham số theo hướng ngược lại của gradient. Quá trình này lặp đi lặp lại qua nhiều vòng (epochs).

- **Forward Pass (Lan truyền xuôi):** Dữ liệu đầu vào được đưa qua mạng từ lớp đầu tiên đến lớp cuối cùng, qua các phép tính tuyến tính và hàm kích hoạt, để tạo ra dự đoán cuối cùng.

- **Tính Loss:** So sánh dự đoán này với giá trị thực tế (nhãn) bằng cách sử dụng hàm mất mát đã chọn.
- **Backward Pass (Lan truyền ngược - Backpropagation):** Tính toán gradient của hàm mất mát đối với từng tham số trong mạng, bắt đầu từ lớp cuối cùng và lan truyền ngược về các lớp trước đó bằng cách sử dụng quy tắc chuỗi (chain rule) trong giải tích.
- **Cập nhật Tham số:** Điều chỉnh giá trị của các tham số theo hướng ngược lại của gradient tương ứng, nhân với một hệ số nhỏ gọi là tốc độ học (learning rate - η): $w_{\text{new}} = w_{\text{old}} - \eta \times \nabla wL$. Tốc độ học kiểm soát độ lớn của bước cập nhật.

2.1.2.4. Stochastic Gradient Descent và các Thuật toán Tối ưu hóa

Thay vì tính gradient trên toàn bộ tập dữ liệu (Batch Gradient Descent - tốn kém), SGD và các biến thể của nó tính gradient và cập nhật tham số trên từng mẫu hoặc một lô nhỏ (mini-batch) dữ liệu.

- **Stochastic Gradient Descent (SGD):**
 - ♦ Cập nhật: $w = w - \text{learning_rate} * \text{gradient}$
 - ♦ Đơn giản nhưng có thể dao động và hội tụ chậm
- **SGD với Momentum:**
 - ♦ Thêm "quán tính" để không mắc kẹt trong cực tiểu cục bộ.
 - ♦ Cập nhật: $v = \text{momentum} * v - \text{learning_rate} * \text{gradient}; w = w + v$
- **RMSprop:**
 - ♦ Điều chỉnh tốc độ học riêng cho từng tham số. Nó chia tốc độ học cho căn bậc hai của trung bình động các bình phương gradient gần đây. Giúp tăng tốc độ học cho các tham số có gradient nhỏ và giảm tốc độ học cho các tham số có gradient lớn.
- **Adam:**
 - ♦ Kết hợp cả ý tưởng của Momentum (sử dụng trung bình động của gradient) và RMSprop (sử dụng trung bình động của bình phương gradient). Nó tính toán tốc độ học thích ứng cho từng tham số. Thường là thuật toán tối ưu hóa mặc định hiệu quả và được sử dụng rộng rãi cho nhiều bài toán học sâu. Các biến thể khác như AdamW, Nadam cũng tồn tại.

```
# Ví dụ cấu hình optimizer trong Keras
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

2.2. Xây dựng và Đánh giá Mô hình Cơ bản

2.2.1. Mô hình Kết nối Dày đặc (Densely Connected Networks)

- Đây là kiến trúc mạng nơ-ron "cổ điển", bao gồm một lớp đầu vào, một hoặc nhiều lớp ẩn (hidden layers), và một lớp đầu ra. Mỗi nơ-ron trong một lớp được kết nối với tất cả các nơ-ron trong lớp liền kề trước đó (do đó có tên là "kết nối dày đặc" hay "fully connected").
- MLPs rất linh hoạt và có thể xấp xỉ bất kỳ hàm liên tục nào (universal approximation theorem), nhưng chúng không hiệu quả trong việc nắm bắt các cấu trúc cục bộ hoặc tuần tự trong dữ liệu.
- Chúng phù hợp nhất cho dữ liệu dạng bảng (tabular data), nơi mỗi hàng là một mẫu và mỗi cột là một đặc trưng, hoặc khi các đặc trưng đã được trích xuất thủ công hoặc bằng các phương pháp khác.
- Ví dụ: dự đoán khả năng khách hàng rời bỏ dịch vụ dựa trên thông tin nhân khẩu học và lịch sử sử dụng, phân loại ảnh dựa trên các đặc trưng đã được trích xuất sẵn (thay vì ảnh gốc).

```
# Xây dựng mô hình MLP đơn giản trong Keras
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)), # Làm phẳng đầu vào
    tf.keras.layers.Dense(128, activation='relu'), # Lớp ẩn 1
    tf.keras.layers.Dense(64, activation='relu'), # Lớp ẩn 2
    tf.keras.layers.Dense(10, activation='softmax') # Lớp đầu ra
])
```

Ưu điểm của MLP:

- Đơn giản, dễ hiểu và triển khai
- Hoạt động tốt với dữ liệu dạng bảng và vector đặc trưng
- Yêu cầu tính toán thấp hơn so với các kiến trúc phức tạp

Nhược điểm:

- Không tận dụng được cấu trúc không gian (như hình ảnh) hoặc thời gian (như chuỗi)
- Cần nhiều tham số hơn cho cùng một hiệu suất so với kiến trúc chuyên biệt
- Có thể dễ bị overfitting với dữ liệu kích thước lớn

2.2.2. Quy trình Đánh giá Mô hình

Việc đánh giá mô hình một cách khách quan là cực kỳ quan trọng để biết được hiệu suất thực sự của nó khi triển khai trong thực tế, tránh việc đánh giá quá lạc quan dựa trên dữ liệu huấn luyện.

2.2.2.1. Chia Tập Dữ liệu (Train/Validation/Test Split)

- **Training Set (Tập huấn luyện):** Phần lớn nhất của dữ liệu (thường 60-80%), được sử dụng để "dạy" mô hình bằng cách điều chỉnh các trọng số thông qua quá trình tối ưu hóa. Mô hình sẽ "nhìn thấy" và học hỏi từ tập này.

- **Validation Set (Tập kiểm định):** Một phần dữ liệu (thường 10-20%) được giữ riêng, không dùng để cập nhật trọng số. Thay vào đó, nó được sử dụng trong quá trình huấn luyện để:

- ♦ Theo dõi hiệu suất của mô hình trên dữ liệu chưa từng thấy.
- ♦ Tinh chỉnh các siêu tham số (hyperparameters) - là các tham số không được học trực tiếp từ dữ liệu, ví dụ: tốc độ học, số lượng lớp, số nơ-ron mỗi lớp, cường độ điều chuẩn.
- ♦ Quyết định khi nào nên dừng huấn luyện (Early Stopping) để tránh overfitting.

- **Test Set (Tập kiểm tra):** Một phần dữ liệu (thường 10-20%) được giữ riêng hoàn toàn và *chỉ được sử dụng một lần duy nhất* sau khi quá trình huấn luyện và lựa chọn mô hình (dựa trên tập validation) đã hoàn tất. Tập này cung cấp một ước lượng không thiên vị về hiệu suất của mô hình cuối cùng trên dữ liệu hoàn toàn mới trong thế giới thực.

2.2.2.2. K-Fold Cross-Validation

Khi dữ liệu hạn chế, K-Fold Cross-Validation giúp đánh giá mô hình một cách tin cậy hơn:

1. Chia dữ liệu thành K phần bằng nhau
2. Huấn luyện mô hình K lần, mỗi lần dùng K-1 phần làm tập huấn luyện và 1 phần còn lại làm tập validation
3. Tính trung bình hiệu suất trên K lần chạy

```
# K-Fold Cross-Validation với Keras
from sklearn.model_selection import KFold

def build_model():
    return tf.keras.Sequential([...]) # định nghĩa mô hình

k_fold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = []

for train_idx, val_idx in k_fold.split(X):
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    model = build_model()
    model.fit(X_train, y_train, epochs=10, validation_data=(X_val,
y_val))
    scores.append(model.evaluate(X_val, y_val)[1]) # lưu accuracy

print(f"Average accuracy: {np.mean(scores)}")
```

2.2.2.3. Số liệu Đánh giá (Metrics)

Các số liệu khác nhau đánh giá các khía cạnh khác nhau của hiệu suất mô hình:

- **Regression Metrics:**
 - ♦ **Mean Absolute Error (MAE):** Trung bình độ lệch tuyệt đối
 - ♦ **Mean Squared Error (MSE):** Trung bình bình phương sai số
 - ♦ **Root Mean Squared Error (RMSE):** Căn bậc hai của MSE
 - ♦ **R² Score:** Tỷ lệ phương sai được giải thích bởi mô hình
- **Classification Metrics:**
 - ♦ **Accuracy:** Tỷ lệ dự đoán đúng
 - ♦ **Precision:** Tỷ lệ dự đoán dương tính đúng / tổng dự đoán dương tính
 - ♦ **Recall:** Tỷ lệ dự đoán dương tính đúng / tổng dương tính thật
 - ♦ **AUC-ROC:** Diện tích dưới đường cong ROC

```
# Định nghĩa nhiều metrics trong Keras
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy', tf.keras.metrics.Precision(),
tf.keras.metrics.Recall()]
)
```

2.3. Xử lý Dữ liệu và Kỹ thuật Điều chuẩn Phổ biến

Dữ liệu thô hiếm khi có thể được đưa trực tiếp vào mạng nơ-ron. Nó cần được tiền xử lý cẩn thận. Đồng thời, việc ngăn chặn mô hình học quá khớp (overfitting) - hiện tượng mô hình hoạt động tốt trên dữ liệu huấn luyện nhưng kém trên dữ liệu mới - là một thách thức cốt lõi trong học sâu.

2.3.1. Xử lý Dữ liệu

2.3.1.1. Vector hóa (Vectorization)

Vector hóa là quá trình chuyển đổi dữ liệu phi số (như văn bản, hình ảnh) thành vector số, cho phép mô hình neural network xử lý:

- **One-hot Encoding:** Biểu diễn mỗi giá trị hạng mục bằng một vector nhị phân có độ dài bằng số lượng hạng mục duy nhất, trong đó chỉ có một phần tử bằng 1 tại vị trí tương ứng với hạng mục đó, còn lại là 0. Ví dụ: màu ["đỏ", "xanh", "vàng"] -> "đỏ" = [1, 0, 0], "xanh" = [0, 1, 0]. Đơn giản nhưng có thể tạo ra vector rất lớn và thừa nếu số lượng hạng mục nhiều.

```
• # One-hot encoding với Pandas/Scikit-learn
• one_hot = pd.get_dummies(categorical_data)
• # Hoặc
• from sklearn.preprocessing import OneHotEncoder
• encoder = OneHotEncoder()
• one_hot = encoder.fit_transform(categorical_data.reshape(-1, 1))
```

- **Multi-hot Encoding:** Tương tự one-hot, nhưng cho phép nhiều phần tử bằng 1 trong vector, dùng khi một mẫu có thể thuộc về nhiều hạng mục cùng lúc (ví dụ: gán nhiều thẻ thể loại cho một bộ phim).
- **TF-IDF (Term Frequency-Inverse Document Frequency):** Kỹ thuật phổ biến để vector hóa văn bản. Nó gán trọng số cho mỗi từ trong một tài liệu dựa trên tần suất xuất hiện của từ đó trong tài liệu (TF) và tần suất xuất hiện của từ đó trong toàn bộ kho văn bản (IDF). Các từ xuất hiện thường xuyên trong một tài liệu nhưng hiếm trong các tài liệu khác sẽ có trọng số TF-IDF cao.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(text_documents)
```

2.3.1.2. Chuẩn hóa Dữ liệu (Data Normalization)

Đưa các đặc trưng (features) có thang đo hoặc đơn vị khác nhau về một phạm vi chung. Điều này rất quan trọng vì các thuật toán tối ưu hóa dựa trên gradient thường hoạt động tốt hơn khi các đặc trưng có cùng thang đo.

- **Min-Max Scaling:** Biến đổi tuyến tính các giá trị để nằm trong một khoảng xác định, thường là [0, 1] hoặc [-1, 1]. Công thức: $X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$. Nhạy cảm với outliers.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

- **Standardization (Z-score):** Biến đổi dữ liệu sao cho có trung bình (mean) bằng 0 và độ lệch chuẩn (standard deviation) bằng 1.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2.3.1.3. Padding và Masking

Các kỹ thuật cần thiết khi xử lý dữ liệu tuần tự có độ dài thay đổi, phổ biến trong Xử lý Ngôn ngữ Tự nhiên (NLP).

- **Padding:** Thêm giá trị đặc biệt (thường là 0) để đưa tất cả chuỗi về cùng độ dài


```
# Padding trong Keras
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_sequences = pad_sequences(sequences, maxlen=100, padding='post')
```

- **Masking:** Sau khi padding, cần có cơ chế để thông báo cho các lớp mạng nơ-ron (đặc biệt là các lớp RNN hoặc lớp Attention) biết rằng các giá trị padding này không chứa thông tin thực sự và nên được bỏ qua trong quá trình tính toán (ví dụ: không tính toán attention score cho các vị trí padding).

```
# Masking trong Keras
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
input_length=max_length),
    tf.keras.layers.Masking(mask_value=0.0),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

2.3.2. Các Kỹ thuật Điều chuẩn Cơ bản

Là các phương pháp được thêm vào quá trình huấn luyện để giảm overfitting, giúp mô hình tổng quát hóa tốt hơn trên dữ liệu mới. Overfitting xảy ra khi mô hình quá phức tạp và bắt đầu học cả nhiễu trong dữ liệu huấn luyện thay vì chỉ học các mẫu cơ bản.

2.3.2.1. Giảm Kích thước Mạng

Cách đơn giản nhất để chống overfitting là làm cho mô hình có ít "khả năng học" hơn. Điều này có thể đạt được bằng cách:

- Giảm số lượng lớp ẩn.
- Giảm số lượng nơ-ron (units/filters) trong mỗi lớp.
- Tìm sự cân bằng giữa mô hình quá nhỏ (underfitting - không học được đủ các mẫu) và mô hình quá lớn (overfitting).

2.3.2.2. Dropout

Một trong những kỹ thuật điều chuẩn mạnh mẽ và phổ biến nhất. Trong mỗi bước huấn luyện, dropout ngẫu nhiên "vô hiệu hóa" (đặt đầu ra của chúng bằng 0) một tỷ lệ nơ-ron được chọn trong một lớp. Các nơ-ron bị tắt khác nhau ở mỗi bước.

- Điều này buộc các nơ-ron còn lại phải học các đặc trưng mạnh mẽ hơn và không phụ thuộc quá nhiều vào bất kỳ nơ-ron cụ thể nào.
- Có thể xem dropout như một cách huấn luyện đồng thời một tập hợp lớn các mạng con khác nhau (ensemble) và lấy trung bình kết quả của chúng trong quá trình dự đoán (inference). Tỷ lệ dropout (p) thường nằm trong khoảng 0.2 đến 0.5.

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5), # Tắt 50% nơ-ron
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3), # Tắt 30% nơ-ron
    tf.keras.layers.Dense(10, activation='softmax')
])
```

2.3.2.3. Weight Regularization

Thêm một thành phần "phạt" (penalty term) vào hàm mất mát, phạt các trọng số có giá trị lớn. Ý tưởng là các mô hình đơn giản hơn (có trọng số nhỏ hơn) thường tổng quát hóa tốt hơn.

- **L1 Regularization:** Thêm vào loss một thành phần tỷ lệ với tổng giá trị tuyệt đối của các trọng số ($\lambda \sum |w|$). L1 có xu hướng đẩy một số trọng số về chính xác bằng 0, dẫn đến mô hình thưa (sparse model) và có thể được sử dụng để lựa chọn đặc trưng.
- **L2 Regularization:** Thêm vào loss một thành phần tỷ lệ với tổng bình phương của các trọng số ($\lambda \sum w^2$). L2 có xu hướng làm cho các trọng số nhỏ hơn nhưng hiếm khi bằng 0. Đây là dạng điều chuẩn trọng số phổ biến nhất trong học sâu. λ là siêu tham số kiểm soát cường độ điều chuẩn.

```
from tensorflow.keras.regularizers import l1, l2, l1_l2

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu',
        kernel_regularizer=l2(0.001)), #  $\lambda = 0,001$ 
    tf.keras.layers.Dense(64, activation='relu',
        kernel_regularizer=l2(0.001)),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

2.3.2.4. Early Stopping

Một kỹ thuật điều chuẩn dựa trên việc theo dõi hiệu suất của mô hình trên tập kiểm định (validation set) trong quá trình huấn luyện.

- Huấn luyện được dừng lại khi chỉ số hiệu suất trên tập kiểm định (thường là validation loss) ngừng cải thiện hoặc bắt đầu tệ đi trong một số lượng epoch nhất định (gọi là *patience*).
- Trọng số của mô hình tại thời điểm có hiệu suất tốt nhất trên tập kiểm định thường được lưu lại và sử dụng làm mô hình cuối cùng.
- Đây là một cách hiệu quả và trực quan để ngăn mô hình tiếp tục huấn luyện khi nó đã bắt đầu overfitting vào dữ liệu huấn luyện.

```

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,          # Số epoch chờ đợi
    restore_best_weights=True # Khôi phục trọng số tốt nhất
)

model.fit(
    X_train, y_train,
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping]
)

```

2.3.2.5. Data Augmentation

Tạo ra các mẫu dữ liệu huấn luyện mới từ các mẫu hiện có bằng cách áp dụng các phép biến đổi ngẫu nhiên nhưng hợp lý về mặt ngữ nghĩa.

- Đối với hình ảnh: xoay, dịch chuyển, cắt, thu phóng, lật ngang/dọc, thay đổi độ sáng/tương phản/màu sắc.
- Đối với âm thanh: thêm nhiễu nền, thay đổi tốc độ, dịch chuyển cao độ.
- Đối với văn bản: thay thế từ bằng từ đồng nghĩa, back-translation (dịch sang ngôn ngữ khác rồi dịch ngược lại), thêm/xóa từ ngẫu nhiên.
- Data augmentation giúp mô hình học được các đặc trưng bất biến với các biến đổi này và tăng kích thước hiệu quả của tập dữ liệu huấn luyện, từ đó cải thiện khả năng tổng quát hóa.

```

# Data augmentation cho hình ảnh trong Keras
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.1),
    tf.keras.layers.RandomZoom(0.1),
    tf.keras.layers.RandomTranslation(0.1, 0.1)
])

model = tf.keras.Sequential([
    data_augmentation,
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    # Các lớp tiếp theo...
])

```

2.4. Các Kiến trúc Nâng cao cho Dữ liệu Có cấu trúc

Ngoài MLP, có các kiến trúc mạng nơ-ron được thiết kế đặc biệt để khai thác hiệu quả các loại cấu trúc dữ liệu cụ thể, như cấu trúc không gian trong ảnh hoặc cấu trúc tuần tự trong văn bản.

2.4.1. Mạng Nơ-ron Tích chập (Convolutional Neural Networks - CNNs)

Là kiến trúc thống trị trong lĩnh vực thị giác máy tính (computer vision) và cũng được áp dụng thành công cho các loại dữ liệu dạng lưới khác. Ý tưởng cốt lõi là sử dụng các lớp tích chập để tự động học các hệ thống phân cấp đặc trưng (hierarchies of features) từ dữ liệu đầu vào. Các lớp đầu tiên học các đặc trưng đơn giản (cạnh, góc), các lớp sâu hơn kết hợp chúng để học các đặc trưng phức tạp hơn (kết cấu, bộ phận, đối tượng).

2.4.1.1. Các Lớp Chính

- **Convolutional Layer (Lớp Tích chập):** Thành phần trung tâm. Áp dụng một tập hợp các bộ lọc (kernels) có thể học được lên các vùng cục bộ của đầu vào. Mỗi bộ lọc chuyên phát hiện một loại đặc trưng cụ thể. Đầu ra của lớp tích chập là các bản đồ đặc trưng (feature maps), mỗi bản đồ tương ứng với một bộ lọc. Các tham số quan trọng bao gồm kích thước bộ lọc (kernel size), số lượng bộ lọc (number of filters), sải bước (stride), và đệm (padding).
- **Pooling Layer (Lớp Gộp):** Thường được chèn giữa các lớp tích chập liên tiếp. Mục đích chính là giảm dần kích thước không gian (chiều rộng, chiều cao) của các bản đồ đặc trưng, giúp giảm số lượng tham số, kiểm soát overfitting và làm cho biểu diễn đặc trưng trở nên bất biến hơn với các thay đổi nhỏ về vị trí. MaxPooling (lấy giá trị cực đại) và AveragePooling (lấy giá trị trung bình) là hai loại phổ biến.

```
# Mô hình CNN đơn giản trong Keras
model = tf.keras.Sequential([
    # Lớp đầu vào: 32x32 pixel, 3 kênh màu (RGB)
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.GlobalAveragePooling2D(),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

2.4.1.2. Ứng dụng của CNN

CNN được ứng dụng trong nhiều lĩnh vực như: phân loại hình ảnh (image classification), phát hiện vật thể (object detection), phân đoạn ảnh (image segmentation), nhận dạng khuôn mặt, phân tích ảnh y tế, xử lý video, và thậm chí cả phân loại văn bản (áp dụng tích chập 1D trên chuỗi từ embeddings).

2.4.2. Mạng Nơ-ron Hồi quy (Recurrent Neural Networks - RNNs)

RNN Được thiết kế đặc biệt để xử lý dữ liệu tuần tự (sequential data), nơi thông tin ở các bước trước đó có thể liên quan đến việc hiểu hoặc dự đoán ở các bước sau. RNN duy trì một "trạng thái ẩn" (hidden state) được cập nhật ở mỗi bước thời gian, mang thông tin từ quá khứ. Trạng thái ẩn này cùng với đầu vào hiện tại được sử dụng để tính toán đầu ra và trạng thái ẩn cho bước tiếp theo.

2.4.2.1. Các Lớp Chính

- **SimpleRNN:** Kiến trúc RNN cơ bản nhất. Tuy nhiên, nó gặp phải vấn đề gradient biến mất hoặc bùng nổ (vanishing/exploding gradients) khi xử lý các chuỗi dài, khiến nó khó học được các phụ thuộc xa (long-range dependencies).
- **LSTM (Long Short-Term Memory):** Một loại RNN phức tạp hơn được thiết kế để giải quyết vấn đề của SimpleRNN. LSTM có một cấu trúc "ô nhớ" (memory cell) và ba "cổng" (gates) - cổng vào (input gate), cổng quên (forget gate), và cổng ra (output gate). Các cổng này điều khiển luồng thông tin vào, ra và lưu trữ trong ô nhớ, cho phép LSTM học và nhớ thông tin qua các khoảng thời gian dài.
- **GRU (Gated Recurrent Unit):** Một biến thể của LSTM được đề xuất sau đó, có cấu trúc cổng đơn giản hơn (chỉ có cổng cập nhật - update gate và cổng đặt lại - reset gate). GRU thường có hiệu suất tương đương LSTM trên nhiều tác vụ nhưng có ít tham số hơn và tính toán nhanh hơn một chút.

```
# Mô hình LSTM đơn giản cho phân loại văn bản
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.LSTM(64, return_sequences=False),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

2.4.2.2. Xếp chồng RNN (Stacking RNNs)

Đặt nhiều lớp RNN (ví dụ: LSTM hoặc GRU) lên nhau. Đầu ra của lớp RNN thứ nhất (dưới dạng một chuỗi các trạng thái ẩn) trở thành đầu vào cho lớp RNN thứ hai, v.v. Điều này cho phép mạng học các biểu diễn tuần tự phức tạp hơn ở các cấp độ trừu tượng khác nhau. Để xếp chồng, các lớp RNN ở dưới (trừ lớp cuối cùng nếu chỉ cần đầu ra cuối) cần được cấu hình để trả về toàn bộ chuỗi đầu ra

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.LSTM(128, return_sequences=True), # Trả về chuỗi
    đầu ra
    tf.keras.layers.LSTM(64), # Chỉ lấy đầu ra cuối cùng
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

2.4.2.3. RNN Hai chiều (Bidirectional RNNs)

Để hiểu ngữ cảnh của một từ hoặc một điểm dữ liệu trong chuỗi, thông tin từ cả quá khứ (phía trước) và tương lai (phía sau) đều quan trọng. RNN hai chiều giải quyết vấn đề này bằng cách sử dụng hai lớp RNN riêng biệt: một lớp xử lý chuỗi theo thứ tự từ trái sang phải, và lớp kia xử lý theo thứ tự ngược lại từ phải sang trái. Đầu ra của hai lớp này tại mỗi bước thời gian thường được kết hợp (ví dụ: cộng hoặc ghép lại) để tạo ra biểu diễn cuối cùng, nắm bắt ngữ cảnh từ cả hai phía.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

2.4.2.4. Recurrent Dropout

Dropout đặc biệt cho RNN, Áp dụng kỹ thuật dropout cho các kết nối bên trong các đơn vị RNN (LSTM/GRU). Cần có cách tiếp cận đặc biệt (ví dụ: áp dụng cùng một dropout mask cho tất cả các bước thời gian trong một chuỗi) để không làm gián đoạn việc lan truyền thông tin qua thời gian, vốn là cốt lõi của RNN. Giúp chống overfitting hiệu quả trong các mô hình RNN.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

2.4.2.5. Ứng dụng của RNN

Ứng dụng phổ biến của RNN là xử lý ngôn ngữ tự nhiên (mô hình ngôn ngữ, dịch máy, phân loại văn bản, nhận dạng thực thể tên, phân tích cảm xúc), nhận dạng giọng nói, tổng hợp giọng nói, dự báo chuỗi thời gian (thời tiết, giá cổ phiếu), phân tích dữ liệu sinh học (DNA, protein).

2.5. Các Kỹ thuật Kiến trúc và Chuẩn hóa Nâng cao

2.5.1. Kết nối Thặng dư (Residual Connections)

- Một kỹ thuật kiến trúc đột phá được giới thiệu trong mạng ResNet (Residual Network), cho phép huấn luyện các mạng nơ-ron cực sâu (hàng trăm, thậm chí hàng nghìn lớp) mà không gặp vấn đề suy giảm hiệu suất (degradation problem) và gradient biến mất.
- Ý tưởng chính là thêm các "đường tắt" (shortcut connections) hoặc "kết nối thặng dư" (residual connections) cho phép tín hiệu đầu vào của một khối (block) được cộng trực tiếp vào đầu ra của khối đó (thường là sau các lớp tích chập và trước hàm kích hoạt cuối cùng của khối).
- Thay vì bắt các lớp trong khối phải học trực tiếp ánh xạ mong muốn $H(x)$, chúng được học phần "thặng dư" (residual) $F(x) = H(x) - x$. Sau đó, đầu ra là $F(x) + x$. Việc học phần thặng dư này thường dễ dàng hơn, đặc biệt khi ánh xạ tối ưu gần với hàm đồng nhất (identity mapping, tức là $H(x) = x$), khi đó $F(x)$ chỉ cần học giá trị gần bằng 0.
- Kết nối thặng dư đảm bảo rằng gradient có thể lan truyền trực tiếp qua các đường tắt, giúp giảm thiểu vấn đề gradient biến mất trong các mạng rất sâu.

```
def residual_block(x, filters, kernel_size=3):  
    # Lưu đầu vào  
    residual = x  
  
    # Xử lý qua các lớp  
    y = tf.keras.layers.Conv2D(filters, kernel_size, padding='same')(x)  
    y = tf.keras.layers.BatchNormalization()(y)  
    y = tf.keras.layers.Activation('relu')(y)  
  
    y = tf.keras.layers.Conv2D(filters, kernel_size, padding='same')(y)  
    y = tf.keras.layers.BatchNormalization()(y)  
  
    # Cộng với đầu vào ban đầu  
    output = tf.keras.layers.add([residual, y])  
    output = tf.keras.layers.Activation('relu')(output)  
  
    return output
```

2.5.2. Chuẩn hóa Lô (Batch Normalization)

- Một kỹ thuật chuẩn hóa rất phổ biến và hiệu quả, thường được thêm vào sau lớp tích chập hoặc lớp dense và trước hàm kích hoạt.
- BatchNorm chuẩn hóa đầu ra của lớp trước đó bằng cách trừ đi trung bình và chia cho độ lệch chuẩn của lô (mini-batch) hiện tại. Sau đó, nó thực hiện một phép biến đổi

tuyến tính với hai tham số có thể học được (γ và β) để khôi phục khả năng biểu diễn của mạng.

- Lợi ích chính:
 - ♦ Giảm sự thay đổi phân phối đầu vào của các lớp (internal covariate shift), giúp quá trình huấn luyện ổn định hơn.
 - ♦ Cho phép sử dụng tốc độ học (learning rate) cao hơn, tăng tốc độ hội tụ.
 - ♦ Có tác dụng điều chuẩn nhẹ, giảm nhu cầu sử dụng các kỹ thuật điều chuẩn khác như Dropout.
- Nhược điểm: Hiệu suất phụ thuộc vào kích thước lô (batch size) - hoạt động kém với lô nhỏ; có sự khác biệt giữa hành vi lúc huấn luyện và lúc dự đoán (inference).

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),

    tf.keras.layers.Conv2D(64, 3, padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),

    # Các lớp tiếp theo...
])
```

2.5.3. Chuẩn hóa Lớp (Layer Normalization)

- Một kỹ thuật chuẩn hóa thay thế cho BatchNorm, đặc biệt hiệu quả trong các mạng nơ-ron hồi quy (RNN) và Transformer.
- Khác với BatchNorm chuẩn hóa trên chiều lô, LayerNorm chuẩn hóa trên chiều đặc trưng (features) của *từng mẫu dữ liệu riêng lẻ* trong lô. Nó tính toán trung bình và phương sai trên tất cả các nơ-ron trong cùng một lớp cho một mẫu cụ thể.
- Ưu điểm:
 - Hoạt động không phụ thuộc vào kích thước lô, hiệu quả ngay cả với batch size = 1.
 - Hành vi giống nhau trong cả quá trình huấn luyện và dự đoán.
 - Rất phù hợp cho RNN và Transformer nơi độ dài chuỗi có thể thay đổi.

```
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```


2.5.4. Tích chập Tách biệt theo Chiều sâu (Depthwise Separable Convolutions)

- Là một dạng tích chập hiệu quả hơn về mặt tính toán so với tích chập 2D tiêu chuẩn, giúp giảm đáng kể số lượng tham số và phép toán (FLOPs) mà vẫn duy trì hiệu suất tương đương hoặc thậm chí tốt hơn. Được sử dụng rộng rãi trong các kiến trúc mạng nhẹ (lightweight architectures) như MobileNets, Xception, EfficientNets, phù hợp cho các thiết bị di động và nhúng.
- Nó tách một phép tích chập tiêu chuẩn thành hai bước riêng biệt:
 1. **Depthwise Convolution:** Áp dụng một bộ lọc tích chập riêng biệt cho từng kênh (channel) của đầu vào một cách độc lập. Nếu đầu vào có C_{in} kênh, bước này sẽ sử dụng C_{in} bộ lọc (thường là 3×3 hoặc 5×5). Nó chỉ thực hiện lọc không gian (spatial filtering) mà không kết hợp thông tin giữa các kênh.
 2. **Pointwise Convolution:** Áp dụng tích chập 1×1 (bộ lọc có kích thước 1×1) lên đầu ra của bước depthwise convolution. Bước này thực hiện kết hợp thông tin tuyến tính giữa các kênh (channel combination) để tạo ra các đặc trưng mới.
- Sự kết hợp của hai bước này đạt được hiệu quả tương tự tích chập tiêu chuẩn nhưng với chi phí tính toán thấp hơn nhiều.

```
model = tf.keras.Sequential([
    tf.keras.layers.SeparableConv2D(32, 3, padding='same',
activation='relu'),
    tf.keras.layers.SeparableConv2D(64, 3, padding='same',
activation='relu'),
    # Các lớp tiếp theo...
])
```

2.6. Kỹ thuật Chuyên sâu cho Xử lý Ngôn ngữ Tự nhiên

Xử lý Ngôn ngữ Tự nhiên (NLP) là một lĩnh vực trọng tâm của AI, tập trung vào việc cho phép máy tính hiểu, diễn giải và tạo ra ngôn ngữ của con người. Học sâu đã mang lại những tiến bộ vượt bậc cho NLP, và các kỹ thuật dưới đây là nền tảng của nhiều hệ thống NLP hiện đại.

2.6.1. Word Embeddings

- **Khái niệm cốt lõi:** Máy tính không thể hiểu trực tiếp các từ dưới dạng văn bản. Word embeddings là kỹ thuật biểu diễn các từ (hoặc đôi khi là các đơn vị nhỏ hơn như ký tự hoặc tiền tố/hậu tố) dưới dạng các vector số thực dày đặc (dense vectors) trong một không gian nhiều chiều. Khác với các biểu diễn thưa như one-hot encoding (vector rất dài, chủ yếu là số 0), word embeddings có số chiều tương đối thấp (ví dụ: 50 đến 1000 chiều) và mỗi chiều thể hiện một khía cạnh tiềm ẩn nào đó của ngữ nghĩa hoặc cú pháp.

- **Nắm bắt ngữ nghĩa:** Điểm mạnh chính của word embeddings là khả năng nắm bắt mối quan hệ ngữ nghĩa và cú pháp giữa các từ. Các từ có ý nghĩa tương tự hoặc

thường được sử dụng trong các ngữ cảnh giống nhau (ví dụ: "chó" và "mèo", "đi bộ" và "chạy") sẽ có các vector embedding gần nhau trong không gian vector (theo một độ đo khoảng cách nào đó như cosine similarity). Các mối quan hệ tương tự cũng có thể được biểu diễn dưới dạng các phép toán vector (ví dụ: $\text{vector}(\text{"vua"}) - \text{vector}(\text{"đàn ông"}) + \text{vector}(\text{"phụ nữ"}) \approx \text{vector}(\text{"nữ hoàng"})$).

- **Cách tạo ra:** Word embeddings có thể được học theo hai cách chính:
 - **Học từ đầu (From Scratch):** Một lớp embedding được thêm vào đầu mạng nơ-ron NLP, và các vector embedding được khởi tạo ngẫu nhiên rồi được tinh chỉnh cùng với các trọng số khác của mạng trong quá trình huấn luyện trên tác vụ cụ thể. Cách này phù hợp khi có đủ dữ liệu cho tác vụ đó.
 - **Sử dụng Pre-trained Embeddings:** Tận dụng các bộ embedding đã được huấn luyện trước trên các kho văn bản cực lớn (như Wikipedia, tin tức). Các bộ embedding này chứa kiến thức ngôn ngữ tổng quát phong phú. Các mô hình phổ biến để tạo pre-trained embeddings bao gồm Word2Vec (sử dụng các thuật toán CBOW hoặc Skip-gram để dự đoán từ dựa trên ngữ cảnh hoặc ngược lại), GloVe (dựa trên thống kê đồng xuất hiện của các từ), và FastText (mở rộng Word2Vec bằng cách xem xét cả các đơn vị nhỏ hơn từ - character n-grams, giúp xử lý từ hiếm tốt hơn). Sử dụng pre-trained embeddings thường giúp cải thiện hiệu suất, đặc biệt khi dữ liệu cho tác vụ đích bị hạn chế.

```
# Sử dụng lớp Embedding trong Keras
embedding_layer = tf.keras.layers.Embedding(
    input_dim=vocab_size,      # Kích thước từ điển
    output_dim=300,            # Kích thước embedding
    input_length=max_length    # Độ dài chuỗi đầu vào
)
```

2.6.2. Attention và Self-Attention

Vấn đề của các mô hình tuần tự truyền thống: Các mô hình như RNN, LSTM, GRU xử lý chuỗi tuần tự từng bước một và cố gắng nén thông tin của toàn bộ chuỗi đầu vào vào một vector trạng thái ẩn có kích thước cố định. Điều này tạo ra một "nút thắt cổ chai thông tin", đặc biệt với các chuỗi dài, khiến mô hình khó nhớ các chi tiết quan trọng từ đầu chuỗi khi xử lý phần cuối chuỗi.

Cơ chế Attention: Được giới thiệu để giải quyết vấn đề này, đặc biệt trong các mô hình sequence-to-sequence. Thay vì dựa vào một vector ngữ cảnh cố định duy nhất, attention cho phép mô hình (thường là decoder) "nhìn lại" toàn bộ chuỗi đầu vào (hoặc các trạng thái ẩn của encoder) ở mỗi bước tạo ra đầu ra. Nó tính toán một tập "trọng số tập trung" (attention weights) cho mỗi phần tử đầu vào, cho biết mức độ liên quan của phần tử đó đối với việc tạo ra phần tử đầu ra hiện tại. Các phần tử đầu vào có

trọng số cao hơn sẽ đóng góp nhiều hơn vào việc tính toán đầu ra. Điều này cho phép mô hình tập trung linh hoạt vào các phần thông tin cần thiết nhất tại mỗi thời điểm.

Self-Attention: Là một biến thể đặc biệt và mạnh mẽ của attention, nơi cơ chế tập trung được áp dụng *bên trong cùng một chuỗi*. Thay vì tính toán sự liên quan giữa hai chuỗi khác nhau (như encoder và decoder), self-attention tính toán mức độ liên quan giữa các phần tử (ví dụ: các từ) khác nhau trong *cùng một chuỗi*. Đối với mỗi từ trong câu, self-attention tính toán một trọng số tập trung cho tất cả các từ khác (bao gồm cả chính nó) trong câu đó. Điều này cho phép mô hình xây dựng các biểu diễn nhạy cảm với ngữ cảnh cho từng từ, dựa trên mối quan hệ của nó với các từ khác trong câu, bất kể khoảng cách giữa chúng. Self-attention rất hiệu quả trong việc nắm bắt các phụ thuộc ngữ pháp và ngữ nghĩa phức tạp, kể cả các phụ thuộc xa. Nó là thành phần trung tâm của kiến trúc Transformer.

2.6.3. Kiến trúc Transformer

Một kiến trúc mạng nơ-ron được giới thiệu vào năm 2017 ("Attention Is All You Need"), đã cách mạng hóa lĩnh vực NLP và trở thành nền tảng cho hầu hết các mô hình ngôn ngữ lớn (Large Language Models - LLMs) hiện đại như BERT, GPT, T5, v.v. Điểm đột phá chính của Transformer là nó loại bỏ hoàn toàn các lớp hồi quy (RNN/LSTM/GRU) và chỉ dựa vào cơ chế self-attention để xử lý các mối quan hệ trong chuỗi.

- **Ưu điểm chính:**

- ♦ *Song song hóa:* Việc tính toán self-attention cho các vị trí khác nhau trong chuỗi có thể được thực hiện độc lập và song song, giúp tận dụng hiệu quả phần cứng hiện đại (GPU/TPU) và tăng tốc đáng kể quá trình huấn luyện so với tính toán tuần tự của RNN.
- ♦ *Nắm bắt phụ thuộc xa:* Đường đi thông tin giữa hai vị trí bất kỳ trong chuỗi thông qua self-attention là trực tiếp và có độ dài không đổi ($O(1)$), giúp mô hình dễ dàng nắm bắt các phụ thuộc xa hơn so với RNN (nơi đường đi thông tin dài hơn và gradient có thể bị suy yếu).

- **Các thành phần chính:**

- *Multi-Head Self-Attention:* Thay vì chỉ tính toán self-attention một lần, Transformer thực hiện nó nhiều lần song song với các bộ trọng số khác nhau (gọi là các "đầu" - heads). Mỗi "đầu" có thể học cách tập trung vào các loại mối quan hệ hoặc các khía cạnh khác nhau của chuỗi. Kết quả từ các đầu sau đó được kết hợp lại.
- *Position-wise Feed-Forward Networks:* Sau lớp self-attention, đầu ra cho mỗi vị trí được xử lý độc lập bởi một mạng nơ-ron feed-forward nhỏ (thường là hai lớp Dense với hàm kích hoạt ReLU ở giữa). Lớp này giúp xử lý và biến đổi thông tin tại từng vị trí.
- *Residual Connections & Layer Normalization:* Giống như trong ResNet, các kết nối thẳng dư được sử dụng xung quanh mỗi khối con (self-attention và feed-

forward) để giúp gradient lan truyền tốt hơn và cho phép xây dựng mạng sâu hơn. Chuẩn hóa lớp (Layer Normalization) được áp dụng sau mỗi kết nối thẳng dư để ổn định quá trình huấn luyện.

- *Positional Encoding*: Vì self-attention vốn không xem xét thứ tự của các phần tử trong chuỗi (permutation-invariant), Transformer cần một cách để đưa thông tin vị trí vào. Positional encoding là các vector có cùng chiều với embedding, được tạo ra dựa trên vị trí tuyệt đối hoặc tương đối của token trong chuỗi (thường dùng các hàm sin và cosin với tần số khác nhau). Các vector này được cộng (hoặc ghép) vào input embeddings để cung cấp cho mô hình thông tin về thứ tự.

- **Cấu trúc Encoder-Decoder**: Kiến trúc Transformer gốc bao gồm một chuỗi các lớp Encoder giống hệt nhau và một chuỗi các lớp Decoder giống hệt nhau. Encoder xử lý chuỗi đầu vào, và Decoder tạo ra chuỗi đầu ra một cách tự hồi quy, sử dụng cả self-attention trên đầu ra đã tạo và cross-attention để tập trung vào đầu ra của Encoder. Các mô hình sau này như BERT chỉ sử dụng phần Encoder, còn GPT chỉ sử dụng phần Decoder.

2.6.4. Học Sequence-to-Sequence

- **Khung mô hình tổng quát**: Seq2Seq là một lớp các mô hình được thiết kế để giải quyết các bài toán mà đầu vào là một chuỗi và đầu ra cũng là một chuỗi, nhưng độ dài của hai chuỗi có thể khác nhau. Đây là một mô hình rất linh hoạt và có nhiều ứng dụng.

- **Kiến trúc Encoder-Decoder**: Cốt lõi của mô hình Seq2Seq là kiến trúc encoder-decoder:

- *Encoder (Bộ mã hóa)*: Xử lý toàn bộ chuỗi đầu vào và tạo ra một biểu diễn tổng hợp của nó, thường là một vector trạng thái ẩn cuối cùng hoặc một tập hợp các trạng thái ẩn (trong trường hợp có attention). Mục tiêu của encoder là "hiểu" ý nghĩa và thông tin quan trọng của chuỗi đầu vào.
- *Decoder (Bộ giải mã)*: Nhận biểu diễn từ encoder và tạo ra chuỗi đầu ra từng phần tử một. Ở mỗi bước, decoder dự đoán phần tử tiếp theo dựa trên biểu diễn của encoder và các phần tử đã được tạo ra trước đó. Quá trình này tiếp tục cho đến khi một token kết thúc chuỗi đặc biệt được tạo ra.

- **Vai trò của Attention**: Cơ chế attention đóng vai trò quan trọng trong việc cải thiện hiệu suất của các mô hình Seq2Seq. Nó cho phép decoder tập trung vào các phần cụ thể và liên quan nhất của chuỗi đầu vào khi tạo ra từng phần tử của chuỗi đầu ra, thay vì chỉ dựa vào một vector ngữ cảnh cố định.

- **Triển khai**: Ban đầu, encoder và decoder thường được xây dựng bằng các mạng RNN như LSTM hoặc GRU. Tuy nhiên, với sự thành công của Transformer, việc sử dụng kiến trúc dựa trên attention (như Transformer) cho cả encoder và decoder đã trở nên phổ biến và thường mang lại kết quả tốt hơn.

- **Ứng dụng:** Dịch máy (machine translation), tóm tắt văn bản (text summarization), trả lời câu hỏi (question answering), tạo chatbot đối thoại, chuyển đổi giọng nói thành văn bản (ASR), và nhiều tác vụ khác liên quan đến việc biến đổi chuỗi.

CHƯƠNG 3: ỨNG DỤNG HỌC SÂU TRONG PHÂN LOẠI HÌNH ẢNH

3.1. Huấn luyện mô hình

3.1.1. Mô hình phân loại chim (Fine-tuning ResNet50)

3.1.1.1. Mục tiêu bài toán

Mục tiêu của bài toán này là xây dựng một mô hình học sâu có khả năng **phân loại các loài chim khác nhau** dựa trên ảnh đầu vào. Mô hình sẽ được huấn luyện để nhận diện và phân loại chính xác 6 loài chim có trong bộ dữ liệu.

3.1.1.2. Tập dữ liệu

Nguồn dữ liệu: [Bird Species Dataset](#)

Định dạng ảnh: Ảnh màu (RGB).

Kích thước ảnh đầu vào cho mô hình: (224, 224) pixel.

Số lớp (loài chim): 6.

Nhãn lớp và chỉ số tương ứng:

- 'american goldfinch': 0
- 'barn owl': 1
- 'carmine bee-eater': 2
- 'downy woodpecker': 3
- 'emperor penguin': 4
- 'flamingo': 5

3.1.1.3. Thực hiện tiền xử lý và chia tập huấn luyện/kiểm tra

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

image_size = (224, 224)
input_shape_resnet = (image_size[0], image_size[1], 3)
batch_size = 32
data_dir = "/content/dataset3"
validation_split_ratio = 0.2

train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input,
    validation_split=validation_split_ratio
)

val_datagen = ImageDataGenerator(
```

```

        preprocessing_function=preprocess_input,
        validation_split=validation_split_ratio
    )

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='training'
)

val_generator = val_datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='validation',
    shuffle=False
)

num_classes = train_generator.num_classes
print(f"Số lớp (loài chim): {num_classes}")
print(f"Nhãn lớp: {train_generator.class_indices}")

class_labels = list(train_generator.class_indices.keys())

```

Tỷ lệ chia tập kiểm định (*validation_split_ratio*): 0.2 (20% dữ liệu được dùng cho tập kiểm định).

Tiền xử lý dữ liệu:

Hàm tiền xử lý chuyên biệt của ResNet50 (*preprocess_input*): Được áp dụng cho cả tập huấn luyện và kiểm định. Hàm này thực hiện các bước chuẩn hóa pixel cần thiết mà ResNet50 đã được huấn luyện cùng (ví dụ: chuyển đổi không gian màu, trừ giá trị trung bình kênh).

Tăng cường dữ liệu trên tập huấn luyện (*train_datagen*):

rotation_range=15: Xoay ảnh ngẫu nhiên trong khoảng ± 15 độ.

width_shift_range=0.1 và *height_shift_range=0.1*: Dịch chuyển ảnh theo chiều ngang và chiều dọc tối đa 10%.

shear_range=0.1: Áp dụng phép biến đổi cắt (shear).

zoom_range=0.1: Phóng to/thu nhỏ ảnh ngẫu nhiên trong khoảng 10%.

horizontal_flip=True: Lật ngang ảnh ngẫu nhiên.

fill_mode='nearest': Điền các pixel mới được tạo ra trong quá trình biến đổi bằng giá trị của pixel gần nhất.

→ Giúp mô hình học được tính tổng quát tốt hơn và tránh overfitting.

Chia tập dữ liệu:

- Sử dụng *ImageDataGenerator* với tham số *validation_split* để tự động chia dữ liệu từ *data_dir* thành tập huấn luyện và kiểm định.
- *train_generator* sử dụng *train_datagen* với *subset='training'*.
- *val_generator* sử dụng *val_datagen* (chỉ có *preprocess_input* và *validation_split*) với *subset='validation'* và *shuffle=False*.

– Tập huấn luyện (Training set): 652 ảnh thuộc 6 lớp.

– Tập kiểm định (Validation set): 159 ảnh thuộc 6 lớp.

3.1.1.4. Xây dựng mô hình CNN (Fine-tuning ResNet50)

Mô hình được xây dựng bằng cách sử dụng kỹ thuật học chuyên giao (Transfer Learning) với kiến trúc ResNet50 đã được huấn luyện trước trên bộ dữ liệu ImageNet.

```
base_model_resnet = ResNet50(weights='imagenet',
                               include_top=False,
                               input_shape=input_shape_resnet)

base_model_resnet.trainable = False

model = Sequential([
    base_model_resnet,
    GlobalAveragePooling2D(),
    Dense(1024, activation='relu'),
    Dense(512, activation='relu'),
    Dense(num_classes, activation='softmax')
])

learning_rate_head = 1e-4
optimizer_head = Adam(learning_rate=learning_rate_head)

model.compile(optimizer=optimizer_head,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Tải mô hình ResNet50 tiền huấn luyện (*base_model_resnet*):

- *weights='imagenet'*: Sử dụng trọng số đã huấn luyện trên ImageNet.
- *include_top=False*: Bỏ đi lớp phân loại fully-connected cuối cùng của ResNet50 (dành cho 1000 lớp của ImageNet).
- *input_shape=(224, 224, 3)*: Định nghĩa kích thước đầu vào là ảnh màu 224x224 pixel.

Đóng băng các lớp của base model:

`base_model_resnet.trainable = False`: Tất cả các lớp của ResNet50 gốc được đóng băng, trọng số của chúng sẽ không được cập nhật trong giai đoạn huấn luyện đầu tiên. Điều này giúp giữ lại các đặc trưng tổng quát mà ResNet50 đã học được.

Thêm các lớp tùy chỉnh (Custom Head) vào trên ResNet50:

- Lớp đầu tiên là `base_model_resnet` đã đóng băng.
- `GlobalAveragePooling2D()`: Giảm chiều dữ liệu từ output của ResNet50, chuyển feature map thành một vector cho mỗi ảnh.
- `Dense(1024, activation='relu')`: Lớp kết nối đầy đủ với 1024 neuron và hàm kích hoạt ReLU.
- `Dense(512, activation='relu')`: Lớp kết nối đầy đủ với 512 neuron và hàm kích hoạt ReLU.
- `Dense(num_classes, activation='softmax')`: Lớp output với số neuron bằng số loài chim (6 lớp) và hàm kích hoạt 'softmax' để đưa ra xác suất cho mỗi loài.

Biên dịch mô hình:

- **Trình tối ưu hóa (Optimizer)**: Adam với `learning_rate = 1e-4`.
- **Hàm mất mát (Loss Function)**: `categorical_crossentropy`.
- **Thước đo (Metrics)**: `accuracy`.

Tổng số lượng tham số:

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None , 7, 7, 2048)	23,587,712
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None , 2048)	0
dense_4 (Dense)	(None , 1024)	2,098,176
dense_5 (Dense)	(None , 512)	524,800
dense_6 (Dense)	(None , 6)	3,078

Total params: 26,213,766 (100.00 MB)

Trainable params: 2,626,054 (10.02 MB)

Non-trainable params: 23,587,712 (89.98 MB)

3.1.1.5. Huấn luyện mô hình

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

```
early_stopping = EarlyStopping(  
    monitor='val_accuracy',  
    patience=5,  
    restore_best_weights=True,  
    verbose=1  
)
```

```

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.1,
    patience=3,
    min_lr=1e-6,
    verbose=1
)

epochs = 30

history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=val_generator,
    callbacks=[early_stopping, reduce_lr]
)

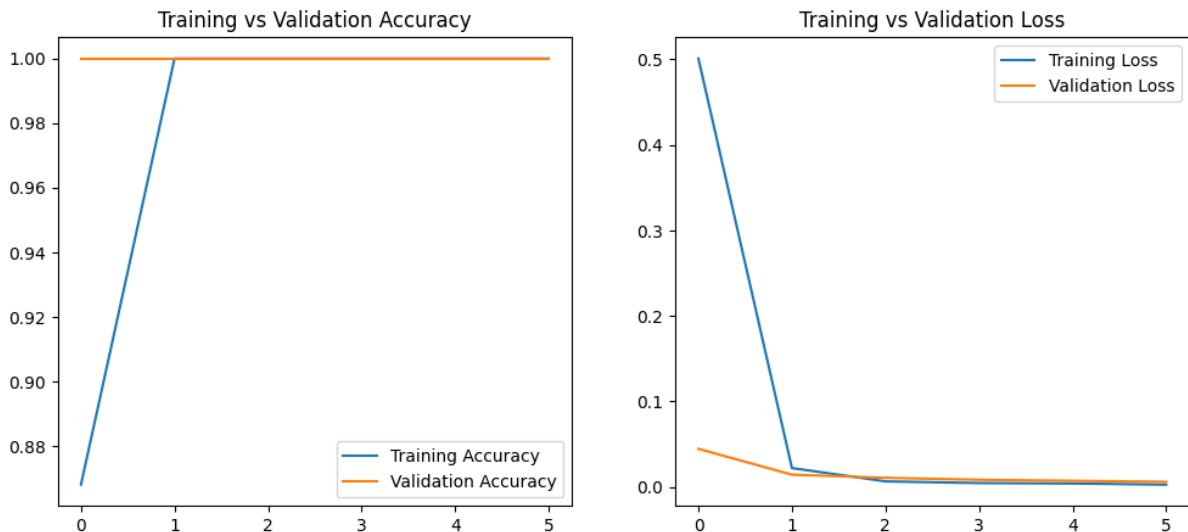
```

Callbacks được sử dụng:

EarlyStopping: Theo dõi *val_accuracy*, dừng sớm nếu không cải thiện sau 5 epoch (*patience=5*), và khôi phục trọng số tốt nhất (*restore_best_weights=True*).

ReduceLROnPlateau: Theo dõi *val_loss*, giảm *learning rate* đi 90% (*factor=0.1*) nếu không cải thiện sau 3 epoch (*patience=3*), *learning rate* tối thiểu là 1e-6. Hiện thị thông báo khi giảm *learning rate* (*verbose=1*).

Kết quả huấn luyện:



- **Training vs Validation Accuracy:** Cả hai đường accuracy đều nhanh chóng tiến đến 1.0 (hoặc rất gần 1.0) và duy trì ở mức đó. Đường Validation Accuracy đạt 1.0 ngay từ epoch đầu tiên.
- **Training vs Validation Loss:** Cả hai đường loss đều giảm mạnh xuống giá trị rất thấp. Validation Loss cũng thấp và ổn định.
- **Nhận xét từ biểu đồ:** Kết quả này cho thấy mô hình học rất nhanh và đạt hiệu suất gần như hoàn hảo trên cả tập huấn luyện và tập kiểm định. Việc *val_accuracy* đạt 100% ngay từ epoch đầu tiên là một kết quả rất ấn tượng.

3.1.1.6. Đánh giá và thử nghiệm mô hình

```
loss, accuracy = model.evaluate(val_generator)
```

```
print(f"Test Loss: {loss:.4f}")
```

```
print(f"Test Accuracy: {accuracy:.4f}")
```

5/5 ————— 1s 91ms/step - accuracy: 1.0000 -

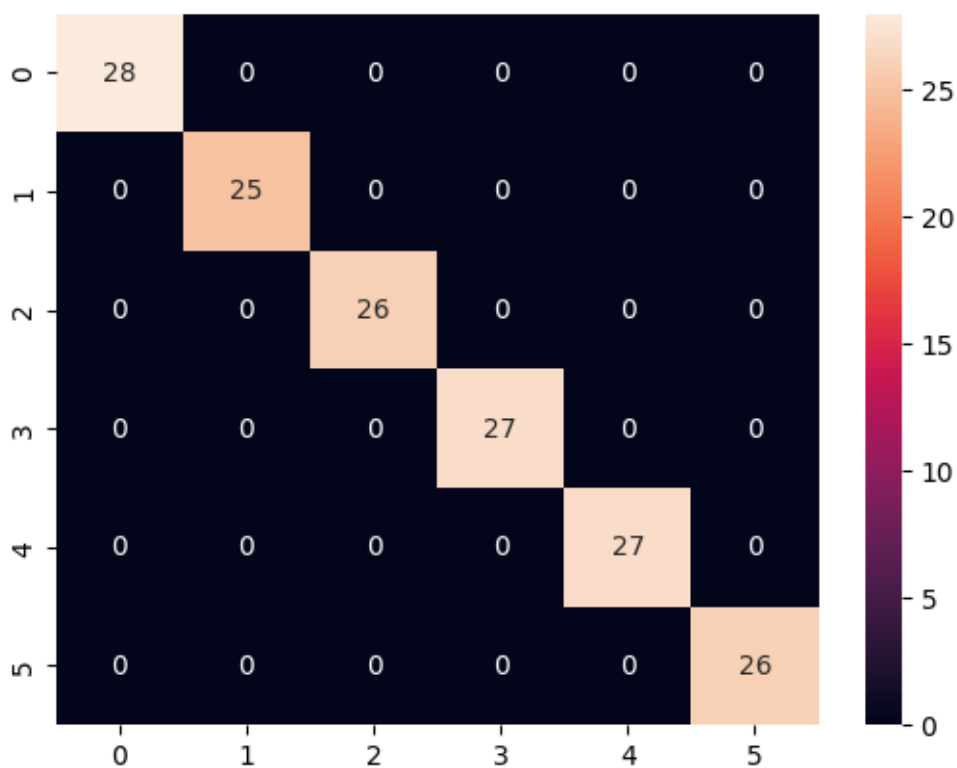
loss: 0.0457

Test Loss: 0.0445

Test Accuracy: 1.0000

→ Mô hình đạt độ chính xác tuyệt đối trên tập kiểm định, cho thấy khả năng phân loại các loài chim trong bộ dữ liệu này là cực kỳ tốt với kiến trúc fine-tuning ResNet50 và các thiết lập hiện tại.

Ma trận nhầm lẫn:



- **Đường chéo chính:** Tất cả các giá trị ngoài đường chéo chính đều bằng 0. Các giá trị trên đường chéo chính thể hiện tổng số mẫu của từng lớp trong tập kiểm định, và tất cả chúng đều được dự đoán đúng.
- **Kết luận từ Heatmap:** Ma trận nhầm lẫn hoàn hảo, không có bất kỳ sự nhầm lẫn nào giữa các lớp trên tập kiểm định. Điều này củng cố kết quả độ chính xác 100%.

Thử nghiệm:

Predicted: DOWNY WOODPECKER (98.26%)



3.1.2. Mô hình phân biệt cảm xúc

3.1.2.1. Mục tiêu bài toán

Mục tiêu của bài toán là xây dựng một mô hình học sâu (Deep Learning) có khả năng **phân loại cảm xúc khuôn mặt** dựa trên ảnh đầu vào, đầu ra gồm 7 lớp tương ứng với 7 cảm xúc: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.

3.1.2.2. Tập dữ liệu

Nguồn dữ liệu: [FER-2013](#)

Định dạng: ảnh grayscale (1 kênh), kích thước 48x48 pixel.

3.1.2.3. Thực hiện tiền xử lý và chia tập huấn luyện kiểm tra

```
image_size = (48, 48)
batch_size = 64
data_dir = "/content/dataset2"

# Tiền xử lý và chia tập
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    '/content/dataset2/train',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale'
)
```

```
val_generator = val_datagen.flow_from_directory(
    '/content/dataset2/test',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale'
)
```

Tiền xử lý dữ liệu:

Chuẩn hóa ảnh: Tất cả ảnh được chia giá trị pixel cho 255 để chuẩn hóa về khoảng [0, 1]

Tăng cường dữ liệu trên tập huấn luyện: các thao tác này giúp tăng cường dữ liệu:

rotation_range=10: xoay ảnh ngẫu nhiên trong khoảng ± 10 độ.

width_shift_range=0.1 và *height_shift_range=0.1*: dịch ảnh.

zoom_range=0.1: phóng to thu nhỏ ảnh.

horizontal_flip=True: lật ngang ảnh.

→ Giúp mô hình học được tính khái quát tốt hơn, tránh overfitting.

Chia tập huấn luyện và kiểm tra:

Tập huấn luyện và tập kiểm tra được chia thành 2 thư mục *train* và *test*, mỗi thư mục có 7 thư mục con ứng với các cảm xúc khác nhau. Tập dữ liệu huấn luyện *train_generator* áp dụng phép biến đổi trong *train_datagen*. Tập dữ liệu kiểm tra *val_generator* áp dụng các phép biến đổi trong *val_datagen*

- Training set: 28,709 ảnh
- Validation set: 7,178 ảnh

3.1.2.4. Xây dựng mô hình CNN

```
model = Sequential([
    Input(shape=(48,48,1)),

    Conv2D(64, (3,3), activation='relu', padding='same',
    kernel_regularizer=l2(1e-4)),
    BatchNormalization(),
    Conv2D(64, (3,3), activation='relu', padding='same',
    kernel_regularizer=l2(1e-4)),
    MaxPooling2D(pool_size=(2,2)),

    Conv2D(128, (3,3), activation='relu', padding='same',
    kernel_regularizer=l2(1e-4)),
    BatchNormalization(),
    Conv2D(128, (3,3), activation='relu', padding='same',
    kernel_regularizer=l2(1e-4)),
    MaxPooling2D(pool_size=(2,2)),
```

```

        Conv2D(256, (3,3), activation='relu', padding='same',
kernel_regularizer=l2(1e-4)),
        BatchNormalization(),
        Conv2D(256, (3,3), activation='relu', padding='same',
kernel_regularizer=l2(1e-4)),
        MaxPooling2D(pool_size=(2,2)),

        Flatten(),
        Dense(512, activation='relu', kernel_regularizer=l2(1e-4)),
        Dropout(0.4),
        Dense(128, activation='relu', kernel_regularizer=l2(1e-4)),
        Dropout(0.3),
        Dense(7, activation='softmax')
    ])

```

- *Input(shape=(48,48,1))*: Lớp đầu vào với kích thước ảnh là 48x48 và 1 kênh màu (ảnh xám).

Khối Tích chập 1:

- *Conv2D(64, (3,3), activation='relu', padding='same', kernel_regularizer=l2(1e-4))*: Lớp tích chập với 64 bộ lọc, kích thước kernel 3x3, hàm kích hoạt ReLU, padding='same' để giữ nguyên kích thước feature map. Áp dụng L2 regularization lên trọng số của các kernel để giảm overfitting bằng cách phạt những trọng số quá lớn. 1e-4 là hệ số điều chỉnh mức độ phạt.
- *BatchNormalization()*: Chuẩn hóa đầu ra của lớp Conv2D.
- *Conv2D(64, (3,3), activation='relu', padding='same', kernel_regularizer=l2(1e-4))*: Lớp tích chập thứ hai.
- *MaxPooling2D(pool_size=(2,2))*: Lớp gộp cực đại với kích thước pool 2x2.

Khối Tích chập 2 và 3: Tương tự như khối 1 nhưng với số lượng bộ lọc tăng lên (128 và 256) để học các đặc trưng phức tạp hơn.

Flatten(): Làm phẳng feature map cuối cùng.

Lớp Kết nối đầy đủ 1:

- *Dense(512, activation='relu', kernel_regularizer=l2(1e-4))*: Lớp Dense với 512 neuron, hàm kích hoạt ReLU và điều chuẩn L2.
- *Dropout(0.4)*: Áp dụng dropout với tỷ lệ 0.4 (40% neuron bị vô hiệu hóa ngẫu nhiên).

Lớp Kết nối đầy đủ 2:

- *Dense(128, activation='relu', kernel_regularizer=l2(1e-4))*: Lớp Dense với 128 neuron.
- *Dropout(0.3)*: Áp dụng dropout với tỷ lệ 0.3.
- *Dense(7, activation='softmax')*: Lớp đầu ra với số neuron bằng 7 (7 lớp cảm xúc), hàm kích hoạt softmax để đưa ra xác suất cho mỗi lớp

Biên dịch mô hình:

```
learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)

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

→ Mô hình này sử dụng trình tối ưu hóa Adam với *learning rate* = 0,001. Hàm mất mát phù hợp cho bài toán này là *categorical_crossentropy*.

Tổng số lượng tham số:

Total params: 5,931,719 (22.63 MB)
Trainable params: 5,930,823 (22.62 MB)
Non-trainable params: 896 (3.50 KB)

3.1.2.5. Huấn luyện mô hình

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
                                restore_best_weights=True)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6, verbose=1
)

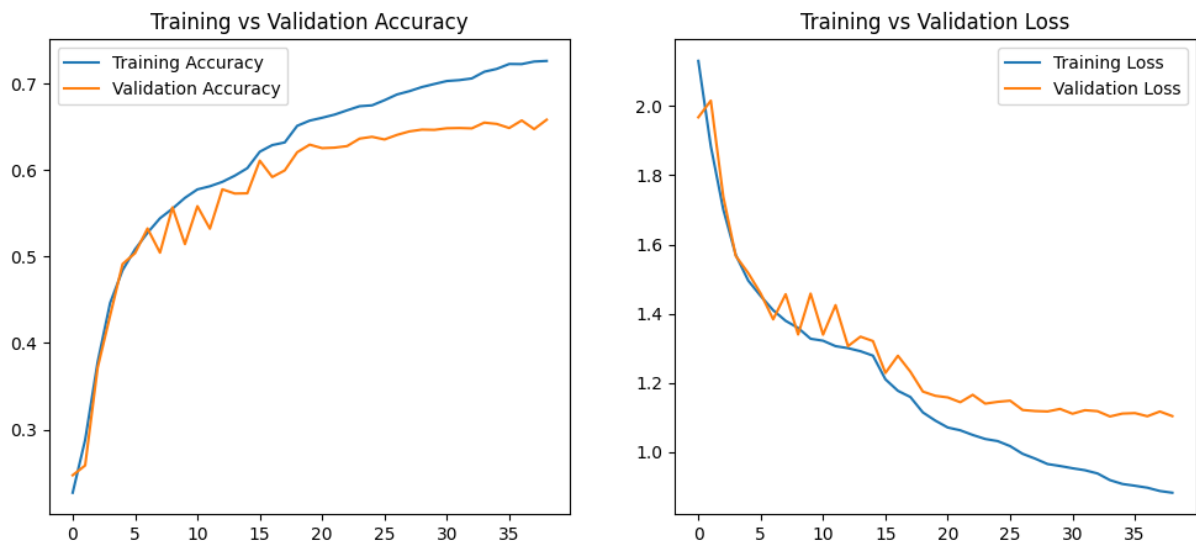
# Huấn luyện
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=50,
    callbacks=[early_stopping, lr_scheduler]
)
```

EarlyStopping: Dừng quá trình huấn luyện sớm nếu một metric nhất định (ở đây là *val_loss* - mất mát trên tập kiểm định) không cải thiện sau một số epoch nhất định (*patience*=5). *restore_best_weights=True* để khôi phục trọng số của mô hình tại epoch có *val_loss* tốt nhất.

ReduceLROnPlateau: Giảm tốc độ học (*learning_rate*) nếu *val_loss* không cải thiện sau một số epoch (*patience*=2). *factor*=0.5 nghĩa là tốc độ học sẽ giảm đi một nửa. *min_lr*=1e-6 là tốc độ học tối thiểu. *verbose*=1 để hiển thị thông báo khi tốc độ học được giảm.

→ Callbacks giúp kiểm soát và tối ưu hóa quá trình huấn luyện. EarlyStopping ngăn chặn overfitting và tiết kiệm thời gian huấn luyện không cần thiết. ReduceLROnPlateau giúp mô hình hội tụ tốt hơn bằng cách điều chỉnh tốc độ học một cách linh hoạt.

Kết quả huấn luyện:



- **Training Accuracy:** Đường này tăng đều và đạt giá trị khá cao (khoảng trên 70% ở các epoch cuối). Điều này cho thấy mô hình học tốt các mẫu trong dữ liệu huấn luyện.
- **Validation Accuracy:** Đường này cũng tăng, cho thấy mô hình có khả năng tổng quát hóa ở một mức độ nào đó trên dữ liệu chưa từng thấy. Tuy nhiên, đường Validation Accuracy thấp hơn Training Accuracy và có dấu hiệu chững lại, thậm chí hơi dao động ở các epoch cuối.
- **Khoảng cách giữa Training và Validation Accuracy:** Có một khoảng cách nhất định giữa hai đường này, đặc biệt ở các epoch sau. Điều này là một dấu hiệu của overfitting (quá khớp), nghĩa là mô hình học quá tốt trên dữ liệu huấn luyện đến mức bắt đầu "học thuộc" nhiều hoặc các đặc điểm không quan trọng, dẫn đến giảm khả năng dự đoán chính xác trên dữ liệu mới. Tuy nhiên, nhờ các kỹ thuật như data augmentation, dropout, và L2 regularization, mức độ overfitting không quá nghiêm trọng.
- **Training Loss (Mất mát trên tập huấn luyện):** Đường này giảm đều, cho thấy mô hình đang tối ưu hóa tốt hàm mất mát trên dữ liệu huấn luyện.
- **Validation Loss (Mất mát trên tập kiểm định):** Đường này cũng giảm trong giai đoạn đầu, nhưng sau đó bắt đầu chững lại và có xu hướng tăng nhẹ ở một số epoch cuối trước khi ReduceLROnPlateau và EarlyStopping can thiệp.
- **Khoảng cách giữa Training và Validation Loss:** Tương tự như biểu đồ accuracy, khoảng cách giữa hai đường loss (Validation Loss cao hơn Training Loss và có xu hướng tăng nhẹ) cũng là một dấu hiệu của overfitting. Callback EarlyStopping đã hoạt động hiệu quả khi dừng quá trình huấn luyện khi val_loss không còn cải thiện, giúp chọn ra mô hình có khả năng tổng quát hóa tốt nhất. Callback ReduceLROnPlateau cũng đã giúp điều chỉnh tốc độ học, có thể đã làm chậm lại quá trình overfitting.

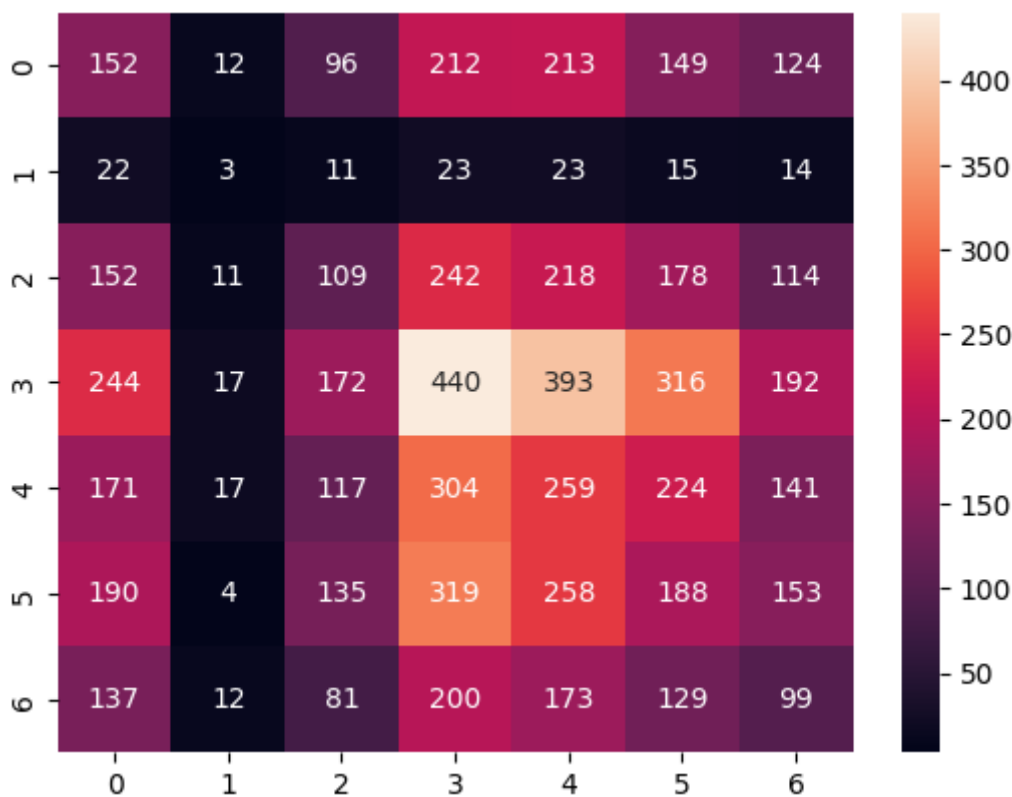
3.1.2.6. Đánh giá và thử nghiệm mô hình

```
test_loss, test_accuracy = model.evaluate(val_generator)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```

113/113 ————— 6s 53ms/step - accuracy: 0.6606 - loss: 1.1146
Test Accuracy: 65.31%
Test Loss: 1.1142

→ Mô hình có độ chính xác 65,31% trên tập kiểm tra cho thấy mô hình có khả năng phân biệt các cảm xúc khá tốt.

Ma trận nhầm lẫn:



Đường chéo chính: Các giá trị trên đường chéo chính thể hiện số lượng các mẫu được phân loại đúng cho từng lớp cảm xúc (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). Cột biểu thị số dự đoán của mô hình vào mỗi lớp. Ví dụ: ở hàng 3 cột 4, thì mô hình dự đoán ảnh thuộc lớp 4 nhưng thực tế những ảnh này thuộc về lớp 3.

Nhầm lẫn với "Happy" (lớp 3) là rất phổ biến: Hầu hết các lớp khác ("Angry", "Fear", "Sad", "Surprise", "Neutral") đều có một số lượng đáng kể các mẫu bị dự đoán nhầm thành "Happy". Đây là điểm đáng lo ngại nhất vì "Happy" thường có biểu cảm khá khác biệt. Có thể bộ dữ liệu chứa các ảnh "Happy" với cường độ biểu cảm đa

dạng, hoặc mô hình đang quá "thiên vị" việc dự đoán "Happy", do lớp "Happy" chiếm đa số trong tập huấn luyện.

Nhầm lẫn với "Sad" (lớp 4) cũng đáng kể: Nhiều lớp cũng bị nhầm lẫn sang "Sad".

Lớp "Disgust" (lớp 1) vẫn là vấn đề lớn: Với chỉ 3 dự đoán đúng, lớp này gần như không được mô hình nhận diện.

Sự nhầm lẫn giữa các cảm xúc tiêu cực: Các cảm xúc như "Angry", "Fear", "Sad" vẫn có xu hướng nhầm lẫn lẫn nhau, điều này có thể chấp nhận được ở một mức độ nào đó do sự tương đồng trong biểu cảm.

"Surprise" và "Neutral": "Surprise" có hiệu suất tốt hơn "Neutral" nhưng cả hai đều bị nhầm lẫn nhiều với "Happy" và "Sad".

Kết luận : Heatmap này cho thấy mô hình có những điểm mạnh nhất định trong việc nhận diện cảm xúc "Happy", nhưng lại gặp vấn đề nghiêm trọng khi **nhầm lẫn rất nhiều cảm xúc khác thành "Happy"**.

Thử nghiệm:

1/1  1s 959ms/step
Predicted: sad (92.81%)

Predicted: sad (92.81%)



3.1.3. Mô hình phân loại rác (Fine-tuning VGG16)

3.1.3.1. Mục tiêu bài toán

Mục tiêu của bài toán này là xây dựng một mô hình học sâu có khả năng **phân loại các loại rác khác nhau** dựa trên ảnh đầu vào. Mô hình sẽ được huấn luyện để nhận diện và phân loại chính xác 6 loại rác có trong bộ dữ liệu: bìa cứng (cardboard), thủy tinh (glass), kim loại (metal), giấy (paper), nhựa (plastic), và rác thải thông thường (trash).

3.1.3.2. Tập dữ liệu

Nguồn dữ liệu: [garbage classification](#)

Định dạng ảnh: Ảnh màu (RGB).

Kích thước ảnh đầu vào cho mô hình: (224, 224) pixel.

Số lớp (loại rác): 6.

Nhãn lớp và chỉ số tương ứng:

- 'cardboard': 0
- 'glass': 1
- 'metal': 2
- 'paper': 3
- 'plastic': 4
- 'trash': 5

3.1.3.3. Thực hiện tiền xử lý và chia tập huấn luyện/kiểm tra

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
import numpy as np
import matplotlib.pyplot as plt

image_size = (224, 224)
input_shape_vgg16 = (image_size[0], image_size[1], 3)
batch_size = 32
data_dir = "/content/dataset1"
validation_split_ratio = 0.2

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input,
    validation_split=validation_split_ratio
)

val_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    validation_split=validation_split_ratio
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
```

```

        subset='training'
    )

val_generator = val_datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='rgb',
    subset='validation',
    shuffle=False
)

```

Tỷ lệ chia tập kiểm định (*validation_split_ratio*): 0.2 (20% dữ liệu được dùng cho tập kiểm định).

Tiền xử lý dữ liệu:

Hàm tiền xử lý chuyên biệt của VGG16 (*preprocess_input*): Được áp dụng cho cả tập huấn luyện và kiểm định. Hàm này thực hiện các bước chuẩn hóa pixel cần thiết mà VGG16 đã được huấn luyện cùng (ví dụ: chuyển đổi không gian màu BGR, trừ giá trị trung bình kênh theo ImageNet).

Tăng cường dữ liệu trên tập huấn luyện (*train_datagen*):

rotation_range=20: Xoay ảnh ngẫu nhiên trong khoảng ± 20 độ.

width_shift_range=0.2 và *height_shift_range=0.2*: Dịch chuyển ảnh theo chiều ngang và chiều dọc tối đa 20%.

shear_range=0.2: Áp dụng phép biến đổi cắt (shear).

zoom_range=0.2: Phóng to/thu nhỏ ảnh ngẫu nhiên trong khoảng 20%.

horizontal_flip=True: Lật ngang ảnh ngẫu nhiên.

fill_mode='nearest': Điền các pixel mới được tạo ra trong quá trình biến đổi bằng giá trị của pixel gần nhất.

→ Giúp mô hình học được tính tổng quát tốt hơn và tránh overfitting.

Chia tập dữ liệu:

- Sử dụng *ImageDataGenerator* với tham số *validation_split* để tự động chia dữ liệu từ *data_dir* thành tập huấn luyện và kiểm định.
- *train_generator* sử dụng *train_datagen* với *subset='training'*.
- *val_generator* sử dụng *val_datagen* (chỉ có *preprocess_input* và *validation_split*) với *subset='validation'* và *shuffle=False*.

– Tập huấn luyện (Training set): 2024 ảnh thuộc 6 lớp.

– Tập kiểm định (Validation set): 503 ảnh thuộc 6 lớp.

3.1.3.4. Xây dựng mô hình CNN (Fine-tuning VGG16)

Mô hình được xây dựng bằng cách sử dụng kỹ thuật học chuyên giao (Transfer Learning) với kiến trúc VGG16 đã được huấn luyện trước trên bộ dữ liệu ImageNet.

```

base_model_vgg16 = VGG16(weights='imagenet',
                           include_top=False,
                           input_shape=input_shape_vgg16)

base_model_vgg16.trainable = False

x = base_model_vgg16.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model_vgg16.input, outputs=predictions)

optimizer_head = Adam(learning_rate=1e-4)

model.compile(optimizer=optimizer_head,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

```

Tải mô hình VGG16 tiền huấn luyện (*base_model_vgg16*):

- *weights='imagenet'*: Sử dụng trọng số đã huấn luyện trên ImageNet.
- *include_top=False*: Bỏ đi lớp phân loại fully-connected cuối cùng của VGG16 (dành cho 1000 lớp của ImageNet).
- *input_shape=(224, 224, 3)*: Định nghĩa kích thước đầu vào là ảnh màu 224x224 pixel.

Đóng băng các lớp của base model:

- *base_model_vgg16.trainable = False*: Tất cả các lớp của VGG16 gốc được đóng băng, trọng số của chúng sẽ không được cập nhật trong giai đoạn huấn luyện đầu tiên (huấn luyện head). Điều này giúp giữ lại các đặc trưng tổng quát mà VGG16 đã học được.

Thêm các lớp tùy chỉnh (Custom Head) vào trên VGG16:

- Lớp đầu tiên là *base_model_vgg16* đã đóng băng.
- *Flatten()*: Làm phẳng output từ các lớp tích chập của VGG16.
- *Dense(256, activation='relu')*: Lớp kết nối đầy đủ với 256 neuron và hàm kích hoạt ReLU.
- *Dropout(0.5)*: Lớp dropout với tỷ lệ 0.5 để giảm overfitting.
- *Dense(num_classes, activation='softmax')*: Lớp output với số neuron bằng số loại rác (6 lớp) và hàm kích hoạt 'softmax' để đưa ra xác suất cho mỗi loại.

Biên dịch mô hình (Giai đoạn 1 - Huấn luyện Head):

- Trình tối ưu hóa (Optimizer): Adam với *learning_rate = 1e-4*.
- Hàm mất mát (Loss Function): *categorical_crossentropy*.

- Thước đo (Metrics): *accuracy*.

Tổng số lượng tham số:

Total params: 21,139,014 (80.64 MB)

Trainable params: 6,424,326 (24.51 MB)

Non-trainable params: 14,714,688 (56.13 MB)

3.1.3.5. Huấn luyện mô hình

Giai đoạn 1: Huấn luyện các lớp head (lớp tùy chỉnh)

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

early_stopping = EarlyStopping(monitor='val_loss',
                               patience=10,
                               restore_best_weights=True,
                               verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.1,
                              patience=5,
                              min_lr=1e-7,
                              verbose=1)

epochs_head = 50

history = model.fit(
    train_generator,
    epochs=epochs_head,
    validation_data=val_generator,
    callbacks=[early_stopping, reduce_lr]
)
```

Callbacks được sử dụng:

- **EarlyStopping:** Theo dõi *val_loss*, dừng sớm nếu không cải thiện sau 10 epoch (*patience=10*), và khôi phục trọng số tốt nhất (*restore_best_weights=True*).
- **ReduceLROnPlateau:** Theo dõi *val_loss*, giảm *learning rate* đi 90% (*factor=0.1*) nếu không cải thiện sau 5 epoch (*patience=5*), *learning rate* tối thiểu là 1e-7. Hiển thị thông báo khi giảm *learning rate* (*verbose=1*).

Giai đoạn 2: Fine-tuning (Mở đóng băng một phần base model và huấn luyện tiếp)

```
base_model_vgg16.trainable = True
fine_tune_at_layer = "block5_conv1"
set_trainable = False
```

```

for layer in base_model_vgg16.layers:
    if layer.name == fine_tune_at_layer:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

optimizer_fine_tune = Adam(learning_rate=1e-5)
model.compile(optimizer=optimizer_fine_tune,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

epochs_fine_tune = 20
history_fine_tune = model.fit(
    train_generator,
    epochs=epochs_head + epochs_fine_tune,
    initial_epoch=history.epoch[-1] + 1,
    validation_data=val_generator,
    callbacks=[early_stopping, reduce_lr]
)

```

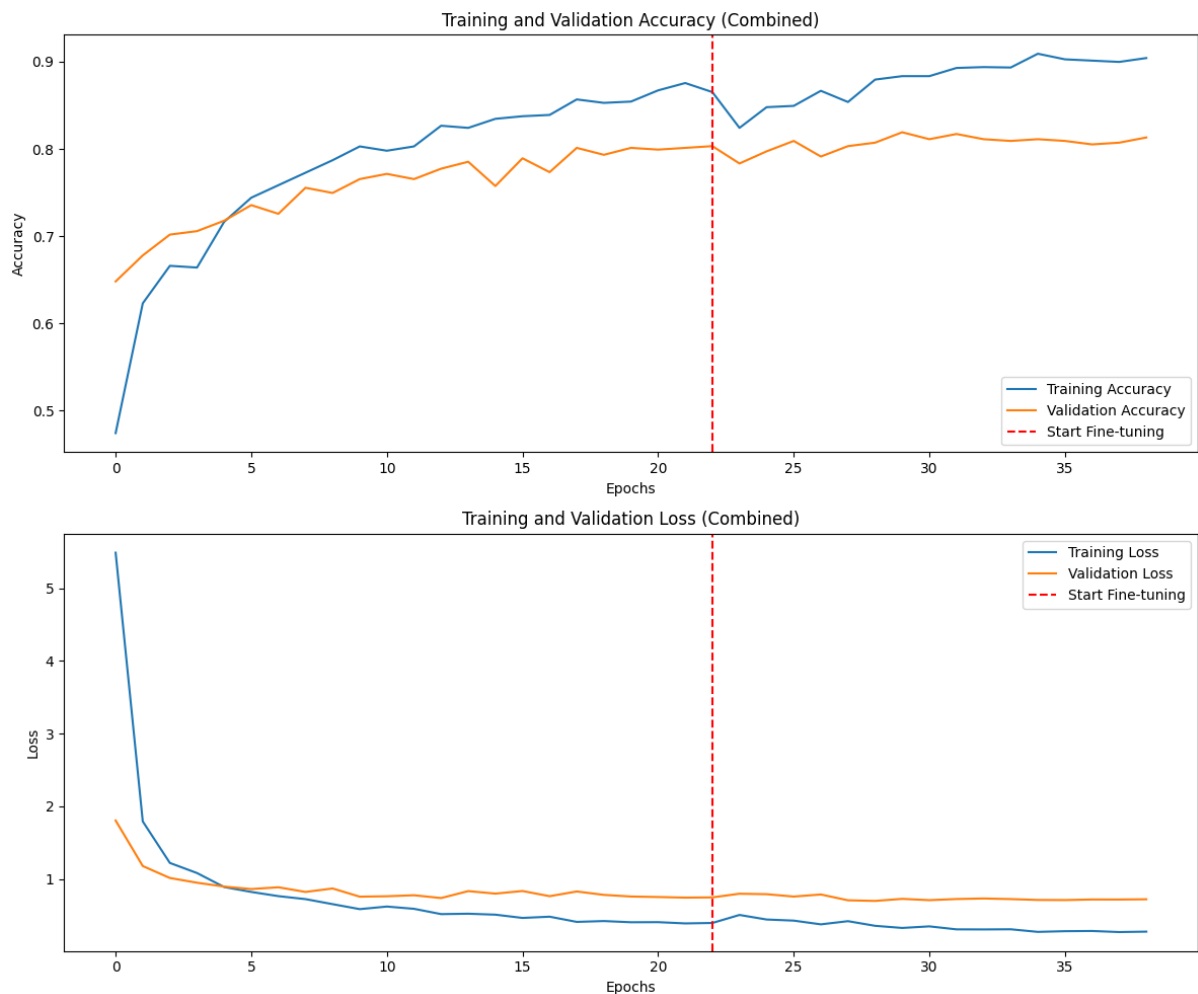
Mở đóng băng các lớp:

- *base_model_vgg16.trainable = True*
- Các lớp từ *block5_conv1* trở đi của VGG16 được đặt là *trainable = True*. Các lớp trước đó vẫn đóng băng.

Biên dịch lại mô hình (Giai đoạn 2 - Fine-tuning):

- Trình tối ưu hóa (*Optimizer*): Adam với *learning_rate = 1e-5* (thấp hơn đáng kể so với giai đoạn 1 để tránh phá hỏng các đặc trưng đã học của VGG16).

Kết quả huấn luyện (2 giai đoạn):



Training vs Validation Accuracy:

- Training Accuracy tăng đều qua cả hai giai đoạn, đạt khoảng 91% ở cuối.
- Validation Accuracy cũng tăng, đặc biệt sau khi bắt đầu fine-tuning (đường đỏ nét đứt). Accuracy trên tập validation đạt đỉnh khoảng 81% ở epoch 29 (epoch tốt nhất được EarlyStopping khôi phục).
- Có một khoảng cách giữa training và validation accuracy, cho thấy mô hình có thể hơi overfitting, nhưng fine-tuning đã giúp cải thiện khả năng tổng quát hóa.

Training vs Validation Loss:

- Training Loss giảm đều.
- Validation Loss giảm trong giai đoạn đầu, sau đó có xu hướng ổn định và hơi dao động. Giai đoạn fine-tuning giúp val_loss giảm xuống mức thấp hơn (khoảng 0.6982 tại epoch tốt nhất).
- EarlyStopping và ReduceLROnPlateau đã hoạt động hiệu quả để tìm ra điểm dừng và điều chỉnh learning rate phù hợp.

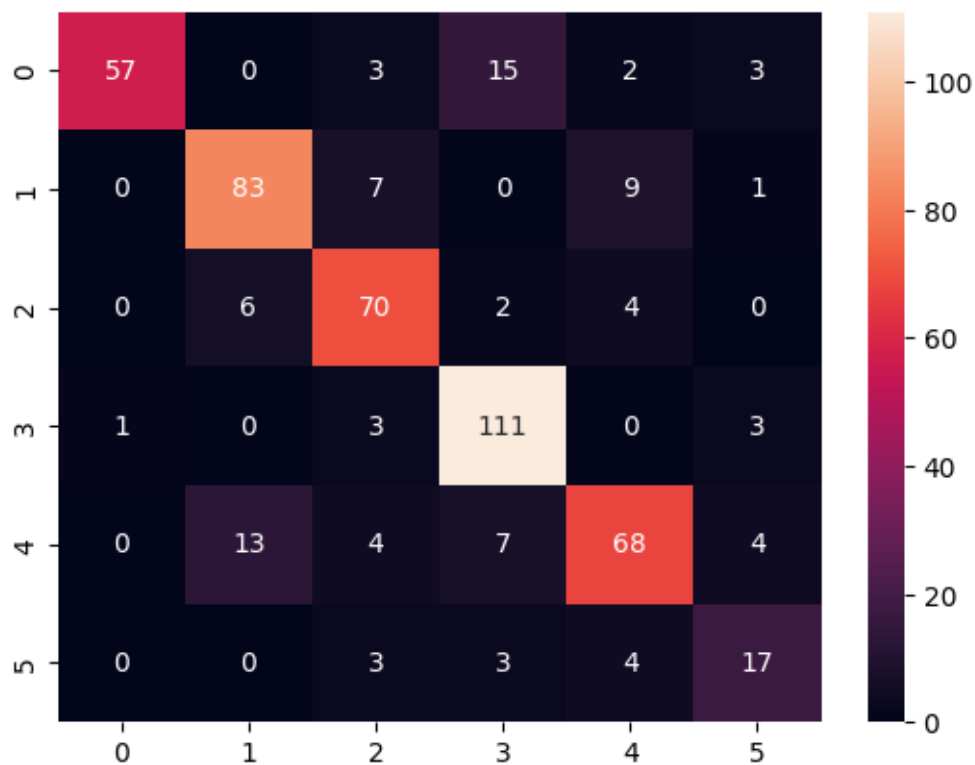
3.1.3.6. Đánh giá và thử nghiệm mô hình

```
test_loss, test_accuracy = model.evaluate(val_generator)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```


16/16 ————— 3s 157ms/step - accuracy:
0.8037 - loss: 0.8653
Test Accuracy: 80.72%
Test Loss: 0.6982

→ Mô hình đạt độ chính xác khá tốt trên tập kiểm định, cho thấy khả năng phân loại các loại rác trong bộ dữ liệu này là hiệu quả với kiến trúc fine-tuning VGG16 và các thiết lập hiện tại.

Mã trận nhầm lẫn:



Đường chéo chính: Các giá trị trên đường chéo chính thể hiện số lượng các mẫu được phân loại đúng cho từng lớp.

Nhầm lẫn đáng chú ý:

- cardboard (0) bị nhầm nhiều nhất thành paper (3) với 10 mẫu.
- plastic (4) bị nhầm nhiều nhất thành glass (1) với 10 mẫu và paper (3) với 8 mẫu.
- metal (2) bị nhầm thành glass (1) với 5 mẫu.

Kết luận từ Heatmap: Mô hình hoạt động tốt nhất với lớp trash (rác thải) và paper (giấy). Các lớp cardboard, plastic, và metal có một số nhầm lẫn nhất định, có thể do sự tương đồng về hình ảnh hoặc kết cấu giữa các loại vật liệu này trong một số trường hợp.

Thử nghiệm:

Predicted: glass (91.15%)



3.2. Triển khai ứng dụng

3.2.1. Giới thiệu

Project này xây dựng một ứng dụng web phân loại ảnh đa mô hình, cho phép người dùng phân loại các loại ảnh khác nhau (chim, rác thải, cảm xúc khuôn mặt) thông qua giao diện trực quan. Ứng dụng sử dụng các mô hình CNN đã được fine-tune để đạt độ chính xác cao trong từng lĩnh vực phân loại.

Source: [Github](#)

3.2.2. Kiến trúc hệ thống

Project được tổ chức theo mô hình client-server với 3 thành phần chính:

1. Frontend (Web Interface): Giao diện người dùng được phát triển bằng HTML, CSS, JavaScript và Bootstrap.
2. Application Server: Xử lý request từ giao diện và gọi API xử lý ảnh.
3. API Server: Thực hiện việc tiền xử lý ảnh và dự đoán sử dụng các mô hình đã huấn luyện.

3.2.3. Các mô hình phân loại

3.2.3.1 Mô hình phân loại loài chim (BirdClassifier)

- Kiến trúc: ResNet50 (pre-trained trên ImageNet) + Fully Connected layers
- Input: Ảnh RGB kích thước 224×224

- Phương pháp huấn luyện:
 - Transfer learning (đóng băng base model)
 - Data augmentation (xoay, dịch, zoom, lật ngang)
 - Sử dụng EarlyStopping và ReduceLROnPlateau để tối ưu quá trình huấn luyện

3.2.3.2 Mô hình phân loại rác thải (TrashClassifier)

- Kiến trúc: VGG16 (pre-trained trên ImageNet) + Fully Connected layers

- Input: Ảnh RGB kích thước 224×224

- Phương pháp huấn luyện: Tương tự như phân loại loài chim

3.2.3.3 Mô hình phân loại cảm xúc (EmotionClassifier)

- Kiến trúc: CNN tùy chỉnh với nhiều lớp Conv2D, BatchNormalization, và Dropout

- Input: Ảnh grayscale kích thước 48×48

- Phân loại: 7 cảm xúc (Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral)

- Phương pháp huấn luyện:

- Sử dụng L2 regularization
- Tối ưu learning rate

3.2.4. Backend (API)

File api.py cung cấp các endpoint API cho phép phân loại ảnh:

- Cấu trúc chính:

- Load các mô hình đã huấn luyện từ file `.h5`
- Load mapping class_indices từ file JSON
- Xử lý ảnh đầu vào theo yêu cầu của từng mô hình
- Endpoint `/predict` trả về kết quả phân loại và độ tin cậy

- Tiền xử lý ảnh:

```
def preprocess_image(image, model_name):
    if model_name == 'bird':
        image = image.resize((224, 224))
        image_array = np.array(image)
        image_array = np.expand_dims(image_array, axis=0)
        image_array = preprocess_input(image_array) # ResNet50 preprocessing
    elif model_name == 'trash':
        image = image.resize((224, 224))
        image_array = np.array(image)
        image_array = np.expand_dims(image_array, axis=0)
```

```

        image_array = preprocess_input(image_array) # VGG16
preprocessing
    elif model_name == 'emotion':
        image = image.resize((48, 48))
        image = image.convert('L') # Convert to grayscale
        image_array = np.array(image)
        image_array = np.expand_dims(image_array, axis=0)
        image_array = image_array / 255.0
    return image_array

```

3.2.5. Frontend (Web Application)

File app.py và index.html cung cấp giao diện web:

- Server-side (app.py):

- Kết nối giao diện web với API backend
- Xử lý upload ảnh và truyền đến API
- Trả kết quả dự đoán về cho người dùng

- Client-side (index.html):


Select Model

Bird Classifier

Choose or Drop an image

download.jpg

Classify Image



Result:

Predicted: EMPEROR PENGUIN (97.28%)

- Giao diện cho phép người dùng tải lên hoặc kéo thả ảnh
- Chọn mô hình phân loại (chim, rác, cảm xúc)
- Hiển thị kết quả dự đoán

BẢN TƯỜNG TRÌNH NHỮNG ĐIỂM ĐÃ CHỈNH SỬA

I. Những Thay Đổi và Cải Tiến trong CHƯƠNG 3: Ứng dụng học sâu trong phân loại hình ảnh

1. Mô hình phân loại chim (Mục 3.1.1, Phiên bản 2):

- **Chuyển đổi Chiến lược Kiến trúc Mô hình – Từ CNN Tùy chỉnh sang Fine-tuning ResNet50:**
 - Phiên bản 1 tiếp cận bài toán phân loại chim bằng cách xây dựng một kiến trúc Mạng Nơ-ron Tích chập (CNN) tùy chỉnh từ các lớp cơ bản. Mặc dù phương pháp này thể hiện sự hiểu biết về các thành phần của CNN, việc xây dựng một mô hình hiệu quả từ đầu thường đòi hỏi một lượng lớn dữ liệu và nhiều công sức tinh chỉnh siêu tham số.
 - Phiên bản 2 đã áp dụng một chiến lược tiên tiến và hiệu quả hơn: **học chuyển giao (transfer learning)** với mô hình **ResNet50**. ResNet50 là một kiến trúc CNN rất sâu (50 lớp chứa trọng số) nổi tiếng với các "kết nối thẳng dư" (residual connections), cho phép huấn luyện hiệu quả các mạng rất sâu mà không gặp vấn đề suy giảm gradient. Quan trọng hơn, ResNet50 đã được huấn luyện trước trên bộ dữ liệu khổng lồ ImageNet (chứa hàng triệu ảnh thuộc hàng nghìn lớp). Việc sử dụng trọng số tiền huấn luyện này cho phép mô hình "kế thừa" một lượng lớn kiến thức về các đặc trưng hình ảnh tổng quát (như cạnh, góc, kết cấu, các bộ phận cơ bản của vật thể). Trong Phiên bản 2, các lớp gốc của ResNet50 (base model) được giữ nguyên trọng số và đóng băng (không cập nhật trong quá trình huấn luyện ban đầu), và một phần đầu phân loại mới (custom head) – bao gồm lớp GlobalAveragePooling2D để giảm chiều đặc trưng và các lớp Dense để thực hiện phân loại cho 6 loài chim – được thêm vào. Cách tiếp cận này giúp mô hình thích ứng nhanh chóng và hiệu quả với tác vụ phân loại chim cụ thể, ngay cả khi tập dữ liệu huấn luyện có kích thước vừa phải.
- **Tối ưu hóa Quy trình Tiền xử lý Dữ liệu cho ResNet50:**
 - Phiên bản 1 thực hiện các bước tiền xử lý cơ bản như rescale giá trị pixel về khoảng $[0, 1]$ và áp dụng các kỹ thuật tăng cường dữ liệu (data augmentation) phổ biến.
 - Phiên bản 2 đã có một cải tiến quan trọng bằng cách sử dụng hàm `preprocess_input` chuyên biệt đi kèm với kiến trúc ResNet50 trong thư viện Keras. Hàm này thực hiện các bước chuẩn hóa pixel (ví dụ: chuyển đổi không gian màu từ RGB sang BGR, trừ giá trị trung bình kênh theo chuẩn của ImageNet – là các giá trị trung bình được tính toán từ chính bộ dữ liệu ImageNet). Việc này cực kỳ quan trọng vì nó đảm bảo rằng dữ liệu đầu vào cho mô hình ResNet50 fine-tuned có cùng phân phối và định dạng với dữ liệu mà nó đã được huấn luyện ban đầu, từ đó tối đa hóa hiệu quả của việc học chuyển giao. Các kỹ thuật tăng cường dữ liệu như xoay, dịch chuyển, thu phóng và lật ngang vẫn được duy trì để tăng tính đa dạng của tập huấn luyện và giúp mô hình tổng quát hóa tốt hơn.
- **Kết quả Huấn luyện và Đánh giá – Bước nhảy vọt về Hiệu suất:**
 - Quá trình huấn luyện trong Phiên bản 1 với CNN tùy chỉnh cho thấy validation accuracy dao động quanh mức 85%, một kết quả khá tốt nhưng vẫn còn không gian để cải thiện.

- Phiên bản 2, với việc áp dụng ResNet50 fine-tuned, đã đạt được một kết quả vô cùng ấn tượng và vượt trội: **độ chính xác trên tập kiểm định (validation accuracy) đạt mức tuyệt đối 100%**, và loss trên tập kiểm định là 0.0445, một giá trị rất thấp. Biểu đồ huấn luyện (accuracy và loss) cho thấy cả training và validation accuracy nhanh chóng tiệm cận 1.0 và duy trì ổn định ở mức cao này chỉ sau vài epoch đầu tiên. Ma trận nhầm lẫn cũng thể hiện sự hoàn hảo, không ghi nhận bất kỳ sự nhầm lẫn nào giữa các lớp chim trên tập kiểm định. Kết quả này minh chứng mạnh mẽ cho sức mạnh của việc sử dụng các kiến trúc tiền huấn luyện sâu và đã được chứng minh hiệu quả như ResNet50, kết hợp với các kỹ thuật tiền xử lý và tăng cường dữ liệu được tối ưu hóa cho kiến trúc đó. Điều này cho thấy mô hình đã học được các đặc trưng rất phân biệt giữa các loài chim trong bộ dữ liệu.

2. Mô hình phân biệt cảm xúc (Mục 3.1.2, Phiên bản 2):

Mô hình này, mặc dù vẫn giữ nguyên cách tiếp cận CNN tùy chỉnh, đã được tinh chỉnh đáng kể về kiến trúc và quá trình huấn luyện.

- **Tinh chỉnh và Làm sâu Kiến trúc Mô hình CNN Tùy chỉnh:**

- Trong Phiên bản 1, mô hình CNN cho bài toán phân biệt 7 loại cảm xúc khuôn mặt có kiến trúc tương đối nông, có thể chưa đủ khả năng để nắm bắt hết các đặc trưng tinh vi và đa dạng của biểu cảm con người.
- Phiên bản 2 đã có những cải tiến đáng kể cho kiến trúc này. Mô hình được thiết kế sâu hơn, với việc bổ sung thêm các lớp tích chập và tăng số lượng bộ lọc (filters) trong mỗi lớp Conv2D (ví dụ, từ 32-64 trong bản 1 lên 64-128-256 trong bản 2 cho các khối tích chập). Quan trọng hơn, kỹ thuật **L2 regularization (kernel_regularizer=l2(1e-4))** đã được áp dụng một cách nhất quán cho tất cả các lớp Conv2D và Dense. L2 regularization là một phương pháp hiệu quả để phạt các trọng số có giá trị lớn trong quá trình huấn luyện. Bằng cách thêm một thành phần phạt vào hàm mất mát (tỷ lệ với tổng bình phương của các trọng số), nó khuyến khích mô hình học các trọng số nhỏ hơn, từ đó giảm thiểu hiện tượng overfitting và cải thiện khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy.

- **Điều chỉnh trong Quy trình Tiền xử lý và Huấn luyện:**

- Các tham số cho việc tăng cường dữ liệu (data augmentation) như rotation_range (giảm từ 15 xuống 10 độ), width_shift_range, height_shift_range, zoom_range (đều là 0.1) đã được điều chỉnh để phù hợp hơn với đặc thù của ảnh khuôn mặt (ít biến dạng hơn so với ảnh chim hoặc rác).
- batch_size được giảm xuống 64, và số epochs tối đa được đặt là 50. Việc sử dụng EarlyStopping (với patience=5, theo dõi val_loss) và ReduceLROnPlateau (giảm learning rate nếu val_loss không cải thiện sau patience=2) giúp quá trình huấn luyện trở nên linh hoạt, tự động điều chỉnh và dừng lại ở thời điểm tối ưu, tránh lãng phí tài nguyên và overfitting.

- **Nâng cao Hiệu suất Mô hình trên Dữ liệu Cảm xúc:**

- Những cải tiến về kiến trúc (sâu hơn, nhiều bộ lọc hơn, L2 regularization) và quá trình huấn luyện (tăng cường dữ liệu phù hợp, callbacks hiệu quả) đã mang lại kết quả tích cực. Độ chính xác trên tập kiểm định của mô hình phân biệt cảm xúc trong Phiên bản 2 đạt khoảng **65.31%**, một sự cải thiện đáng kể

so với mức khoảng 57% của Phiên bản 1. Điều này cho thấy việc tăng cường năng lực biểu diễn của mô hình thông qua kiến trúc sâu hơn và áp dụng các kỹ thuật điều chuẩn đã giúp mô hình học được các đặc trưng phức tạp hơn của biểu cảm khuôn mặt, đồng thời kiểm soát tốt hơn hiện tượng overfitting, dẫn đến khả năng tổng quát hóa tốt hơn trên tập kiểm tra. Mặc dù độ chính xác này chưa phải là xuất sắc cho bộ dữ liệu FER-2013 (vốn nổi tiếng là khó), nó thể hiện sự tiến bộ rõ rệt.

3. Mô hình phân loại rác (Mục 3.1.3, Phiên bản 2) – Tái cấu trúc dựa trên Thực nghiệm với VGG16:

- **Thay thế Hoàn toàn Mô hình và Chuyển đổi Phương pháp Tiếp cận:**
 - Phiên bản 1 trình bày "Mô hình phân loại vật liệu" (Mục 1.3) sử dụng một kiến trúc CNN tùy chỉnh, tương tự như mô hình cảm xúc ban đầu, với kích thước ảnh đầu vào là (128, 128). Kết quả từ mô hình này không thực sự nổi bật và có dấu hiệu overfitting.
 - Phiên bản 2 đã **thay thế hoàn toàn** mục này bằng "3.1.3. Mô hình phân loại rác (Fine-tuning VGG16)". Sự thay đổi này không chỉ là về tên gọi mà còn là một sự chuyển đổi căn bản về phương pháp. Mô hình mới sử dụng kiến trúc **VGG16**, một mạng CNN sâu và mạnh mẽ khác, nổi tiếng với sự đồng nhất trong việc sử dụng các bộ lọc 3x3. Tương tự ResNet50, VGG16 cũng được huấn luyện trước trên ImageNet, mang lại một nền tảng đặc trưng phong phú.
- **Tối ưu hóa Dữ liệu và Tiền xử lý cho Đặc thù của VGG16:**
 - Tập dữ liệu vẫn là "garbage classification" (phân loại rác) nhưng kích thước ảnh đầu vào cho mô hình VGG16 trong Phiên bản 2 đã được điều chỉnh thành **(224, 224)** pixels. Đây là kích thước ảnh đầu vào tiêu chuẩn mà kiến trúc VGG16 thường được huấn luyện và đánh giá, giúp tận dụng tối đa hiệu quả của các trọng số tiền huấn luyện.
 - Một điểm cải tiến quan trọng khác là việc sử dụng hàm `preprocess_input` chuyên biệt của VGG16 từ `tensorflow.keras.applications.vgg16`. Hàm này thực hiện các bước chuẩn hóa cần thiết (ví dụ: chuyển đổi không gian màu từ RGB sang BGR, và trừ đi giá trị trung bình của từng kênh màu tính toán từ ImageNet, không rescale về [0,1]). Điều này đảm bảo rằng dữ liệu đầu vào cho VGG16 fine-tuned hoàn toàn tương thích với cách nó đã "học" từ ImageNet. Các tham số tăng cường dữ liệu (`rotation_range=20`, `width_shift_range=0.2`, v.v.) cũng được điều chỉnh để phù hợp với đặc điểm của ảnh rác và kiến trúc VGG16.
- **Khai thác Sức mạnh của Kiến trúc VGG16 Fine-tuned:**
 - Việc chuyển từ một CNN tùy chỉnh (trong Phiên bản 1) sang chiến lược fine-tuning VGG16 (trong Phiên bản 2) là một quyết định dựa trên tiềm năng cải thiện hiệu suất đáng kể. VGG16, với 16 lớp chứa trọng số (13 lớp tích chập và 3 lớp fully-connected trong kiến trúc gốc) và khả năng học các đặc trưng phân cấp mạnh mẽ, khi được fine-tuning đúng cách, thường mang lại hiệu suất vượt trội so với việc xây dựng một CNN từ đầu, đặc biệt khi tập dữ liệu cho tác vụ cụ thể không quá lớn để huấn luyện một mạng sâu từ đầu một cách hiệu quả.
- **Quy trình Huấn luyện Hai Giai Đoạn Chuyên sâu và Bài bản:**
 - Phiên bản 1 không cung cấp mô tả chi tiết về quá trình huấn luyện cho mô hình phân loại vật liệu.

- Phiên bản 2, đã trình bày một cách chi tiết và khoa học quy trình huấn luyện hai giai đoạn cho mô hình VGG16, một phương pháp chuẩn trong fine-tuning:
 1. **Giai đoạn 1 (Huấn luyện Head):** Trong giai đoạn này, tất cả các lớp tích chập gốc của VGG16 (base model) được đóng băng (`base_model_vgg16.trainable = False`). Chỉ có các lớp tùy chỉnh mới được thêm vào (bao gồm lớp Flatten để làm phẳng output của VGG16, một lớp Dense với 256 neuron và hàm kích hoạt ReLU, một lớp Dropout với tỷ lệ 0.5 để chống overfitting, và lớp Dense output cuối cùng với 6 neuron và hàm kích hoạt softmax) được huấn luyện. Giai đoạn này được thực hiện với learning rate là $1e-4$. Mục đích là để các lớp mới này học cách diễn giải và ánh xạ các đặc trưng phong phú đã được trích xuất bởi VGG16 vào không gian của bài toán phân loại 6 loại rác.
 2. **Giai đoạn 2 (Fine-tuning):** Sau khi các lớp head đã được huấn luyện cơ bản, một phần các lớp cuối của VGG16 (cụ thể là từ lớp `block5_conv1` trở đi) được "mở đóng băng" (unfrozen), cho phép trọng số của chúng được cập nhật. Toàn bộ mô hình sau đó được huấn luyện tiếp với một learning rate thấp hơn nhiều ($1e-5$). Việc sử dụng learning rate thấp trong giai đoạn fine-tuning là rất quan trọng để tránh phá hỏng các trọng số tiền huấn luyện quý giá của VGG16, chỉ thực hiện những điều chỉnh nhỏ để chúng thích ứng tốt hơn với đặc thù của dữ liệu rác.
- Các callbacks như EarlyStopping (theo dõi `val_loss`, `patience=10`) và ReduceLROnPlateau (giảm learning rate nếu `val_loss` không cải thiện sau `patience=5`) cũng được áp dụng hiệu quả trong cả hai giai đoạn, giúp tự động hóa việc tìm điểm dừng tối ưu và điều chỉnh tốc độ học.
- **Kết quả Đánh giá Vượt Trội và Phân tích Sâu Sắc hơn về Hiệu năng:**
 - Mô hình CNN tùy chỉnh trong Phiên bản 1 cho thấy validation accuracy chỉ khoảng 62% và biểu đồ loss không ổn định, có dấu hiệu overfitting rõ rệt, cho thấy mô hình chưa học được các đặc trưng đủ tốt hoặc quá phức tạp so với lượng dữ liệu.
 - Ngược lại, mô hình VGG16 fine-tuned trong Phiên bản 2 (dựa trên kết quả từ notebook) đã đạt được validation accuracy khoảng **80.72%**, một sự cải thiện đáng kể. Quá trình huấn luyện cũng cho thấy sự ổn định và hiệu quả hơn nhiều. Biểu đồ accuracy và loss kết hợp từ cả hai giai đoạn huấn luyện minh họa rõ ràng sự cải thiện về cả training và validation metrics, đặc biệt là sự "nhảy vọt" về hiệu suất sau khi bắt đầu giai đoạn fine-tuning, cho thấy các lớp của VGG16 đã thích ứng tốt với dữ liệu mới.
 - Một điểm cộng lớn của Phiên bản 2 là việc bổ sung và phân tích **ma trận nhầm lẫn (confusion matrix)** một cách chi tiết. Ma trận này cung cấp cái nhìn sâu sắc về hiệu suất của mô hình trên từng loại rác cụ thể, không chỉ là một con số accuracy tổng thể. Ví dụ, phân tích cho thấy mô hình hoạt động tốt với lớp "trash" (rác thải thông thường) và "paper" (giấy). Tuy nhiên, có một số nhầm lẫn đáng chú ý giữa "cardboard" (bìa cứng) và "paper", hoặc giữa "plastic" (nhựa) với "glass" (thủy tinh) và "paper". Những nhầm lẫn này có thể xuất phát từ sự tương đồng về hình ảnh, kết cấu hoặc màu sắc giữa các loại vật liệu này trong một số điều kiện ánh sáng hoặc góc chụp nhất định. Phân tích này không chỉ đánh giá mô hình mà còn gợi ý các hướng cải thiện tiềm năng

(ví dụ: thu thập thêm dữ liệu cho các lớp dễ nhầm lẫn, hoặc áp dụng các kỹ thuật xử lý ảnh nâng cao hơn).

II. Cập Nhật Phần Triển Khai Ứng Dụng (Mục 3.2, Phiên bản 2)

Phần mô tả triển khai ứng dụng web cũng được điều chỉnh và làm rõ để phản ánh chính xác những thay đổi trong các mô hình đã được huấn luyện và phương pháp tiếp cận tổng thể.

1. Giới thiệu và Kiến trúc Hệ thống được Mô tả Rõ ràng hơn:

- Phiên bản 2 đã bổ sung các mục giới thiệu tổng quan về mục tiêu của ứng dụng web, đó là cung cấp một giao diện thân thiện cho người dùng để thực hiện phân loại hình ảnh đa mô hình, bao gồm phân loại chim, rác thải và cảm xúc khuôn mặt.
- Kiến trúc hệ thống được mô tả rõ ràng hơn, bao gồm 3 thành phần chính:
 - **Frontend (Web Interface):** Giao diện người dùng được xây dựng bằng HTML, CSS, JavaScript và thư viện Bootstrap để đảm bảo tính thẩm mỹ và khả năng tương tác tốt.
 - **Application Server (ví dụ, sử dụng Flask trong app.py):** Chịu trách nhiệm xử lý các yêu cầu HTTP từ giao diện người dùng (ví dụ: khi người dùng tải ảnh lên), sau đó gọi đến API Server để thực hiện việc phân loại, và cuối cùng là trả kết quả về cho frontend hiển thị.
 - **API Server (ví dụ, sử dụng Flask trong api.py):** Đây là nơi chứa logic xử lý chính, bao gồm việc tải các mô hình học sâu đã huấn luyện, tiền xử lý ảnh đầu vào theo yêu cầu của từng mô hình cụ thể, và thực hiện dự đoán để trả về kết quả phân loại.
- Việc phân tách thành Application Server và API Server (mặc dù trong thực tế có thể chạy trên cùng một máy chủ vật lý) thể hiện một thiết kế hướng module, dễ bảo trì và mở rộng.

2. Cập nhật Quan trọng trong Backend (API - api.py):

- Đây là một cập nhật kỹ thuật quan trọng để đảm bảo tính chính xác của ứng dụng. Hàm `preprocess_image` trong file `api.py` của Phiên bản 2 đã được sửa đổi một cách cẩn thận để:
 - Sử dụng các hàm `preprocess_input` **chuyên biệt** cho từng mô hình học chuyên giao: `tensorflow.keras.applications.resnet50.preprocess_input` cho mô hình phân loại chim (sử dụng ResNet50) và `tensorflow.keras.applications.vgg16.preprocess_input` cho mô hình phân loại rác (sử dụng VGG16). Điều này là tối quan trọng vì mỗi kiến trúc tiền huấn luyện (ResNet50, VGG16) có những yêu cầu chuẩn hóa đầu vào riêng biệt (ví dụ: dải giá trị pixel, thứ tự kênh màu, giá trị trung bình cần trừ) mà chúng đã được huấn luyện cùng trên ImageNet.
 - Điều chỉnh kích thước ảnh đầu vào cho mô hình phân loại rác thành **(224, 224)** pixels trong API, để hoàn toàn khớp với kích thước đầu vào của mô hình VGG16 đã được huấn luyện từ notebook `trainTrashClassify.ipynb`. Điều này khác với kích thước (128, 128) được sử dụng trong Phiên bản 1 cho mô hình phân loại vật liệu cũ.
- Đối với mô hình phân loại cảm xúc (CNN tùy chỉnh), ảnh vẫn được resize về (48, 48), chuyển sang ảnh xám (`convert('L')`) và rescale giá trị pixel về [0, 1] như trong quá trình huấn luyện.

- Những thay đổi này đảm bảo rằng ảnh đầu vào từ người dùng, bất kể kích thước hay định dạng ban đầu, đều được tiền xử lý một cách chính xác và nhất quán với quá trình huấn luyện của từng mô hình cụ thể được chọn. Đây là yếu tố then chốt để đạt được hiệu suất dự đoán tốt nhất và đáng tin cậy trong ứng dụng thực tế.

KẾT LUẬN

Bài tiểu luận đã trình bày một cách hệ thống các kiến thức tổng quan về Trí tuệ nhân tạo, đi sâu vào các khái niệm và kỹ thuật cốt lõi của học sâu, đồng thời thực hiện xây dựng và đánh giá các mô hình ứng dụng cụ thể trong lĩnh vực phân loại hình ảnh. Qua **Chương 1**, bức tranh toàn cảnh về AI đã được phác họa, từ lịch sử phát triển, các định nghĩa nền tảng, cho đến những ứng dụng đa dạng và những thách thức tiềm ẩn. Điều này cho thấy vai trò ngày càng quan trọng và sự thâm nhập sâu rộng của AI vào mọi mặt của đời sống hiện đại. **Chương 2** đã cung cấp một nền tảng lý thuyết vững chắc về học sâu. Các thành phần cơ bản của mạng nơ-ron, cơ chế huấn luyện, các hàm mất mát, thuật toán tối ưu hóa, cùng với các kiến trúc tiêu biểu như Mạng Nơ-ron Tích chập (CNN) và Mạng Nơ-ron Hồi quy (RNN) đã được phân tích. Bên cạnh đó, các kỹ thuật xử lý dữ liệu, điều chuẩn mô hình, và các kiến trúc nâng cao như kết nối thặng dư, chuẩn hóa lô, và kiến trúc Transformer cũng được giới thiệu, làm nổi bật sự phức tạp và tiềm năng to lớn của lĩnh vực này. Phần trọng tâm của tiểu luận, **Chương 3**, đã đi vào thực nghiệm xây dựng ba mô hình phân loại hình ảnh. Mô hình phân loại chim, thông qua việc áp dụng kỹ thuật học chuyển giao với kiến trúc ResNet50, đã đạt được hiệu suất ấn tượng với độ chính xác cao trên tập dữ liệu thử nghiệm. Điều này khẳng định sức mạnh của việc sử dụng các mô hình tiền huấn luyện cho các tác vụ thị giác máy tính. Mô hình phân biệt cảm xúc khuôn mặt, được xây dựng bằng kiến trúc CNN tùy chỉnh và tinh chỉnh với các kỹ thuật như L2 regularization, cũng cho thấy khả năng học và phân loại các biểu cảm ở một mức độ nhất định, mặc dù bài toán này vốn có độ phức tạp cao. Đặc biệt, mô hình phân loại rác thải, được phát triển dựa trên fine-tuning kiến trúc VGG16, đã thể hiện hiệu quả đáng kể với độ chính xác cao và quá trình huấn luyện ổn định. Việc áp dụng quy trình huấn luyện hai giai đoạn (huấn luyện head và fine-tuning) cùng với các hàm tiền xử lý chuyên biệt đã chứng tỏ là một phương pháp hiệu quả. Cuối cùng, việc mô tả triển khai một ứng dụng web tích hợp các mô hình này đã minh họa khả năng đưa các giải pháp học sâu vào ứng dụng thực tế. Kết quả đạt được từ các mô hình, đặc biệt là với các kiến trúc fine-tuning, cho thấy tiềm năng to lớn của học sâu trong việc giải quyết các bài toán phân loại hình ảnh phức tạp. Mặc dù vẫn còn những hạn chế và không gian để cải thiện, các kết quả này là minh chứng cho sự nỗ lực học hỏi và áp dụng kiến thức. Trong tương lai, các hướng phát triển có thể bao gồm việc khám phá các kiến trúc mạng tiên tiến hơn, thử nghiệm với các bộ dữ liệu lớn hơn và đa dạng hơn, cũng như tối ưu hóa các siêu tham số để nâng cao hơn nữa hiệu suất của các mô hình. Đồng thời, việc tìm hiểu sâu hơn về các kỹ thuật giải thích mô hình (Explainable AI - XAI) và các vấn đề đạo đức trong AI cũng là những định hướng quan trọng.