**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**CƠ SỞ TẠI TP.HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN 2**

Logo

Description automatically generated

**BÁO CÁO MÔN HỌC: NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: TÌM HIỂU MẠNG NEURAL NETWORK**

Sinh viên thực hiện: Nguyễn Văn Hậu – MSSV: N23DCCN018

Huỳnh Hoàng Quân – MSSV: N23DCCN184

Đỗ Thị Thanh Thủy – MSSV: N23DCCN197

Lớp: D22CQDTVM

Giáo viên hướng dẫn: Hồ Nhựt Minh

*TP.Hồ Chí Minh, ngày 16 tháng 10 năm 2025*

Mục lục

[1. Giới thiệu 5](#_Toc217508953)

[1.1. Bối cảnh 5](#_Toc217508954)

[1.2. Mục tiêu báo cáo 7](#_Toc217508955)

[1.3. Phạm vi 7](#_Toc217508956)

[2. Cơ sở lý thuyết 8](#_Toc217508957)

[2.1. TensorFlow 2 - Eager Execution, tf.keras, tf.data 8](#_Toc217508958)

[2.1.1.Tổng quan TensorFlow 2 8](#_Toc217508959)

[2.1.2. Eager Execution 9](#_Toc217508960)

[2.1.3. tf.keras API 10](#_Toc217508961)

[2.1.4. tf.data Pipeline 10](#_Toc217508962)

[2.2. Neural Networks Fundamentals 12](#_Toc217508963)

[2.2.1. Cấu trúc mạng neuron 12](#_Toc217508964)

[2.2.2. Forward Propagation 13](#_Toc217508965)

[2.2.3. Backpropagation 13](#_Toc217508966)

[2.2.4. Các thuật toán tối ưu hóa (Optimizers) phổ biến: 14](#_Toc217508967)

[2.3. Phân loại Nhị phân (Binary Classification) – Dữ liệu Pokémon 14](#_Toc217508968)

[2.3.1. Mô tả Bài toán 14](#_Toc217508969)

[2.3.2. Hàm mất mát Binary Cross-Entropy (BCE) 14](#_Toc217508970)

[2.3.3. Mã hóa Dữ liệu: So sánh Integer Encoding và One-hot Encoding 15](#_Toc217508971)

[2.4. Multi-class classification & CNN basics (softmax, categorical crossentropy, Conv2D, pooling) 17](#_Toc217508972)

[2.4.1. Softmax và Categorical Cross-Entropy 17](#_Toc217508973)

[2.4.2. Các Khối Xây Dựng CNN 18](#_Toc217508974)

[2.4.3. Các Phép Toán Tăng cường Dữ liệu (Data Augmentation Ops) 21](#_Toc217508975)

[2.4.4. Cấu trúc cơ bản cho CNN CIFAR-10 24](#_Toc217508976)

[2.5 Training techniques — Backpropagation, Weight initialization, Batch Normalization 27](#_Toc217508977)

[2.5.1.Mô tả backpropagation (chain rule) và công thức cập nhật. 27](#_Toc217508978)

[2.5.2. Phương pháp Normal, Glorot/Xavier và He Initialization : 28](#_Toc217508979)

[2.5.3. Kĩ thuật chuẩn hóa đầu vào Batch Normalization (mean/variance normalization, gamma & beta). 30](#_Toc217508980)

[2.6 Advanced TensorFlow (custom layer/loss/metric/callback) & Advanced TensorBoard (tf.summary, HParams) 32](#_Toc217508981)

[2.6.1 Khi nào cần viết custom Layer (subclass tf.keras.layers.Layer)? 32](#_Toc217508982)

[2.6.2 Khi nào cần viết custom loss ? 32](#_Toc217508983)

[2.6.3 Khi nào cần custom metric ? 32](#_Toc217508984)

[2.6.4 khi nào cần viết custom callback ? 32](#_Toc217508985)

[2.6.5 Dùng tf.summary để log và bật TensorBoard 32](#_Toc217508986)

[2.6.6 HParams dashboard & khuyến nghị thực nghiệm 33](#_Toc217508987)

[3. Phương pháp thực hiện 35](#_Toc217508988)

[3.1. Tổng quan Quy trình Thực nghiệm 35](#_Toc217508989)

[3.2. Bộ Dữ liệu (Datasets) 36](#_Toc217508990)

[3.3. Tiền xử lý dữ liệu và pipeline 36](#_Toc217508991)

[3.3.1. Tiền xử lý dữ liệu (Preprocessing) 36](#_Toc217508992)

[3.3.2. Xây dựng pipeline bằng tf.data 37](#_Toc217508993)

[3.3.3. Tăng cường Dữ liệu (Data Augmentation) 38](#_Toc217508994)

[3.4. Kiến trúc Mô hình và Thiết lập Huấn luyện 38](#_Toc217508995)

[3.4.1. Thí nghiệm Phân loại Nhị phân (Pokémon) 38](#_Toc217508996)

[3.4.2. Thí nghiệm So sánh Kỹ thuật Huấn luyện (CIFAR-10) 39](#_Toc217508997)

[4. Kết quả thực nghiệm 40](#_Toc217508998)

[4.1. Môi trường và Metrics đánh giá 40](#_Toc217508999)

[4.1.1. Môi trường thực nghiệm 40](#_Toc217509000)

[4.1.2. Metrics đánh giá: 41](#_Toc217509001)

[4.2. Kết quả thí nghiệm 1: Phân loại nhị phân (Dữ liệu Pokemon) 41](#_Toc217509002)

[4.2.1. So sánh Integer Encoding (Model-1) và One-hot Encoding (Model-2): 41](#_Toc217509003)

[4.3. Kết quả thí nghiệm 2: Phân loại đa lớp và CNN cơ bản (CIFAR-10) 42](#_Toc217509004)

[4.3.1. Chi tiết kiến trúc 42](#_Toc217509005)

[4.3.2. Phân tích kết quả 44](#_Toc217509006)

[4.4. Kết quả thí nghiệm 3: Khởi tạo trọng số (Weight Initialization) : Normal, Glorot/Xavier, He 46](#_Toc217509007)

[Weight Initialization 48](#_Toc217509008)

[4.4.1. Phân tích kết quả khởi tạo trọng số 49](#_Toc217509009)

[4.5. Thí nghiệm 4 : Kiểm chứng batch normalization 56](#_Toc217509010)

[4.5.1. Phân tích tác động của Batch Normalization (Model-4) 57](#_Toc217509011)

[4.6. Thí nghiệm 5 : Triển khai 2 mô hình sử dụng 2 API khác nhau (keras API cấp cao và API tùy chỉnh) 57](#_Toc217509012)

[4.7. Thí nghiệm 6: Sử dụng tf.summary.image API 59](#_Toc217509013)

[4.8. Thí nghiệm 7: Điều chỉnh siêu tham số với TensorBoard HParams 60](#_Toc217509014)

[4.9. Thí nghiệm 8: Triển khai mạng Inception-v3 (212) 63](#_Toc217509015)

[5. Thảo luận : 64](#_Toc217509016)

[6. Kết luận và hướng phát triển 67](#_Toc217509017)

[7. Tài liệu tham khảo 68](#_Toc217509018)

Phụ lục ảnh

[Hình 1: Ví dụ minh họa cơ chế Eager Execution trong TensorFlow 2 — các phép toán được thực thi ngay khi khai báo, trả về kết quả trực tiếp dưới dạng tf.Tensor 10](#_Toc217508802)

[Hình 2: Ví dụ xây dựng mô hình mạng nơ-ron bằng tf.keras.Sequential API 11](#_Toc217508803)

[Hình 3: Ví dụ xây dựng mô hình mạng nơ-ron bằng tf.keras.Functional API 11](#_Toc217508804)

[Hình 4: Pipeline xử lý dữ liệu bằng tf.data trong TensorFlow 2 12](#_Toc217508805)

[Hình 5: Ví dụ minh họa pipeline dữ liệu đơn giản trong TensorFlow 2 sử dụng tf.data.Dataset 12](#_Toc217508806)

[Hình 6:Cấu trúc mạng neuron 13](#_Toc217508807)

[Hình 7: So sánh các thuật toán tối ưu hóa phổ biến trong huấn luyện mạng nơ-ron nhân tạo. 15](#_Toc217508808)

[Hình 8: So sánh Integer vs One-hot Encoding. 17](#_Toc217508809)

[Hình 9. Ví dụ về các phương pháp tăng cường dữ liệu 22](#_Toc217508810)

[Hình 10. Một đoạn code ví dụ cho việc lật ảnh 22](#_Toc217508811)

[Hình 11. Ví dụ về Image Shifting 23](#_Toc217508812)

[Hình 12. Ví dụ cho Color Conversion 23](#_Toc217508813)

[Hình 13 Ví dụ cho làm mờ ảnh (Blur Image) hoặc thêm nhiễu (add noise) 24](#_Toc217508814)

[Hình 14 Sơ đồ tính toán lan truyền ngược (Backpropagation - BP) của mạng nơ-ron một lớp. 28](#_Toc217508815)

[Hình 15 Bảng điều khiển HParams ở chế độ hiển thị song song 34](#_Toc217508816)

[Hình 16 Table View – Bảng so sánh kết quả từng run 34](#_Toc217508817)

[Hình 17: Kết quả thí nghiệm 2: Phân loại đa lớp và CNN cơ bản (CIFAR-10) 42](#_Toc217508818)

[Hình 18 Kết quả của accuracy và loss trên tập train của 3 loại model (FCNN – CNN – CNN với data augmentation) trong bài toán phân loại đa lớp CIFAR-10 44](#_Toc217508819)

[Hình 19 Kết quả của accuracy và loss trên tập validation test của 3 loại model (FCNN – CNN – CNN với data augmentation) trong bài toán phân loại đa lớp CIFAR-10 45](#_Toc217508820)

[Hình 20: DISTRIBUTIONS cho thấy sự thay đổi của trọng số của Model-1 khi sử dụng phân phối chuẩn (μ = 0 và σ = 0.01). 50](#_Toc217508821)

[Hình 21 HISTOGRAMS minh họa sự thay đổi của trọng số của Model-1 khi sử dụng phân phối chuẩn (μ = 0 và σ = 0.01). 51](#_Toc217508822)

[Hình 22 Sử dụng công cụ DISTRIBUTIONS để quan sát sự thay đổi trọng số của Model-2 (Glorot) 52](#_Toc217508823)

[Hình 23 Using HISTOGRAMS tool 53](#_Toc217508824)

[Hình 24 Sử dụng công cụ DISTRIBUTIONS cho thấy sự thay đổi của trọng số của Model-3 khi sử dụng khởi tạo He 54](#_Toc217508825)

[Hình 25 cho thấy sự thay đổi của trọng số của lớp convolution thứ nhất và thứ hai, được quan sát bằng cách sử dụng công cụ HISTOGRAMS. 55](#_Toc217508826)

[Hình 26 Bảng so sánh hiệu suất của 4 models 57](#_Toc217508827)

[Hình 27 : Kết quả confusion matrix được trực quan hóa trên images dashboard. 59](#_Toc217508828)

[Hình 28 : Thông tin chi tiết các siêu tham số có thể được sử dụng trong các models 60](#_Toc217508829)

[Hình 29 TensorBoard HParams chứa thông tin 36 models vừa được huấn luyện 61](#_Toc217508830)

[Hình 30 : Quan sát toàn bộ kết quả (36 models) bằng Parallel Coordinates View. 61](#_Toc217508831)

[Hình 31 : làm mờ các model có độ chính xác không đạt yêu cầu (0.85) 62](#_Toc217508832)

[Hình 32 TensorBoard HParams chiếu các model có độ chính xác đạt yêu cầu (0.85) 62](#_Toc217508833)

[Hình 33 Kết quả dự đoán 63](#_Toc217508834)

[Hình 34 Kết quả dự đoán voi Châu Phi 64](#_Toc217508835)

[Hình 35 Kết quả dự đoán xác xuất cao nhất 64](#_Toc217508836)

[Hình 36 tusker, voi Ấn Độ, voi Châu Phi 64](#_Toc217508837)

[Hình 37 Bảng hiển thị kết quả độ chính xác và sự mất mát của 3 models 65](#_Toc217508838)

[Hình 38: Lưu đồ của source code cho việc xây dụng mô hình mạng và trược quan hóa kết quả huấn luyện mô hình 75](#_Toc217508839)

[Hình 39 : Kết quả confusion matrix được trực quan hóa trên images dashboard. 75](#_Toc217508840)

[Hình 40: Thông tin chi tiết các siêu tham số có thể được sử dụng trong các models 76](#_Toc217508841)

[Hình 41 TensorBoard HParams chứa thông tin 36 models vừa được huấn luyện 77](#_Toc217508842)

[Hình 42 : Quan sát toàn bộ kết quả (36 models) bằng Parallel Coordinates View. 77](#_Toc217508843)

[Hình 43 : làm mờ các model có độ chính xác không đạt yêu cầu (0.85) 78](#_Toc217508844)

[Hình 44 TensorBoard HParams chiếu các model có độ chính xác đạt yêu cầu (0.85) 78](#_Toc217508845)

[Hình 45 Kết quả dự đoán 79](#_Toc217508846)

[Hình 46 Kết quả dự đoán voi Châu Phi 80](#_Toc217508847)

[Hình 47 Kết quả dự đoán xác xuất cao nhất 80](#_Toc217508848)

[Hình 48 tusker, voi Ấn Độ, voi Châu Phi 80](#_Toc217508849)

Phụ lục bảng

[Bảng 1 Các khối xây dựng CNN 22](#_Toc216890368)

[Bảng 2 Khối đầu vào 26](#_Toc216890369)

[Bảng 3 Mô tả các lớp đặc trưng 27](#_Toc216890370)

[Bảng 4 Mô tả bộ phân loại 27](#_Toc216890371)

[Bảng 5 Mô tả bộ dữ liệu 37](#_Toc216890372)

[Bảng 6 Thí nghiệm Phân loại nhị phân 40](#_Toc216890373)

[Bảng 7 Thí nghiệm so sánh kỹ thuật huấn luyện 41](#_Toc216890374)

[Bảng 8 So sánh 3 mô hình chính 44](#_Toc216890375)

[Bảng 9 So sánh các model trên tập validation (kết quả từ Lab 4) 47](#_Toc216890376)

[Bảng 10 So sánh các Phương pháp khởi tạo trọng số và Batch Normalization (CIFAR-10) 48](#_Toc216890377)

[Bảng 11 Kiến trúc của các mô hình phân loại ảnh CIFAR-10 62](#_Toc216890378)

[Bảng 12 Kết quả đo độ chính xác và sự mất mát của các model 63](#_Toc216890379)

# Giới thiệu

## Bối cảnh

Trong kỷ nguyên trí tuệ nhân tạo (AI), **Deep Learning** đã trở thành nền tảng quan trọng giúp máy tính có khả năng tự học và mô phỏng các hành vi của con người. So với các kỹ thuật truyền thống như SVM hay cây quyết định, Deep Learning nổi bật nhờ khả năng tự trích xuất đặc trưng (feature learning) từ dữ liệu thô, cho phép đạt hiệu suất cao trong các bài toán như thị giác máy tính, xử lý ngôn ngữ tự nhiên hay nhận dạng giọng nói. Theo *Huang & Le (2021)*, “deep learning is the most popular because it has been widely and successfully applied to various application areas such as computer vision, speech recognition, natural language processing, and so on” (Ch.1, p.24).

Trong những năm gần đây, trí tuệ nhân tạo (AI) và học sâu (DL) đã không chỉ là lĩnh vực nghiên cứu mà còn trở thành nền tảng công nghệ chủ đạo trong nhiều ngành công nghiệp.

Theo báo cáo của Fortune Business Insights (2024), quy mô thị trường AI toàn cầu đạt khoảng 233.5 tỷ USD, với tốc độ tăng trưởng trung bình hàng năm (CAGR) trên 36% giai đoạn 2024-2030.

Riêng mảng **Deep Learning** được ước tính đạt **24,5 tỷ USD năm 2024**, và được dự đoán sẽ tăng trưởng mạnh mẽ trong những năm tiếp theo.  
Những con số này cho thấy AI/DL đang trở thành động lực chính thúc đẩy chuyển đổi số toàn cầu.

Nhờ sự phát triển mạnh mẽ của phần cứng (GPU, TPU) và các thư viện mã nguồn mở, Deep Learning hiện nay không chỉ giới hạn trong nghiên cứu mà còn được ứng dụng rộng rãi trong công nghiệp, tự động hóa, y học và giải trí.

Về mặt nghiên cứu, tốc độ phát triển của lĩnh vực AI/DL thể hiện rõ qua số lượng công bố khoa học.

Theo thống kê trên nền tảng *arXiv.org*, số lượng bài báo thuộc các chuyên mục *Artificial Intelligence (cs.AI)* và *Machine Learning (cs.LG)* tăng liên tục mỗi năm, đạt hơn **33.000 bài trong năm 2024**, gấp nhiều lần so với năm 2018.  
Nhiều phân tích chỉ ra tốc độ tăng trưởng này **gần như nhân đôi sau mỗi 2 năm**, phản ánh sự bùng nổ nghiên cứu và đổi mới trong cộng đồng học thuật.

Trong số các thư viện Deep Learning phổ biến, **TensorFlow 2** được phát triển bởi nhóm Google Brain là một nền tảng mạnh mẽ, dễ mở rộng, và hỗ trợ đầy đủ các công cụ từ huấn luyện, đánh giá đến triển khai mô hình trên nhiều thiết bị. TensorFlow 2 tích hợp **Eager Execution** giúp lập trình theo phong cách mệnh lệnh (imperative), thuận tiện cho việc gỡ lỗi và thử nghiệm. Ngoài ra, **tf.keras** cung cấp API cấp cao cho phép xây dựng mô hình nhanh chóng, trong khi **tf.data** hỗ trợ xử lý pipeline dữ liệu hiệu quả, tận dụng tối đa tài nguyên tính toán.  
Việc chọn TensorFlow 2 làm công cụ thực hành giúp sinh viên có thể học và làm việc với một hệ sinh thái chuẩn công nghiệp, phù hợp cho các bài toán phân loại, hồi quy và nhận dạng phức tạp.

Không chỉ trong nghiên cứu, AI còn được ứng dụng rộng rãi trong các doanh nghiệp hàng đầu thế giới.

Theo khảo sát của *Gallup và McKinsey (2024)*, hơn **90% các công ty trong nhóm Fortune 500** cho biết đã triển khai hoặc thử nghiệm các giải pháp AI trong hoạt động của mình — từ phân tích dữ liệu, quản lý chuỗi cung ứng, cho đến tự động hóa dịch vụ khách hàng.  
Điều này cho thấy AI/DL đã trở thành công nghệ chiến lược, đóng vai trò thiết yếu trong năng lực cạnh tranh và đổi mới sáng tạo của doanh nghiệp hiện đại.

## Mục tiêu báo cáo

Báo cáo này tập trung vào **ba nhóm thí nghiệm chính**, nhằm mục tiêu hiểu rõ quy trình xây dựng và huấn luyện mô hình Deep Learning với TensorFlow 2:

1. **Xây dựng và so sánh** mô hình **phân loại nhị phân (binary classification)** – ví dụ: phân biệt Pokémon Legendary và Non-Legendary.
2. **Phân loại đa lớp (multi-class classification)** – mở rộng từ bài toán nhị phân để xử lý nhiều nhóm dữ liệu.
3. **Thí nghiệm với mạng CNN** – đánh giá ảnh hưởng của **khởi tạo trọng số (Glorot/Xavier, He)**, **chuẩn hóa Batch Normalization**, và quan sát qua TensorBoard.

## Phạm vi

Báo cáo dựa trên nội dung từ **Chương 1 đến Chương 8** của sách *Principles and Labs for Deep Learning* (Elsevier, 2021), bao gồm:

- Giới thiệu TensorFlow 2, tf.keras, tf.data.

- Cơ sở lý thuyết mạng neural và tối ưu hóa.

- Phân loại nhị phân, đa lớp và mạng CNN.

Ý nghĩa của đề tài

Đề tài có ý nghĩa quan trọng trong việc xây dựng nền tảng kiến thức và kỹ năng thực tiễn về Học sâu:

1. **Áp dụng Công cụ Thực tế:** Cung cấp kinh nghiệm thực tế trong việc sử dụng nền tảng **TensorFlow**, bộ công cụ mã nguồn mở và phổ biến nhất trong cộng đồng học sâu, giúp người học sẵn sàng áp dụng vào các dự án thực tế

2. **Hiểu sâu sắc về Cơ chế Huấn luyện:** Giúp người học không chỉ biết cách xây dựng mô hình mà còn hiểu rõ các yếu tố ảnh hưởng đến quá trình hội tụ và hiệu suất của mạng thần kinh sâu, đặc biệt là tầm quan trọng của các kỹ thuật **Khởi tạo Trọng số** (như He và Glorot) và **Chuẩn hóa Hàng loạt** trong việc giải quyết vấn đề gradient biến mất (vanishing gradient) và tăng tốc độ huấn luyện.

3. **Kỹ năng Phân tích và Tối ưu:** Rèn luyện kỹ năng phân tích sâu (ví dụ: sử dụng TensorBoard để theo dõi phân bố trọng số và điều chỉnh siêu tham số), từ đó cho phép tạo ra các mô hình hiệu suất cao hơn (ví dụ: CNN với Batch Normalization và He initialization đạt độ chính xác cao nhất trong các mô hình so sánh)

# 2. Cơ sở lý thuyết

## 2.1. TensorFlow 2 - Eager Execution, tf.keras, tf.data

### 2.1.1.Tổng quan TensorFlow 2

TensorFlow 2 là một nền tảng mã nguồn mở mạnh mẽ được phát triển bởi nhóm Google Brain, dùng để xây dựng, huấn luyện và triển khai các mô hình học máy và học sâu. TensorFlow hoạt động dựa trên đồ thị luồng dữ liệu (dataflow graph), trong đó mỗi nút biểu diễn một phép toán và mõi cạnh biểu diễn dòng dữ liệu (tensor)

Khác với TensorFlow 1, phiên bản 2 được cải tiến toàn diện nhằm đơn giản hóa lập trình, tăng tốc quá trình huấn luyện và tối ưu khả năng triển khai trên nhiều thiết bị khác nhau.

Các cải tiến nổi bật của TensorFlow 2 gồm:

- Eager Execution: cho phép thực thi lệnh ngay lập tức (thay vì xây đồ thị và chạy trong session), giúp dễ gỡ lỗi và trực quan hơn.

- tf.keras: cung cấp giao diện cấp cao để xây dựng mạng nơ-ron một cách nhanh chóng và dễ hiểu.

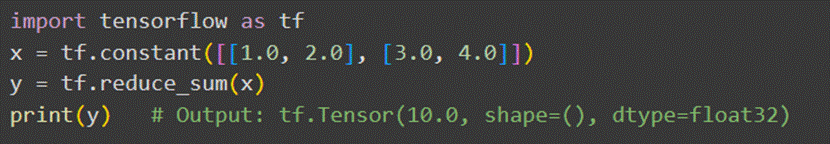
- tf.data: cung cấp API mạnh mẽ để xử lý dữ liệu đầu vào, xây dựng pipeline tối ưu cho huấn luyện.

- TensorFlow Hub, SavedModel, Distribution Strategy: hỗ trợ tải trọng số, lưu mô hình và huấn luyện phân tán.

Nhờ sự kết hợp của các thành phần này, TensorFlow cho phép người dùng thiết kế mô hình, huấn luyện, lưu trữ và triển khai một cách thống nhất trên CPU, GPU hoặc TPU.

### 2.1.2. Eager Execution

Khác với Tensorflow 1.x yêu cầu định nghĩa graph rồi “chạy session”, TensorFlow 2 thực thi lệnh ngay lập tức:



Hình 1: Ví dụ minh họa cơ chế Eager Execution trong TensorFlow 2 — các phép toán được thực thi ngay khi khai báo, trả về kết quả trực tiếp dưới dạng tf.Tensor

Eager Execution giúp:

- Debug dễ hơn (thấy kết quả tức thời)

- Thử nghiệm nhanh, đặc biệt trong notebook như Colab

- Kết hợp tự nhiên với Python control flow (if, while, list comprehension...)

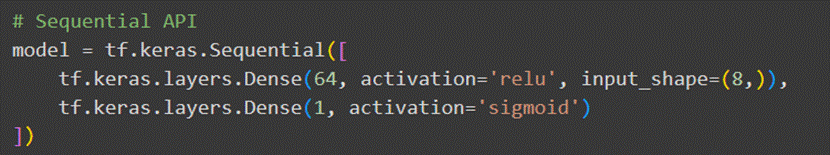
### 2.1.3. tf.keras API

***tf.keras*** là API cấp cao tích hợp sẵn trong TensorFlow 2, cho phép xây dựng mạng nơ-ron theo hai cách phổ biến:

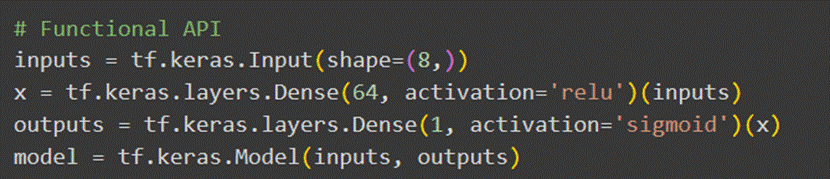
- Sequential API: dùng để xếp các tầng (layer) nối tiếp nhau, thích hợp với các mô hình đơn gianr các đầu vào và đầu ra duy nhất

- Functional API: linh hoạt hơn, cho phép tạo mô hình có nhiều đầu vào/đầu ra, hoặc chia sẻ tầng giữa các nhánh mạng.

Ví dụ:



Hình 2: Ví dụ xây dựng mô hình mạng nơ-ron bằng tf.keras.Sequential API



Hình 3: Ví dụ xây dựng mô hình mạng nơ-ron bằng tf.keras.Functional API

### 2.1.4. tf.data Pipeline

***tf.data*** là một API chuyên dụng để xây dựng pipeline dữ liệu - một yếu tố quan trọng quyết định hiệu quả của huấn luyện. Khi huấn luyện mô hình, việc đọc dữ liệu trực tiếp từ bộ nhớ có thể gây tắc nghẽn CPU hoặc GPU nếu không được tối ưu. tf.data giúp giải quyết vấn đề này bằng cách cho phép:

- Extraction: đọc dữ liệu từ nhiều nguồn (CSV, ảnh, TFRecord,...)

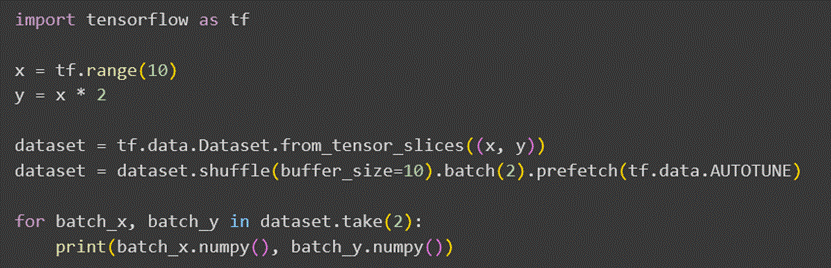
- Transformation: chuyển đổi, shuffle, batch, map và augment dữ liệu.

- **Loading:** tải dữ liệu song song và prefetch vào bộ nhớ GPU để tránh “đói dữ liệu” khi huấn luyện.

A diagram of a process

AI-generated content may be incorrect.

Hình 4: Pipeline xử lý dữ liệu bằng tf.data trong TensorFlow 2



Hình 5: Ví dụ minh họa pipeline dữ liệu đơn giản trong TensorFlow 2 sử dụng tf.data.Dataset

Kết quả là dữ liệu được chia batch cùng với tf.keras và tf.data tạo thành bộ công cụ hoàn chỉnh cho việc xây dựng, huấn luyện và triển khai các mô hình học sâu. Trong các phần tiếp theo, pipeline tf.data sẽ được áp dụng trực tiếp vào thí nghiệm Binary Classification (Pokemon) để xử lý dữ liệu và tối hưu hiệu năng huấn luyện.

## 2.2. Neural Networks Fundamentals

### 2.2.1. Cấu trúc mạng neuron

Một mạng nơ-ron nhân tạo thông thường bao gồm ba thành phần chính:

1. Input layer: Nhập dữ liệu đầu vào, thường là vector đặc trưng.

2. Hidden layer: Bao gồm các lớp nằm giữa lớp đầu vào và lớp đầu ra, nơi thực hiện các phép tính toán chính. Mỗi neuron trong lớp ẩn thường áp dụng một hàm kích hoạt (activation function) như **ReLU (Rectified Linear Unit)**, **Sigmoid**, hoặc **Tanh** ...

3. Output layer: Trả về giá trị dự đoán cuối cùng của mạng (có thể là nhị phân hoặc đa lớp)

Mỗi neuron (trừ neuron đầu vào) nhận một hoặc nhiều đầu vào với trọng số () riêng biệt. Các đầu vào này được tổng hợp (cùng với độ lệch ), sau đó truyền qua hàm phi tuyến tính (hàm kích hoạt) để tạo ra đầu ra.

A diagram of a neural network

AI-generated content may be incorrect.

Hình 6:Cấu trúc mạng neuron

### 2.2.2. Forward Propagation

Forward Propagation là quá trình tính toán đầu ra của mạng nơ-ron. Quá trình này bắt đầu từ input layer di chuyển qua các hidden layers, cho đến khi đạt đến output layer.

Ở mỗi lớp, đầu ra của lớp trước là đầu vào của lớp sau. Giá trị tổng hợp và đầu ra kích hoạt của một neuron được tính như sau:

Trong đó:

* là các đầu vào từ lớp trước
* là các trọng số tương ứng
* là độ lệch pha (bias)
* là hàm kích hoạt phi tuyến (ví dụ: ReLU, Sigmoid)

### 2.2.3. Backpropagation

Backpropagation (BP) là phương pháp quan trọng để tìm kiếm tập hợp trọng số (W) phù hợp, nhằm giảm thiểu hàm mất mát (L)

BP là sự kết hợp giữa thuật toán Gradient Descent (GD) và việc tính toán gradient. Quá trình này tính toán đạo hàm của hàm mất mát theo từng trọng số trong mạng bằng cách sử dụng quy tắc chuỗi (chain rule)

Cụ thể, sai số được lan truyền ngược từ đầu ra -> lớp ẩn -> lớp đầu vào. Các trọng số sau đó được điều chỉnh theo hướng ngược lại với gradient để giảm thiểu sai số.

Công thức cập nhật trọng số cơ bản trong Gradient Descent là:

Trong đó:

* là trọng số
* là tốc độ học (learning rate), xác định kích thước bước cập nhật trọng số.
* là gradient của hàm mất mát L tại điểm W.

### 2.2.4. Các thuật toán tối ưu hóa (Optimizers) phổ biến:

|  |  |  |
| --- | --- | --- |
| Optimizer | Đặc điểm | Khi dùng |
| SGD | Cập nhật gradient từng batch | Dữ liệu nhỏ, đơn giản |
| Adam | Tự điều chỉnh learning rate | Phổ biến, hiệu quả cho hầu hết task |
| RMSProp | Tối ưu dựa trên trung bình bình phương gradient | Cho dữ liệu phi tuyến |

Hình 7: So sánh các thuật toán tối ưu hóa phổ biến trong huấn luyện mạng nơ-ron nhân tạo.

## 2.3. Phân loại Nhị phân (Binary Classification) – Dữ liệu Pokémon

### 2.3.1. Mô tả Bài toán

**Phân loại nhị phân (Binary Classification)** là một dạng cơ bản trong học máy, trong đó mô hình được huấn luyện để **dự đoán một trong hai lớp đầu ra**. Ví dụ điển hình bao gồm phân biệt bệnh hay không bệnh, nam hay nữ, hoặc thắng hay thua

Trong thí nghiệm sử dụng dữ liệu Pokémon, mục tiêu là xây dựng mô hình mạng nơ-ron để phân biệt giữa Pokémon thuộc nhóm **Legendary** (nhãn 1) và **Non-Legendary** (nhãn 0). Mỗi Pokémon được mô tả bằng các thuộc tính như HP, Attack, Defense, Speed, Sp. Attack, Sp. Defense, Generation, v.v..2.3.1. Binary Classification

### 2.3.2. Hàm mất mát Binary Cross-Entropy (BCE)

Đối với bài toán phân loại nhị phân, hàm mất mát tiêu chuẩn được sử dụng là **Binary Cross-Entropy (BCE)**

**Mục đích:** BCE được sử dụng để đo lường sự sai khác (sai số) giữa **xác suất dự đoán** của mô hình và **nhãn thực tế**.

**Cấu tạo:** BCE là sự kết hợp của hàm kích hoạt **Sigmoid** và hàm Cross-Entropy (còn được gọi là Sigmoid Cross-Entropy Loss). Hàm Sigmoid được áp dụng ở tầng đầu ra của mạng để chuyển đổi điểm số thô thành **xác suất** thuộc lớp 1 (Legendary), với giá trị đầu ra nằm trong khoảng từ 0 đến 1.

**Công thức BCE:**

Trong đó:

- là nhãn thực tế của mẫu thứ (0 hoặc 1)

- là giá trị dự đoán của mạng (trước hàm kích hoạt Sigmoid)

- là xác suất dự đoán sau khi qua hàm Sigmoid

- là số lượng mẫu dữ liệu trong một batch

Lưu ý: Nếu giá trị dự đoán () của mô hình khớp với đầu ra mong muốn (), giá trị mất mát (Loss) sẽ tiến về 0 [216, 217f]. Ngược lại, nếu dự đoán sai, giá trị Loss sẽ tăng lên

### 2.3.3. Mã hóa Dữ liệu: So sánh Integer Encoding và One-hot Encoding

Trong các bài toán phân loại, dữ liệu đầu vào thường là dữ liệu phân loại (categorical data). Mạng nơ-ron không thể xử lý trực tiếp dữ liệu này, vì vậy chúng cần được chuyển đổi sang dạng số. Hai phương pháp mã hóa phổ biến là Integer Encoding và One-hot Encoding.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tiêu chí | Integer Encoding (Mã hóa số nguyên) | |  |  | | --- | --- | |  | One-hot Encoding (Mã hóa One-hot) | |
| **Cách thức** | Gán một số nguyên cho mỗi danh mục (ví dụ: 'Mèo' = 1, 'Chó' = 2) | Mỗi danh mục được chuyển thành một vector nhị phân, với giá trị '1' tại vị trí tương ứng với danh mục đó |
| **Mối quan hệ** | Phù hợp khi các danh mục có **mối quan hệ thứ tự tự nhiên** (ordinal relationship) (ví dụ: "nhẹ", "trung bình", "nặng") | Giải quyết vấn đề thứ tự giả định, do đó phù hợp cho các danh mục không có thứ tự tự nhiên |
| **Nhược điểm** | Khi không có thứ tự tự nhiên (ví dụ: "bàn" và "người"), việc gán số nguyên khiến mô hình giả định một thứ tự, dẫn đến hiệu suất kém | Làm tăng kích thước của vector đầu vào (tăng chiều dữ liệu) |

Hình 8: So sánh Integer vs One-hot Encoding.

So sánh hiệu quả thực nghiệm:

Trong thí nghiệm phân loại nhị phân (dự đoán chiến đấu Pokémon), việc mã hóa các thuộc tính (ví dụ: Type 1, Type 2) đã được thực hiện bằng cả hai phương pháp

**- Model-1** (sử dụng **Integer Encoding**) có kích thước đầu vào (20,). Độ chính xác đạt được trên tập kiểm thử là 0.9466

**- Model-2 (sử dụng One-hot Encoding) có kích thước đầu vào lớn hơn (54,). Độ chính xác đạt được trên tập kiểm thử là 0.9654**

**Kết luận:** Kết quả này chứng minh rằng **One-hot Encoding** hiệu quả hơn Integer Encoding trong bài toán phân loại nhị phân (ví dụ: dự đoán chiến đấu Pokémon). Nguyên nhân là do One-hot Encoding tránh được việc gán thứ tự giả định cho các danh mục không liên quan, giúp mô hình học các đặc trưng một cách độc lập và chính xác hơn

## 2.4. Multi-class classification & CNN basics (softmax, categorical crossentropy, Conv2D, pooling)

### 2.4.1. Softmax và Categorical Cross-Entropy

Hai thành phần này là **hàm kích hoạt lớp cuối cùng** và **hàm mất mát**, chúng hoạt động cùng nhau để cho phép phân loại đa danh mục.

#### Hàm Softmax

* **Mục đích:** Hàm Softmax là hàm kích hoạt cuối cùng được sử dụng trong lớp cuối cùng của mạng được thiết kế để phân loại đa danh mục.
* **Chức năng:** Nó chuyển đổi các điểm thô (logits) từ lớp dày đặc (dense) cuối cùng thành một tập hợp **xác suất**.
* **Đặc tính chính:** Các giá trị đầu ra nằm trong khoảng từ 0 đến 1, và **tổng của tất cả các xác suất đầu ra được đảm bảo bằng 1**.
* **Công thức:**

Trong đó:

là xác suất dự đoán cho lớp

là điểm đầu vào cho lớp

là tổng số danh mục.

#### Hàm mất mát Cross-Entropy phân loại (CCE)

* **Mục đích:** CCE là **hàm mất mát** tiêu chuẩn được sử dụng cho các bài toán phân loại đa danh mục, trong đó mỗi mẫu chỉ thuộc về duy nhất một lớp.
* **Chức năng:** Nó đo lường **sai số** (hoặc sự khác biệt) giữa phân phối xác suất dự đoán của mô hình () và nhãn mã hóa one-hot thực tế ().
* **Thành phần:** CCE là sự kết hợp của **hàm Softmax** và **hàm Cross-Entropy (CE)**, đó là lý do tại sao đôi khi nó được gọi là **Softmax Loss**.
* **Công thức:**

trong đó :

* là hàm Softmax.
* *N* là Số lượng sample trong tập dữ liệu hoặc mini-batch
* *C* Số lượng class
* Giá trị nhãn thật của sample thứ tại class
* Xác suất dự đoán của mô hình

### 2.4.2. Các Khối Xây Dựng CNN

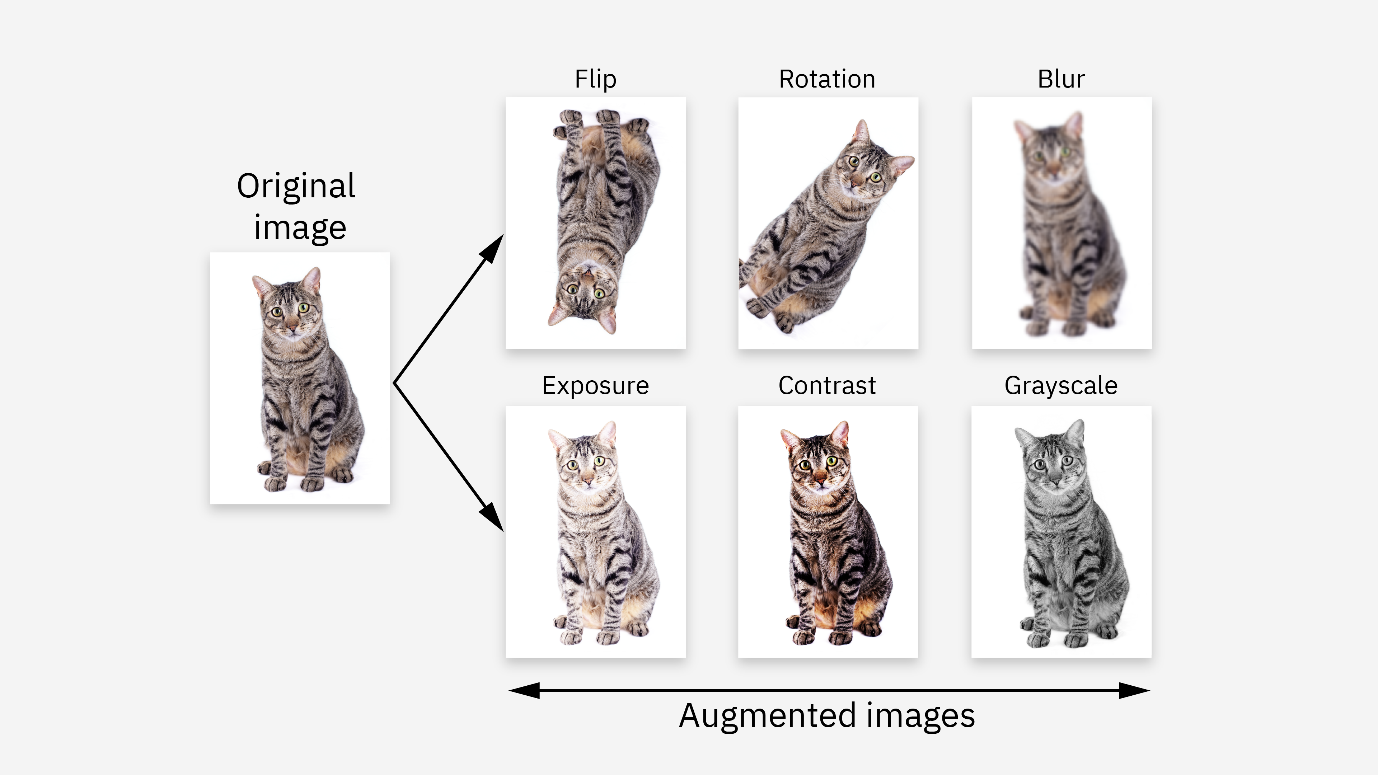
Một CNN chủ yếu bao gồm một **Bộ trích xuất Đặc trưng** (Các lớp Tích chập và Gộp) và một **Bộ phân loại** (Các lớp Liên kết đầy đủ).

| Thành phần | Mô tả | Tham số / Khái niệm Chính |
| --- | --- | --- |
| **Lớp Conv2D** | Thành phần chính để trích xuất đặc trưng trong CNN. Nó học cách nhận dạng các đặc trưng như cạnh, đường thẳng và kết cấu phức tạp trong ảnh bằng cách áp dụng một hạt nhân lên đầu vào. | **Hạt nhân/Bộ lọc (Kernel/Filter):** Một ma trận trọng số nhỏ trượt qua ảnh đầu vào để trích xuất đặc trưng. |
| **Hạt nhân/Bộ lọc** | Cốt lõi của phép toán tích chập; các giá trị của nó được khởi tạo và cập nhật trong quá trình huấn luyện. Các hạt nhân có giá trị phần tử khác nhau có thể tạo ra các hiệu ứng khác nhau. | **Kích thước Hạt nhân (Kernel Size):** Chiều cao và chiều rộng của bộ lọc (ví dụ: 3×3). |
| **Bước dịch (Stride)** | Số lượng pixel mà hạt nhân dịch chuyển trên khối lượng đầu vào khi chuyển sang vị trí tiếp theo. Bước dịch lớn hơn làm giảm kích thước đầu ra. | Mặc định thường là **1**. |
| **Đệm (Padding)** | Kiểm soát kích thước đầu ra của lớp tích chập. Đệm **“Valid”** (hợp lệ) nghĩa là không có đệm nào được thêm vào, dẫn đến kích thước đầu ra nhỏ hơn. Đệm **“Same”** (giữ nguyên) thêm các đường viền để giữ kích thước đầu ra bằng kích thước đầu vào (mặc dù không được đề cập rõ ràng trong đoạn trích). | **Valid** (Hợp lệ) hoặc **Same** (Giữ nguyên). |
| **Lớp Gộp (Pooling)** | Được sử dụng để **giảm mẫu** (giảm) kích thước không gian (chiều cao và chiều rộng) của các bản đồ đặc trưng, giúp giảm số lượng tham số và độ phức tạp tính toán. **Max pooling** (Gộp tối đa) là một phép toán phổ biến. | Các lớp gộp hoạt động **độc lập trong mỗi kênh** của khối lượng đầu vào. |
| **Duỗi thẳng (Flatten) Þ Dense** | Bước cuối cùng của Bộ trích xuất Đặc trưng là **lớp Duỗi thẳng**, lớp này định hình lại các bản đồ đặc trưng 3D thành một **vector đặc trưng một chiều**. Vector này sau đó được đưa vào **các lớp Liên kết đầy đủ (Dense)** (Bộ phân loại) để dự đoán cuối cùng. | Các lớp liên kết đầy đủ kết hợp các đặc trưng đã trích xuất để đưa ra quyết định phân loại. |

Bảng 1 Các khối xây dựng CNN

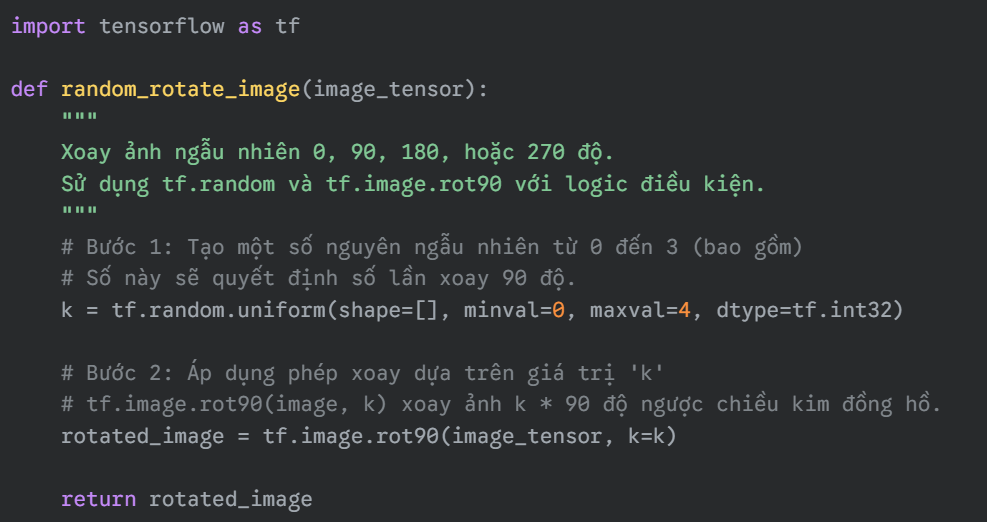
### 2.4.3. Các Phép Toán Tăng cường Dữ liệu (Data Augmentation Ops)

Tăng cường dữ liệu là một kỹ thuật để chống lại **quá khớp (overfitting)** bằng cách tăng **số lượng và sự đa dạng** của dữ liệu huấn luyện thông qua các phép biến đổi khác nhau.

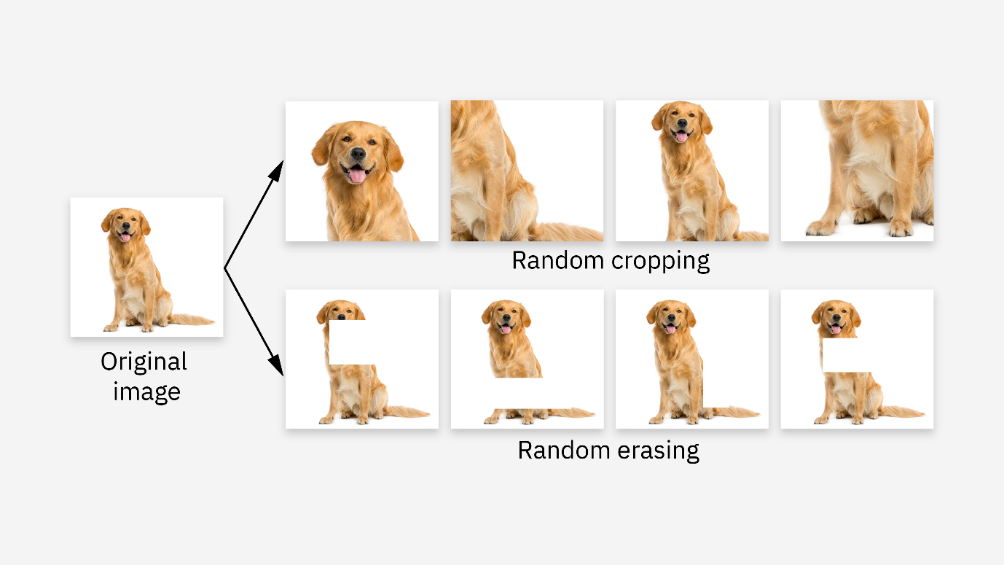


Hình 9. Ví dụ về các phương pháp tăng cường dữ liệu

Các kỹ thuật biến đổi hình ảnh phổ biến được liệt kê là:

* **Lật ảnh (Image Flipping)** (ví dụ: lật ngẫu nhiên trái-phải)
* **Xoay ảnh (Image Rotation)** 

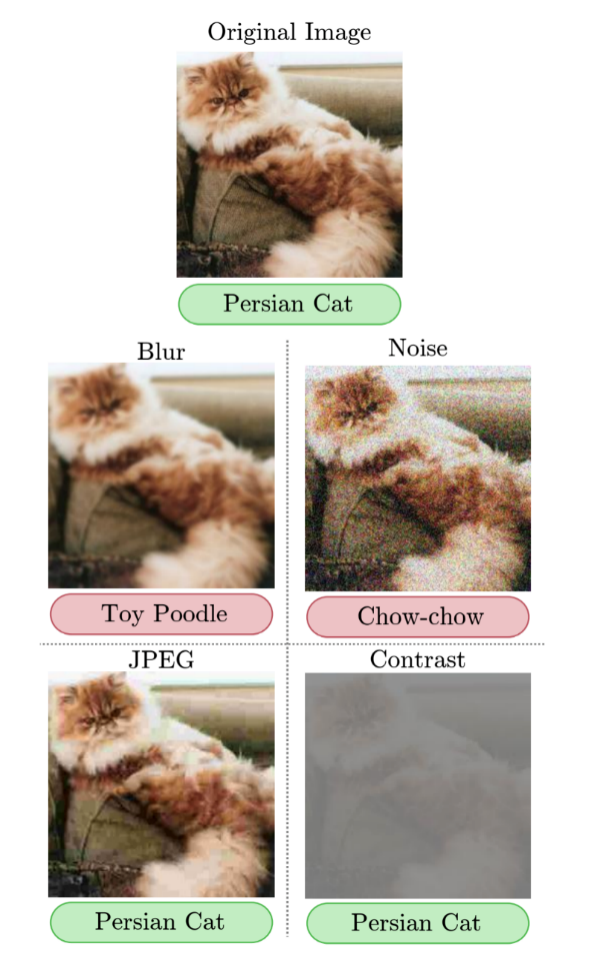
Hình 10. Một đoạn code ví dụ cho việc lật ảnh

* **Dịch chuyển ảnh (Image Shifting)** 

Hình 11. Ví dụ về Image Shifting

* **Thay đổi tỷ lệ ảnh (Image Scaling)**
* **Chuyển đổi màu sắc (Color Conversion)** 

Hình 12. Ví dụ cho Color Conversion

* **Làm mờ ảnh (Blur Image)** hoặc t**hêm nhiễu (Add Noise)**

Hình 13 Ví dụ cho làm mờ ảnh (Blur Image) hoặc thêm nhiễu (add noise)

Bạn có thể triển khai chúng bằng cách sử dụng:

* **tf.image:** Các hàm TensorFlow trực tiếp (ví dụ: tf.image.random\_flip\_left\_right, tf.image.random\_crop).
* **tf.keras.preprocessing.image.ImageDataGenerator:** (Công cụ Keras cấp cao hơn cho augmentation).

### 2.4.4. Cấu trúc cơ bản cho CNN CIFAR-10

Cấu trúc cơ bản cho CNN được sử dụng trong thí nghiệm CIFAR-10 (Model-2 và Model-3) tuân theo khuôn mẫu tiêu chuẩn cho phân loại đa danh mục.

Kiến trúc này được chia thành ba phần chính: **Đầu vào**, **Bộ Trích xuất Đặc trưng** (Feature Extractor), và **Bộ Phân loại** (Classifier).

#### 1. Khối Đầu vào

| Lớp (Layer) | Chức năng | Kích thước Tensor (Đầu ra) |
| --- | --- | --- |
| **Input Layer** | Ảnh màu CIFAR-10 3 kênh (RGB). | 32×32×3 |

Bảng 2 Khối đầu vào

#### 2. Bộ Trích xuất Đặc trưng (Convolutional Base)

Phần này chịu trách nhiệm học các đặc trưng từ đơn giản (cạnh) đến phức tạp (kết cấu). Kiến trúc thường lặp lại một khối Conv + ReLU + MaxPool nhiều lần.

| Lớp (Layer) | Chức năng | Tham số (Ví dụ) | Kích thước Tensor (Đầu ra) |
| --- | --- | --- | --- |
| **Conv2D** | Trích xuất đặc trưng cấp thấp. | 64 filters, 3×3 kernel, Padding = ‘same’ | 32×32×64 |
| **ReLU** | Thêm tính phi tuyến tính. | - | 32×32×64 |
| **MaxPooling2D** | Giảm kích thước không gian 4 lần. | 2×2 pool size, Stride = 2 | 16×16×64 |
| **Conv2D** | Trích xuất đặc trưng phức tạp hơn. | 128 filters, 3×3 kernel, Padding = ‘same’ | 16×16×128 |
| **ReLU** | Thêm tính phi tuyến tính. | - | 16×16×128 |
| **MaxPooling2D** | Giảm kích thước không gian lần 2. | 2×2 pool size, Stride = 2 | 8×8×128 |

Bảng 3 Mô tả các lớp đặc trưng

#### 3. Bộ Phân loại (Classifier Head)

Phần này lấy các đặc trưng đã trích xuất, duỗi thẳng chúng thành một vector duy nhất, và sử dụng các lớp Dense để đưa ra dự đoán cuối cùng.

| Lớp (Layer) | Chức năng | Tham số (Ví dụ) | Kích thước Tensor (Đầu ra) |
| --- | --- | --- | --- |
| **Flatten** | Biến đổi khối 3D thành vector 1D. | - | 8×8×128 = 8192 |
| **Dense (Hidden)** | Lớp ẩn liên kết đầy đủ. | 512 units | 512 |
| **ReLU** | Hàm kích hoạt cho lớp Dense. | - | 512 |
| **Dense (Output)** | Lớp đầu ra cho số lượng danh mục. | 10 units (số lớp CIFAR-10) | 10 |
| **Softmax** | Chuyển đổi điểm số thành **Xác suất**. | - | 10 |

Bảng 4 Mô tả bộ phân loại

Sơ đồ này minh họa rõ ràng cách dữ liệu đi từ ảnh 32×32×3 ban đầu đến 10 xác suất đầu ra.

[CNN Visualized](https://youtu.be/pj9-rr1wDhM?si=rNjJlEy_NEOo5joL)

## 2.5 Training techniques — Backpropagation, Weight initialization, Batch Normalization

### 2.5.1.Mô tả backpropagation (chain rule) và công thức cập nhật.

* **Backpropagation (Lan truyền ngược)** là phương pháp cập nhật trọng số của mạng nơ-ron bằng cách kết hợp với **thuật toán gradient descent**. Quá trình này sử dụng **quy tắc chuỗi (chain rule)** để tính **đạo hàm của hàm mất mát theo từng trọng số** trong mạng.
* Cụ thể, sai số được lan truyền ngược từ **layer đầu ra → layer ẩn → layer đầu vào**, và các trọng số được điều chỉnh theo hướng **giảm sai số**.  
   Công thức cập nhật trọng số:

**Trong đó:**

* L là **hàm mất mát (loss function)**,
* η là **tốc độ học (learning rate)**,
* W là **trọng số (weights)** của mạng nơ-ron.

Dựa vào công thức trên, **lan truyền ngược (Backpropagation - BP)** được dùng để **tính gradient của hàm mất mát theo từng trọng số** của mạng, ứng với mỗi cặp **dữ liệu đầu vào và đầu ra**.

BP giúp **tính toán các đạo hàm riêng** để **cập nhật trọng số** của mạng, như minh họa trong **Hình 5.2**.









Hình 14 Sơ đồ tính toán lan truyền ngược (Backpropagation - BP) của mạng nơ-ron một lớp.

### 2.5.2. Phương pháp Normal, Glorot/Xavier và He Initialization :

- **Normal distribution initialization**: Là phương pháp khởi tạo đơn giản nhất với μ = 0 và σ = 1 (phân phối chuẩn tắc) hoặc μ=0 hoặc σ = 0,01  
 Khi áp dụng cho mạng dùng hàm kích hoạt sigmoid, nếu σ = 1 sẽ gây mất gradient (vanishing gradient), còn σ nhỏ (như 0.01) giúp ổn định hơn.  
 → Thích hợp cho các mạng nhỏ hoặc khi muốn kiểm thử thủ công ảnh hưởng của trọng số ban đầu.

- **Xavier (Glorot) initialization**: Gồm 2 loại , loại mặc định Glorot Normal (Xavier Normal) và Glorot Uniform (Xavier uniform) . Được thiết kế để giữ cho gradient có quy mô tương đương ở tất cả các lớp, khắc phục hạn chế của khởi tạo Normal.

Cách làm: Điều chỉnh độ rộng của phân phối phụ thuộc vào số lượng neuron đầu vào và đầu ra bằng triển khai Glorot .

Công thức **Glorot Uniform (Xavier uniform)** :

*Trong đó :*

* *: Ma trận (hoặc vector) trọng số của một layer trong mạng nơ ron.*
* *.*
* *: Số lượng input units (số neuron đầu vào) của layer.*

Công thức **Glorot Normal (Xavier Normal)** :

* (8)
* Trong đó :
* *: Ma trận (hoặc vector) trọng số của một layer trong mạng nơ ron.*
* *: Số lượng* ***input units*** *(số neuron đầu vào) của layer.*

Tuy nhiên, khi dùng với hàm ReLU, Glorot initialization vẫn có thể gây mất gradient.  
 → Thích hợp cho mạng dùng hàm kích hoạt sigmoid hoặc tanh.

- He initialization: Giải quyết vấn đề của Glorot khi sử dụng ReLU, bằng cách giữ phân phối đầu ra giữa các lớp cân bằng hơn, giúp tránh mất gradient.

**Cách làm:** Tăng phương sai của trọng số phụ thuộc vào số lượng neuron đầu vào để bù cho phần giá trị âm bị ReLU loại bỏ:

Trong đó :

* *: Ma trận (hoặc vector) trọng số của một layer trong mạng nơ ron.*
* *: Số lượng* ***input units*** *(số neuron đầu vào) của layer.*

→ Thích hợp cho mạng sâu (DNN) dùng ReLU hoặc các biến thể của ReLU như Leaky ReLU, ELU.

**Kết luận:**

* Phương pháp Glorot được khuyến nghị cho khởi tạo trọng số khi sử dụng mạng neural với hàm kích hoạt sigmoid hoặc tanh.
* Phương pháp He được khuyến nghị cho khởi tạo trọng số khi sử dụng mạng neural với hàm kích hoạt ReLU.

- Phương pháp Normalization được khuyến nghị sử dụng để chuẩn hóa đầu ra của các lớp trong mạng neural, giúp quá trình huấn luyện ổn định và tăng tốc độ hội tụ.

### 2.5.3. Kĩ thuật chuẩn hóa đầu vào Batch Normalization (mean/variance normalization, gamma & beta).

* Batch Normalization là kỹ thuật chuẩn hóa đầu vào của mỗi lớp theo **trung bình** **(mean)** và **phương sai** **(variance)** trong từng **mini-batch**, giúp giữ cho phân phối dữ liệu ổn định trong quá trình huấn luyện mạng nơ-ron.
* Ví dụ : Với dữ liệu mini-batch *M* ={x1,x2,…,xn} , phép biến đổi batch normalization gồm **4 bước**:
* Bước 1: Tính trung bình mini-batch:

Ảnh có chứa Phông chữ, biểu đồ, hàng, thiết kế

Nội dung do AI tạo ra có thể không chính xác.

* Bước 2: Tính phương sai mini-batch:

Ảnh có chứa Phông chữ, số, hàng, màu trắng

Nội dung do AI tạo ra có thể không chính xác.

* Bước 3: Chuẩn hóa giá trị:

Ảnh có chứa Phông chữ, hàng, văn bản, màu trắng

Nội dung do AI tạo ra có thể không chính xác.

* Bước 4: Scale và shift (nhân và dịch chuyển):



Trong đó :

* yi là đầu ra sau chuẩn hóa.
* (gamma) và β (beta) là tham số học được giúp mạng vẫn linh hoạt trong quá trình học sau khi chuẩn hóa.
* ε là hằng số nhỏ để tránh chia cho 0.
* là phương sai (variance) của mini-batch.
* là giá trị trung bình (mean) của các trong mini-batch
* là giá trị đầu vào (activation) của sample thứ trong mini-batch.
* là giá trị đã được chuẩn hóa của .

A diagram of a graph

AI-generated content may be incorrect.

Hình 15: So sánh không sử dụng chuẩn hóa dữ liệu và chuẩn hóa dữ liệu

Nhận xét :

* Trong Hình 15A, đầu ra của một lớp trong mạng được phân bố đều trong khoảng –10 đến 10. Sau khi đi qua hàm kích hoạt tanh, phân bố đầu ra trở nên không đều; phần lớn các giá trị đầu ra tập trung gần –1 và 1, từ đó gây ra hiện tượng vanishing gradient (tham khảo đạo hàm của hàm tanh trong
* Trong Hình 15B, đầu ra của lớp mạng trước tiên được chuẩn hóa về khoảng xấp xỉ –2 đến 2, sau đó mới được đưa qua hàm kích hoạt tanh, nhờ đó phân bố đầu ra trở nên đồng đều hơn so với trường hợp không sử dụng chuẩn hóa như trong Hình 15A

Ưu điểm của Batch Normalization:

* Tăng tốc quá trình huấn luyện mạng nơ-ron nhờ cho phép sử dụng learning rate lớn hơn
* Giảm hiện tượng overfitting trong quá trình huấn luyện mạng nơ-ron
* Giảm các vấn đề vanishing gradient và exploding gradient
* Loại bỏ nhu cầu sử dụng kỹ thuật dropout để tránh mất mát thông tin dữ liệu

Vị trí của Batch Normalization :

A diagram of a model

AI-generated content may be incorrect.

Hình 16 Cấu trúc của một mạng nơ-ron tích chập có sử dụng batch normalization.

Nhận xét : Batch normalization đã được áp dụng thành công trong các mạng nơ-ron nhằm cải thiện hiệu năng của nhiều ứng dụng thị giác máy tính, chẳng hạn như phân loại ảnh , nhận dạng ảnh , phát hiện đối tượng, dịch ảnh sang ảnh (image-to-image translation) , v.v. Các mạng nơ-ron có sử dụng batch normalization khác đôi chút so với các mạng nơ-ron thông thường, trong đó batch normalization được đặt giữa lớp convolution và hàm kích hoạt

## 2.6 Advanced TensorFlow (custom layer/loss/metric/callback) & Advanced TensorBoard (tf.summary, HParams)

### 2.6.1 Khi nào cần viết custom Layer (subclass tf.keras.layers.Layer)?

- Khi cần định nghĩa một loại lớp mạng không có sẵn trong TensorFlow bằng cách tạo lớp mạng tùy chỉnh (custom layer) bằng cách kế thừa lớp tf.keras.layers.Layer .  
- Custom layer thường được dùng khi muốn thực hiện các phép tính hoặc hành vi đặc biệt vượt ngoài các lớp tiêu chuẩn.

### 2.6.2 Khi nào cần viết custom loss ?

* TensorFlow cung cấp nhiều hàm mất mát (loss function) cho các bài toán như phân loại, nhận dạng, hồi quy, v.v. Tuy nhiên, nếu không có hàm mất mát nào phù hợp với bài toán cụ thể, ta cần tự định nghĩa hàm loss mới để đáp ứng yêu cầu của mô hình.

### 2.6.3 Khi nào cần custom metric ?

Hàm metric được dùng để đánh giá hiệu suất hoặc chất lượng của mô hình, và mỗi bài toán sẽ có tiêu chí đánh giá riêng. Nhưng nếu TensorFlow không có sẵn metric phù hợp, ta có thể tạo metric tùy chỉnh bằng cách kế thừa lớp tf.keras.metrics.Metric và định nghĩa cách tính riêng.

### 2.6.4 khi nào cần viết custom callback ?

* Callback là công cụ mạnh giúp thực hiện các hành động tại những giai đoạn khác nhau trong quá trình huấn luyện, như lưu mô hình, điều chỉnh learning rate hoặc dừng sớm.
* Nếu TensorFlow không có callback sẵn phù hợp với nhu cầu, ta có thể tạo callback tùy chỉnh bằng cách kế thừa lớp “tf.keras.callbacks.Callback”.

### 2.6.5 Dùng tf.summary để log và bật TensorBoard

Các loại tf.summary thường dùng:

* tf.summary.scalar() — log scalar (loss, lr, metrics).
* tf.summary.histogram() — log phân bố (weights, gradients).
* tf.summary.image() — log ảnh đầu vào / ảnh dự đoán / feature maps.
* tf.summary.text() — log thông tin dạng text hoặc config.

Quy trình cơ bản (viết vào báo cáo):

1. Tạo writer:

writer = tf.summary.create\_file\_writer('logs/run\_x').

1. Log trong training loop hoặc callback: dùng with writer.as\_default(): tf.summary.scalar('loss', loss\_value, step=step) hoặc gọi trong custom Callback.
2. Mở TensorBoard: chạy tensorboard --logdir logs/ và truy cập giao diện để xem scalar/histogram/image/HParams.
3. HParams: log config hyperparameters và metrics để so sánh nhiều runs trong HParams dashboard.

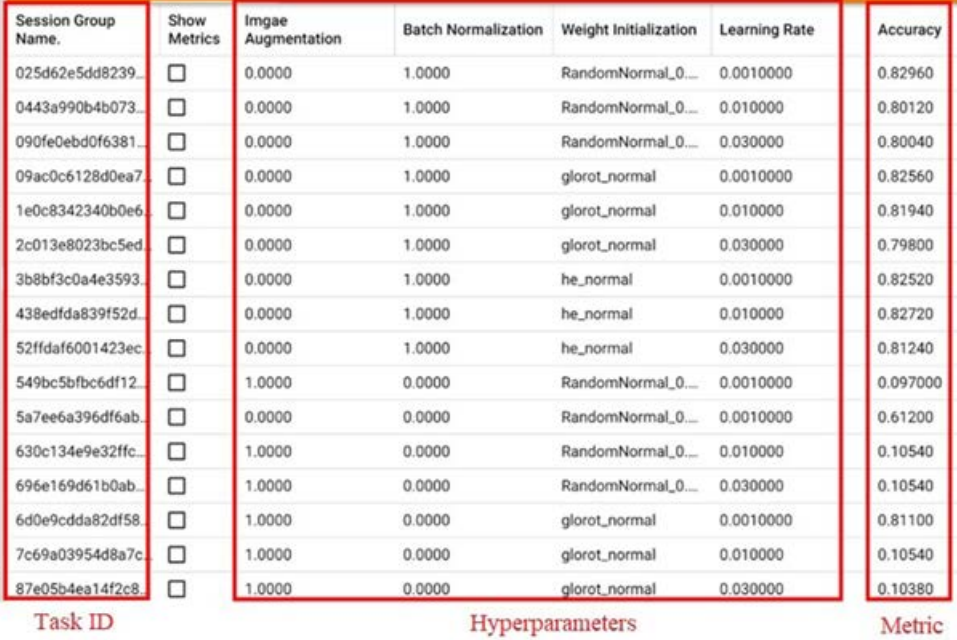
### 2.6.6 HParams dashboard & khuyến nghị thực nghiệm

* Mục đích: so sánh nhiều runs với các hyperparameter khác nhau (ví dụ: learning\_rate, initializer, batch\_size) trên cùng 1 dashboard, dễ phát hiện cặp hyperparam tốt nhất.
* Khuyến nghị thực nghiệm: log ít nhất 3 runs với HParams khác nhau (ví dụ lr = [1e-3, 5e-4, 1e-4] hoặc initializer = [normal, glorot, he]) để có dữ liệu so sánh. Chụp HParams dashboard screenshot để đưa vào báo cáo.

Ảnh có chứa hàng, biểu đồ

Nội dung do AI tạo ra có thể không chính xác.

Hình 17 Bảng điều khiển HParams ở chế độ hiển thị song song



Hình 18 Table View – Bảng so sánh kết quả từng run

Mô tả : Hình 15 và 16 trình bày bảng HParams dashboard trong TensorBoard, thể hiện các giá trị hyperparameter (sự tăng cường ảnh, chuẩn hóa batch, khởi tạo trọng số, tốc độ học) và độ chính xác tương ứng cho từng run.

# 3. Phương pháp thực hiện

Phần này mô tả chi tiết quy trình thực nghiệm được thực hiện trên các bộ dữ liệu khác nhau (Pokémon cho phân loại nhị phân và CIFAR-10 cho phân loại đa lớp và thí nghiệm về kỹ thuật huấn luyện mạng nơ-ron tích chập - CNN). Toàn bộ quá trình được xây dựng bằng nền tảng **TensorFlow 2** và API cấp cao **tf.keras**.

## 3.1. Tổng quan Quy trình Thực nghiệm

Quy trình thực hiện bao gồm các giai đoạn chuẩn hóa trong Deep Learning, đảm bảo tính nhất quán và khả năng tái hiện của các thí nghiệm.

1. **Chuẩn bị Dữ liệu (Data Preparation):** Đọc dữ liệu, thực hiện tiền xử lý, chuẩn hóa các đặc trưng số, mã hóa các đặc trưng phân loại, và xây dựng pipeline dữ liệu hiệu quả bằng **tf.data**

2. **Xây dựng Mô hình (Model Construction):** Thiết kế kiến trúc mạng nơ-ron (FCNN hoặc CNN) bằng API tf.keras.Sequential hoặc Functional API, đồng thời tích hợp các kỹ thuật chống overfitting như Dropout

3. **Huấn luyện Mô hình (Model Training):** Thiết lập hàm mất mát (loss), thuật toán tối ưu hóa (optimizer), và các chỉ số đánh giá (metrics). Sử dụng các **Callback** như ModelCheckpoint để lưu lại trọng số tốt nhất, và TensorBoard để ghi lại lịch sử huấn luyện

4. **Đánh giá và Trực quan hóa (Evaluation and Visualization):** Đánh giá hiệu suất cuối cùng trên tập kiểm thử (test set) và trực quan hóa các kết quả quan trọng như Loss/Accuracy curve, và Confusion Matrix

## 3.2. Bộ Dữ liệu (Datasets)

Các thí nghiệm được thực hiện trên hai bộ dữ liệu chính:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Thí nghiệm | Mục tiêu Thí nghiệm | |  |  | | --- | --- | |  | Đặc điểm | | Nguồn |
| **Pokémon-Weedle’s Cave** | Phân loại nhị phân (Legendary/Non-Legendary) | Dữ liệu dạng bảng (tabular data), mỗi mẫu có đặc trưng như HP, Attack, Defense, Speed | Kaggle |
| **CIFAR-10** | |  |  | | --- | --- | |  | Phân loại đa lớp, so sánh Khởi tạo Trọng số và Batch Normalization | | ảnh RGB kích thước , chia thành lớp (ví dụ: máy bay, ô tô, mèo, chó) | TensorFlow Datasets (Krizhevsky, 2009) |

Bảng 5 Mô tả bộ dữ liệu

## 3.3. Tiền xử lý dữ liệu và pipeline

### 3.3.1. Tiền xử lý dữ liệu (Preprocessing)

1. Chuẩn hóa Đặc trưng số (Feature Normalization)

- Đối với dữ liệu Pokémon, các thuộc tính số (như HP, Attack) được chuẩn hóa bằng **Standard Score** (z-score) để đưa về cùng thang đo, giúp tăng tốc độ hội tụ và ổn định quá trình huấn luyện. Công thức chuẩn hóa là:

- Đối với dữ liệu ảnh CIFAR-10, giá trị pixel (từ 0 đến 255) được chuẩn hóa bằng cách chia cho , đưa chúng về phạm vi

2. Mã hóa đặc trưng phân loại (Categorial Feature Encoding):

- Trong thí nghiệm Pokemon, để xử lý các thuộc tính phân loại (Type 1, Type 2) báo cáo so sánh hai phương pháp:

- Integer Encoding: Gán mỗi danh mục một số nguyên (ví dụ “Fire” = 5)

- One-hot Encoding**:** Biểu diễn mỗi danh mục bằng một vector nhị phân độc lập, giải quyết vấn đề thứ tự giả định. Kết quả thực nghiệm cho thấy One-hot Encoding mang lại độ chính xác cao hơn

3. Mã hóa nhãn (Label Encoding)

- Phân loại nhị phân (Pokémon): Nhãn Legendary được chuyển thành giá trị nhị phân (1 hoặc 0)

- Phân loại đa lớp (CIFAR-10): Nhãn được chuyển sang dạng One-hot Encoding

### 3.3.2. Xây dựng pipeline bằng tf.data

Để tối ưu hóa hiệu năng huấn luyện, tất cả dữ liệu được đóng gói vào đối tượng tf.data.Dataset. Pipeline này cho phép:

**- Xáo trộn (Shuffle):** Xáo trộn dữ liệu huấn luyện để giảm phương sai và tránh thứ tự mẫu ảnh hưởng đến gradient

- **Chia lô (Batch):** Chia dữ liệu thành các lô nhỏ (mini-batch), thường là , để tối ưu hóa quá trình tính toán gradient

- **Tải trước (Prefetch):** Sử dụng tf.data.AUTOTUNE để tải trước các lô dữ liệu vào bộ nhớ GPU/TPU, tránh tình trạng **"đói dữ liệu" (data starvation)** và tận dụng tối đa tài nguyên tính toán

### 3.3.3. Tăng cường Dữ liệu (Data Augmentation)

Kỹ thuật Tăng cường Dữ liệu (Data Augmentation) được áp dụng trong các mô hình CNN (như Model-3 trong Chương 4) để chống overfitting và tăng khả năng tổng quát hóa của mô hình. Các phép biến đổi áp dụng cho dữ liệu ảnh bao gồm

- Lật ảnh ngẫu nhiên trái-phải (flip)

- Thay đổi màu sắc (hue, saturation, brightness, contrast)

- Xoay ảnh ngẫu nhiên (rotate)

- Phóng to/thu nhỏ ảnh (zoom)

## 3.4. Kiến trúc Mô hình và Thiết lập Huấn luyện

### 3.4.1. Thí nghiệm Phân loại Nhị phân (Pokémon)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tham số | |  |  | | --- | --- | |  | Giá trị | | Mục đích |
| **Kiến trúc** | FCNN (4 lớp Dense, 1 lớp Output) | Mạng Liên kết Đầy đủ để xử lý dữ liệu đặc trưng dạng vector |
| **Kích thước Input** | 20 (Integer Encoding) hoặc 54 (One-hot Encoding) | Tùy thuộc vào phương pháp mã hóa thuộc tính phân loại (Type 1, Type 2) |
| **Tầng Ẩn** | **64 -> 64 -> 64 -> 16 nơ-ron** | Tạo độ sâu cho mạng để học các mối quan hệ phức tạp. |
| **Hàm Kích hoạt** | |  |  | | --- | --- | |  | **ReLU** (lớp ẩn), **Sigmoid** (lớp đầu ra) | | ReLU khắc phục vanishing gradient; Sigmoid cho xác suất nhị phân (0,1) |
| **Kỹ thuật Overfitting** | Dropout (0,3) | Ngẫu nhiên bỏ qua 30% nơ-ron để ngăn chặn overfitting |
| Loss Function | BinaryCrossEntropy | Hàm mất mát tiêu chuẩn cho phân loại nhị phân |
| Optimizer | Adam | Thuật toán tối ưu hóa tự điều chỉnh tốc độ học |
| Metrics | Binary Accuracy | Đo lường tỉ lệ dự đoán đúng |
| Callbacks | TensorBoard, Model Check Point | Ghi log và lưu trọng số tốt nhất (dựa trên val\_binary\_accuracy) |

Bảng 6 Thí nghiệm Phân loại nhị phân

### 3.4.2. Thí nghiệm So sánh Kỹ thuật Huấn luyện (CIFAR-10)

Thí nghiệm này tập trung vào việc đánh giá ảnh hưởng của Weight Initialization và Batch Normalization trên kiến trúc CNN.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Kiến trúc Cơ bản | |  |  | | --- | --- | |  | Kỹ thuật Tăng cường | | |  |  | | --- | --- | |  | Initialization | | |  |  | | --- | --- | |  | Chuẩn hóa | |
| **Model-1** (Chương 5) | CNN | |  |  | | --- | --- | |  | Data Augmentation | | Normal () | Không |
| **Model-2** (Chương 5) | CNN | |  |  | | --- | --- | |  | Data Augmentation | | Glorot Normal (Xavier) | Không |
| **Model-3** (Chương 5) | CNN | |  |  | | --- | --- | |  | Data Augmentation | | **He Normal** | Không |
| **Model-4** (Chương 5) | CNN | |  |  | | --- | --- | |  | Data Augmentation | | Glorot/He (Mặc định) | **Batch Normalization (BN)** |

Bảng 7 Thí nghiệm so sánh kỹ thuật huấn luyện

**Thiết lập CNN Cơ bản (Chương 5):** Kiến trúc CNN bao gồm lớp tích chập (Conv2D), theo sau là hàm kích hoạt **ReLU**, lớp gộp cực đại (MaxPool2D), lớp làm phẳng (Flatten), lớp kết nối đầy đủ (Dense), lớp Dropout (0.5), và lớp đầu ra (10 nơ-ron) với hàm **Softmax**.

**Thiết lập Huấn luyện Chung:**

* **Loss Function:** CategoricalCrossentropy•
* **Optimizer:** Adam
* **Metrics:** CategoricalAccuracy
* **Callbacks:** TensorBoard (với histogram\_freq=1 để theo dõi phân bố trọng số) và ModelCheckpoint

# 4. Kết quả thực nghiệm

Phần này trình bày kết quả từ ba nhóm thí nghiệm chính đã được mô tả trong báo cáo (Phân loại nhị phân, Phân loại đa lớp, và Thí nghiệm Kỹ thuật Huấn luyện)

## 4.1. Môi trường và Metrics đánh giá

### 4.1.1. Môi trường thực nghiệm

Nền tảng chính được sử dụng là TensorFlow 2, một nền tảng mã nguồn mở mạnh mẽ để xây dựng, huấn luyện và triển khai các mô hình học sâu. TensorFlow 2 được chọn vì khả năng dễ mở rộng và hỗ trợ đầy đủ các công cụ từ huấn luyện, đánh giá đến triển khai mô hình.

Các công cụ được sử dụng bao gồm:

- tf.keras: Cung cấp API cấp cao giúp xây dựng mô hình mạng nơ-ron một cách nhanh chóng và dễ hiểu.

- tf.data: Cung cấp API mạnh mẽ để xử lý dữ liệu đầu vào, xây dựng pipeline tối ưu cho huấn luyện, giúp tận dụng khả năng tải dữ liệu song song và tránh hiện tượng “đói dữ liệu” cho GPU/CPU.

### 4.1.2. Metrics đánh giá:

Các metrics đánh giá phụ thuộc vào loại bài toán:

1. Phân loại nhị phân (Binary Classification)

- Hàm mất mát (Loss Function): binary\_crossentropy

- Chỉ số đánh giá (Metrics): accuracy (tỷ lệ dự đoán đúng trên tổng số mẫu)

- Trực quan hóa: Ma trận nhầm lẫn (Confusion Matrix) để phân tích chi tiết True Positives, False Positives, True Negatives, False Negatives.

2. Phân loại đa lớp (Multi-class Classification):

- Hàm mất mát (Loss Function): CategorialCrossentropy

- Chỉ số đánh giá (Metrics): CategorialAccuracy

## 4.2. Kết quả thí nghiệm 1: Phân loại nhị phân (Dữ liệu Pokemon)

Mục tiêu của thí nghiệm này là phân biệt Pokémon Legendary và Non-Legendary. Kết quả được đánh giá thông qua so sánh giữa hai mô hình sử dụng cách mã hóa đặc trưng khác nhau (Integer Encoding và One-hot Encoding)

### 4.2.1. So sánh Integer Encoding (Model-1) và One-hot Encoding (Model-2):

Sử dụng One-hot Encoding (Model-2) để biểu diễn các thuộc tính của Pokemon (như Type 1, Type 2), cho thấy hiệu quả vượt trội so với Integer Encoding (Model-1)

- **Hiệu suất:** Độ chính xác trên dữ liệu huấn luyện và kiểm định của Model-2 **tốt hơn** Model-1

- **Lý do:** Khi sử dụng Integer Encoding, mô hình có thể giả định một mối quan hệ thứ tự tự nhiên giữa các danh mục (ví dụ: Type 1: Grass (1) và Fire (2) gần nhau hơn Type: Water (18)), điều này không hợp lý khi không có thứ tự tự nhiên giữa các thuộc tính Pokémon. One-hot Encoding giải quyết vấn đề này bằng cách làm cho mỗi lớp trở nên hoàn toàn độc lập.

## 4.3. Kết quả thí nghiệm 2: Phân loại đa lớp và CNN cơ bản (CIFAR-10)

Thí nghiệm này so sánh hiệu năng giữa Mạng Neural Liên kết đầy đủ (FCNN - Model-1) và Mạng Neural Tích chập (CNN - Model-2), đồng thời đánh giá tác động của kỹ thuật Tăng cường Dữ liệu (Data Augmentation - Model-3) trên tập A screenshot of a graph

AI-generated content may be incorrect.

Hình 19: Kết quả thí nghiệm 2: Phân loại đa lớp và CNN cơ bản (CIFAR-10)

### 4.3.1. Chi tiết kiến trúc

Thí nghiệm so sánh ba mô hình chính: Model-1 (FCNN), Model-2 (CNN không áp dụng tăng cường dữ liệu), và Model-3 (CNN áp dụng tăng cường dữ liệu)

|  |  |  |
| --- | --- | --- |
| Tên mô hình | Kiến trúc (Layers chính) | Mô tả |
| Model-1 (FCNN) | 6 lớp kết nối đầy đủ (Dense), 1 lớp Dropout, 1 lớp đầu ra (10 neurons) | Là một FCNN. |
| Model-2 (CNN) | 5 lớp tích chập (Conv2D), 1 lớp Max Pooling, 1 lớp kết nối đầy đủ (Dense), 1 lớp Dropout, 1 lớp đầu ra (10 neurons) | Là một CNN cơ bản |
| Model-3 (CNN + Data Augmentation) | 5 lớp tích chập (Conv2D), 1 lớp Max Pooling, 1 lớp kết nối đầy đủ (Dense), 1 lớp Dropout, 1 lớp đầu ra (10 neurons) | Có cùng kiến trúc với Model-2, nhưng áp dụng kỹ thuật tăng cường dữ liệu trong quá trình huấn luyện |

Bảng 8 So sánh 3 mô hình chính

Chi tiết thêm về Model-2 /Model-3 (CNN):

Model CNN được xây dựng với đầu vào có kích thước 32 x 32 x 3

* Các lớp tích chập (Conv2D) sử dụng hàm kích hoạt ReLU.
* Sau lớp layers.MaxPool2D() đầu tiên.
* Sau khi làm phẳng (Flatten), có một lớp layers.Dense(64, activation='relu')
* Một lớp layers.Dropout(0.5) được sử dụng
* Lớp đầu ra là layers.Dense(10, activation='softmax')

### 4.3.2. Phân tích kết quả

A screenshot of a graph

AI-generated content may be incorrect.

Hình 20 Kết quả của accuracy và loss trên tập train của 3 loại model (FCNN – CNN – CNN với data augmentation) trong bài toán phân loại đa lớp CIFAR-10

Trong đó :

* Model-1 (đường màu hồng) là mô hình FCNN.
* Model-2 (đường màu cam) là mô hình CNN.
* Model-3 (đường đỏ đậm) là mô hình CNN có sử dụng kỹ thuật tăng cường dữ liệu (image augmentation).

Trong đó :

* Model-1 (đường màu xanh lá) là mô hình FCNN.
* Model-2 (đường màu xanh dương) là mô hình CNN.
* Model-3 (đường xanh ngọc nhạt) là mô hình CNN có sử dụng kỹ thuật tăng cường dữ liệu (image augmentation).

Ở phần training set (Hình 18):

* Kết quả : Hình 18 chỉ ra rằng Model-2 đạt được kết quả tốt nhất trên tập training , theo sau lần lượt là Model-3 và Model-1. Các đường biểu diễn Model không xuất hiện vấn đề .
* Nhận xét : CNN vượt trội hơn FCNN: Model-2 (CNN) đạt độ chính xác ~96.35% và Model-3 (CNN + image agumentation) đạt độ chính xác ~ 73,28%, vượt trội đáng kể so với Model-1 (FCNN) chỉ đạt 58,74%. Loss của Model-2 (~11,88%) và Model-3 (~8,06%) cũng thấp hơn so với Model-1 (~12,86%).

Ở phần validation test (Hình 19):

A graph of loss and loss

AI-generated content may be incorrect.

Hình 21 Kết quả của accuracy và loss trên tập validation test của 3 loại model (FCNN – CNN – CNN với data augmentation) trong bài toán phân loại đa lớp CIFAR-10

* Dựa trên biểu đồ ta rút ra được :

|  |  |  |
| --- | --- | --- |
| Tên mô hình | Độ chính xác trên tập Validation (Accuracy) | Mô tả |
| Model-1 (FCNN) | 39.04% | Hiệu suất kém nhất |
| Model-2 (CNN) | 68.98% | Gặp vấn đề Overfitting |
| Model-3 (CNN + DA) | 78.86% | Ngăn chặn overfitting và cải thiện hiệu suất |

Bảng 9 So sánh các model trên tập validation (kết quả từ Lab 4)

* Kết quả : Hình 19 chỉ ra rằng Model-3 đạt được kết quả tốt nhất (Khác hẳn với training set - nơi mà Model-2 học tốt nhất) , theo sau lần lượt là Model-2 và Model-1. Model-2 còn bị overfitting ở biểu đồ loss và khả năng cải thiện độ chính xác của Model-2 bị chửng lại sau epoch 10 dù số epoch tăng lên.
* Nhận xét : Mặc dù ở training set , Model-2 đạt acc và loss tốt nhất nhưng khi qua validation set , Model-2 lộ ra vấn độ overfitting (loss giảm dần bất ngờ tăng dần qua các epoch) và accurancy không cải thiện qua các epoch. Model-3 với kết hợp của image agumentation lại tỏ ra hiệu quả hơn , giúp Model-3 đạt kết quả tốt nhất.

=> Kết luận :

* Image agumentation là một phương pháp tăng cường dữ liệu rất tốt để tăng độ chính xác , giảm loss , giảm nguy cơ overfitting, cải thiện tốc độ học của dữ liệu trong bài toán thị giác máy tính nhờ tăng số lượng và sự đa dạng của dữ liệu huấn luyện, cải thiện khả năng tổng quát hóa của mô hình.
* CNN là phương án tốt hơn FCNN cho bài toán thị giác máy tính nhờ khả năng trích xuất đặc trưng không gian (spatial feature extraction) thông qua lớp tích chập (Conv2D) và lớp gộp (Pooling).

## 4.4. Kết quả thí nghiệm 3: Khởi tạo trọng số (Weight Initialization) : Normal, Glorot/Xavier, He

Mục tiêu : Kiểm chứng độ hiệu quả của các phương pháp khởi tạo trọng số

Kiến trúc của các mô hình phân loại ảnh CIFAR-10 :

|  |  |  |
| --- | --- | --- |
| **Tên** | **Kiến trúc** | **Mô tả** |
| Mô hình 1 | * Lớp đầu vào có kích thước (32, 32, 3) * Năm lớp tích chập theo sau bởi hàm kích hoạt ReLU * Một lớp gộp cực đại * Một lớp làm phẳng để chuyển đầu vào thành một tensor một chiều * Một lớp kết nối đầy đủ * Một lớp dropout với tỉ lệ loại bỏ 50% * Lớp kết nối đầy đủ đầu ra có 10 nơ-ron, theo sau bởi hàm softmax | Sử dụng phương pháp khởi tạo trọng số theo phân phối chuẩn với μ=0 và σ=0.01 khi huấn luyện mô hình. |
| Mô hình 2 | * Lớp đầu vào có kích thước (32, 32, 3) * Năm lớp tích chập, theo sau bởi hàm kích hoạt ReLU * Một lớp gộp cực đại * Một lớp làm phẳng để chuyển đầu vào thành một tensor một chiều * Một lớp kết nối đầy đủ * Một lớp dropout với tỉ lệ loại bỏ 50% * Lớp kết nối đầy đủ đầu ra có 10 nơ-ron, theo sau bởi hàm kích hoạt softmax | Sử dụng phương pháp Glorot để khởi tạo trọng số khi huấn luyện mô hình. |
| Mô hình 3 | * Lớp đầu vào có kích thước (32, 32, 3) * Năm lớp tích chập, theo sau bởi hàm kích hoạt ReLU * Một lớp gộp cực đại (max pooling layer) * Một lớp làm phẳng để chuyển đầu vào thành một tensor một chiều * Một lớp kết nối đầy đủ, theo sau bởi hàm kích hoạt ReLU * Một lớp dropout với tỉ lệ loại bỏ 50% * Lớp kết nối đầy đủ đầu ra có 10 nơ-ron, theo sau bởi hàm kích hoạt softmax | Sử dụng phương pháp He để khởi tạo trọng số khi huấn luyện mô hình. |

Bảng 11 Kiến trúc của các mô hình phân loại ảnh CIFAR-10

Nhận xét : Các mô hình có cùng kiến trúc nhưng khác nhau ở phương pháp khởi tạo trọng số.

(CIFAR-10)

|  |  |  |  |
| --- | --- | --- | --- |
| Tên mô hình | Weight Initialization | Batch Normalization (BN) | Accuracy (Test) |
| Model-1 (Lab 5) | Normal | x | 10.00% |
| Model-2 (Lab 5) | |  |  | | --- | --- | |  | Glorot Normal (Xavier Normal) | | x | 79.33% |
| Model-3 (Lab5) | He Normal | x | 81.18% |
| Model-4 (Lab5) | Glorot Uniform (Xavier uniform) | v | 85.93% |

Bảng 10 So sánh các Phương pháp khởi tạo trọng số và Batch Normalization (CIFAR-10)

### 4.4.1. Phân tích kết quả khởi tạo trọng số

Kết quả :

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, màu trắng

Nội dung do AI tạo ra có thể không chính xác.

Bảng 12 Kết quả đo độ chính xác và sự mất mát của các model

Trong đó :

* Nhãn 0: đại diện cho Model-1 sử dụng phân phối chuẩn (Normal distribution) với μ=0 và σ=0.01.
* Nhãn 1: đại diện cho Model-2 sử dụng Glorot weight initialization.
* Nhãn 2: đại diện cho Model-3 sử dụng hàm kích hoạt ReLU và He weight initialization.

Nhận xét:

1. **Normal Initialization (Model-1):** Độ chính xác đạt thấp nhất (10.00%). Khởi tạo với phương sai nhỏ () khiến các trọng số ban đầu rất gần 0, dẫn đến **risk of vanishing gradients** và tốc độ hội tụ chậm. Trực quan hóa bằng **TensorBoard DISTRIBUTIONS/HISTOGRAMS** cho thấy sự thay đổi trọng số ở các lớp đầu tiên rất ít, gần như không được cập nhật.

**2. Glorot Initialization (Model-2):** Đạt độ chính xác 79.33%, cải thiện đáng kể so với Normal. Glorot được thiết kế để giữ quy mô gradient ổn định hơn nhưng vẫn chưa tối ưu khi dùng với hàm kích hoạt ReLU

3. **He Initialization (Model-3):** Đạt độ chính xác cao nhất trong nhóm (81.18%). Phương pháp He được thiết kế tối ưu cho mạng sử dụng **ReLU**. Kết quả trực quan hóa trọng số bằng **TensorBoard** cho thấy phân bố trọng số của Model-3 **rộng hơn** và thay đổi hiệu quả hơn Model-2, giúp mạng học được các đặc trưng đa dạng hơn và duy trì magnitude gradient đủ lớn ở tầng sâu.

(a) Sử dụng công cụ DISTRIBUTIONS A screenshot of a graph

AI-generated content may be incorrect.

Hình 22: DISTRIBUTIONS cho thấy sự thay đổi của trọng số của Model-1 khi sử dụng phân phối chuẩn (μ = 0 và σ = 0.01).

Mô tả ảnh :Trục x là trục thời gian , trục y là trục phân bổ trọng số. Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai. Góc dưới bên trái và góc dưới bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ ba.

Nhận xét ảnh :

* + - * Phân bố trọng số gần như không thay đổi đáng kể so với ban đầu
      * Vẫn co cụm quanh 0, biên độ rất hẹp
      * Cho thấy mô hình học yếu, cập nhật trọng số rất nhỏ
      * Chưa khai thác được năng lực của các layer convolution

-> Trọng số *trông chẳng được cập nhật bao nhiêu cả* → dấu hiệu underfitting , tiêu gradient.

(b) Sử dụng công cụ HISTOGRAMS

A screenshot of a computer

AI-generated content may be incorrect.

Hình 23 HISTOGRAMS minh họa sự thay đổi của trọng số của Model-1 khi sử dụng phân phối chuẩn (μ = 0 và σ = 0.01).

Mô tả ảnh : Trục x là trục thời gian , trục y là trục phân bổ trọng số. Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai. Góc dưới bên trái và góc dưới bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ ba.

Nhận xét ảnh :

* + - * Kernel (trọng số): phân bố vẫn gần dạng chuông quanh 0, chỉ giãn ra nhẹ theo thời gian → có học nhưng mức cập nhật yếu.
      * Bias: phân bố lệch và nhiễu, nhưng biên độ nhỏ → bias không đóng vai trò đáng kể trong học đặc trưng.
      * Layer 2 và 3 khá giống nhau → không có layer nào học vượt trội hay bị nổ/tiêu gradient.
      * Với Normal(μ=0, σ=0.01): training ổn định nhưng bảo thủ, learning signal yếu.

Kết luận : Dấu hiệu underfitting , learning rate thấp. CNN + ReLU + khởi tạo trọng số He sẽ phù hợp hơn.

Phân bố trọng số của Model-2 với khởi tạo Glorot :  
(a) Sử dụng công cụ DISTRIBUTIONS

A screenshot of a graph

AI-generated content may be incorrect.

Hình 24 Sử dụng công cụ DISTRIBUTIONS để quan sát sự thay đổi trọng số của Model-2 (Glorot)

Mô tả ảnh : Trục x là trục thời gian , trục y là trục phân bổ trọng số . Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ nhất. Góc dưới bên trái và góc dưới bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai**.**

Nhận xét : Như thể hiện trong hình, thời gian huấn luyện càng dài thì phân bố trọng số càng mở rộng, điều này cho thấy mô hình có thể học được các đặc trưng đa dạng hơn trong suốt quá trình huấn luyện so với khởi tạo Normal .

(b) Sử dụng công cụ HISTOGRAMS

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 25 Using HISTOGRAMS tool

Mô tả ảnh : Trục x là trục thời gian , trục y là trục phân bổ trọng số. HISTOGRAMS cho thấy sự thay đổi của trọng số của Model-2 khi sử dụng khởi tạo Glorot. Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ nhất. Góc dưới bên trái và góc dưới bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai**.**

Nhận xét ảnh :

* Phân bố trọng số (kernel) mở rộng rõ rệt theo epoch → mô hình học được đặc trưng đa dạng hơn so với init Normal(0, 0.01).
* Bias phân tán vừa phải, không bị lệch mạnh → learning ổn định.
* Layer conv2d\_5 học mạnh hơn conv2d\_6 (kernel giãn rộng hơn) → tầng sớm học nhiều pattern hơn.
* Không thấy dấu hiệu gradient nổ hoặc sụp (collapse).

Phân bố trọng số của Model-3 với khởi tạo He

(a) Sử dụng công cụ DISTRIBUTIONS

A graph of a running graph

AI-generated content may be incorrect.

Hình 26 Sử dụng công cụ DISTRIBUTIONS cho thấy sự thay đổi của trọng số của Model-3 khi sử dụng khởi tạo He

Mô tả ảnh : Trục x là trục thời gian , trục y là trục phân bổ trọng số. DISTRIBUTIONS cho thấy sự thay đổi của trọng số của Model-3 khi sử dụng khởi tạo He. Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ nhất. Góc dưới bên trái và góc dưới bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai.

Nhận xét ảnh : Hình cho thấy sự thay đổi của trọng số của lớp convolution thứ nhất và thứ hai, được quan sát bằng công cụ DISTRIBUTIONS, trong quá trình huấn luyện Model-3 qua 100 epoch khi sử dụng khởi tạo trọng số He.

(b) Sử dụng công cụ HISTOGRAMS

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 27 cho thấy sự thay đổi của trọng số của lớp convolution thứ nhất và thứ hai, được quan sát bằng cách sử dụng công cụ HISTOGRAMS.

Mô tả ảnh : Trục x là trục thời gian , trục y là trục phân bổ trọng số khi khởi tạo He. Góc trên bên trái và góc trên bên phải của hình lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ nhất. Góc dưới bên trái và góc dưới bên phải lần lượt là phân bố bias và phân bố kernel của lớp convolution thứ hai.

Nhận xét ảnh :

* Kernel giãn rộng rõ rệt theo epoch → tín hiệu học mạnh, đặc trưng đa dạng hơn.
* Bias tập trung quanh 0, độ phân tán vừa phải → ổn định, không lệch.
* Lớp 1 học mạnh hơn lớp 2 (kernel rộng hơn).
* Không có dấu hiệu nổ/tiêu gradient.

Kết luận : So với phân bố trọng số của Model-2 và Model-1, phân bố trọng số của Model-3 rộng hơn, điều này cho thấy Model-3 với khởi tạo He có thể học được các đặc trưng đa dạng hơn so với Model-2 với khởi tạo Glorot cũng như tránh được vấn đề tiêu gradient và underfitting mà Model-1 gặp phải.

Normal initialization phù hợp với các mạng nông hoặc đơn giản, nhưng dễ gây vanishing hoặc exploding gradient khi mạng sâu nếu không kết hợp với chuẩn hóa.  
Glorot (Xavier) initialization phù hợp với các mạng dùng sigmoid hoặc tanh, giúp cân bằng phương sai giữa đầu vào và đầu ra.  
He initialization phù hợp nhất với các mạng sử dụng ReLU và các biến thể của ReLU, đặc biệt là mạng CNN sâu.

## 4.5. Thí nghiệm 4 : Kiểm chứng batch normalization

Ta xây dựng model 4 rồi tiến hành thí nghiệm.

Thí nghiệm so sánh 4 mô hình CNN cùng kiến trúc chính nhưng khác nhau về kĩ thuật huấn luyện (technical for training) :

* Model-1 (khởi tạo normal distribution ).
* Model-2 (khởi tạo Glorot Normal (Xavier Normal)).
* Model-3 (khởi tạo He).
* Model-4 (khởi tạo Glorot Uniform + Batch Norm).

Kiến trúc Model sử dụng batch norm:

A screenshot of a computer

AI-generated content may be incorrect.

Hình 28 Kiến trúc của mạng nơ-ron có sử dụng Batch Normalization

* Nhận xét Hình : Kiến trúc mô hình đặt theo dạng Conv→BN→ReLU giúp :
* BN “đúng chỗ”: chuẩn hoá tín hiệu tuyến tính trước phi tuyến.
* Thường cho: hội tụ tốt hơn , accuracy cuối cao hơn hoặc ổn định hơn

Còn Conv→ReLU→BN thì rất ít được sử dụng

* ReLU làm phân bố lệch (cắt âm), BN sau đó chuẩn hoá phân bố đã bị biến dạng.
* Có thể:
  + không cải thiện rõ
  + đôi khi kém ổn định hơn

Kết quả :

A screenshot of a computer

AI-generated content may be incorrect.

Bảng 2 So sánh độ chính xác của bốn mô hình mạng nơ-ron

* Mô tả bảng :
* IA: Kỹ thuật tăng cường dữ liệu ảnh (*Image Augmentation*)
* RN: Khởi tạo trọng số theo phân phối chuẩn ngẫu nhiên (*Random Normal*)
* GU: Khởi tạo Glorot dạng phân phối đều (*Glorot Uniform*)
* GN: Khởi tạo Glorot dạng phân phối chuẩn (*Glorot Normal*)
* HN: Khởi tạo He dạng phân phối chuẩn (*He Normal*)
* BN: Chuẩn hóa theo minibatch (*Batch Normalization*)

Nhận xét : Theo hình ta thấy , Model-4, được xây dựng bằng cách thêm Batch Normalization (BN) + Glorot Uniform đã đạt độ chính xác cao nhất **85.93%** trong số 4 model .

**=>** Kết luận **:** Batch Normalization góp công lớn trong việc :

- Cải thiện hiệu suất: BN giúp tăng tốc độ huấn luyện (learning rate), giảm hiện tượng internal covariate shift, và cho phép sử dụng learning rate cao hơn.

- Tăng ổn định: Quá trình huấn luyện của Model-4 mượt mà hơn và ít biến động giữa các epoch . Điều này khẳng định BN cải thiện ổn định huấn luyện và hiệu năng tổng thể, đặc biệt khi kết hợp với khởi tạo phù hợp (He)

A screenshot of a computer

AI-generated content may be incorrect.

Hình 29 Mô tả thông tin của các train và validation của 4 model

Mô tả ảnh :

* Name : Tên của từng lần chạy (run), thể hiện phương pháp khởi tạo hoặc kỹ thuật sử dụng:
* Smoothed : Giá trị accuracy đã được làm mượt (smoothing) để giảm nhiễu khi quan sát xu hướng.
* Value : Giá trị accuracy thực tế tại bước cuối.
* Step : Số bước huấn luyện (ở đây là 99, tương ứng với epoch cuối).
* Relative : Thời gian huấn luyện tương đối cho mỗi run (khoảng 20–27 phút).

Nhận xét : Khi sử dụng batch normalization , run-4 là run duy nhất sử dụng batch normalization :

Ưu điểm:

* Run-4 đạt độ chính xác cao nhất (86.7%)so với các phương pháp còn lại, cho thấy batch normalization giúp mô hình hội tụ tốt và ổn định hơn.
* Giảm sự phụ thuộc vào weight initialization, trong khi Random Normal không hội tụ và Glorot/Variance Scaling cho kết quả thấp hơn.
* Cho phép mô hình học hiệu quả hơn với learning rate lớn, hạn chế hiện tượng vanishing/exploding gradient.

Nhược điểm:

* Thời gian huấn luyện lâu hơn so với các run khác () do phải tính toán thêm thống kê batch. Tức là thời gian huấn luyện bị tăng lên tận ≈30% .
* Tăng độ phức tạp của mô hình so với chỉ sử dụng các phương pháp khởi tạo trọng số.

Kết luận : Batch Normalization là 1 phương pháp rất tốt để cải thiện độ chính xác của mô hình cũng như ngăn chặn hiện tượng vanishing/exploding gradient nhưng đánh đổi lại là thời gian huấn luyện mô hình bị tăng lên rất nhiều.

## 4.6. Thí nghiệm 5 : Triển khai 2 mô hình sử dụng 2 API khác nhau (keras API cấp cao và API tùy chỉnh)

* Lí do thực hiện thí nghiệm **:** không phải tất cả các lớp mạng , hàm loss, hàm đánh giá , hàm callback cho mọi bài toán đều được tích hợp trong API Keras mức cao, do đó nhiều khi ta phải tự xây dựng API phục vụ cho bài toán của ta.
* Mục tiêu : Cho ta thấy , chúng ta có thể tự tạo và tinh chỉnh cũng như triển khai , xây dựng API phục vụ cho các bài toán khác nhau nhằm tối ưu kết quả .
* **Mô tả:** Chúng tôi trình bày hai kịch bản xây dựng mạng nơ-ron cho bài toán **phân loại đa lớp**. Cụ thể, chúng tôi sử dụng:  
  – **Model-1**: API Keras cấp cao của TensorFlow để xây dựng mạng  
  – **Model-2**: API tùy chỉnh (custom API) của TensorFlow để xây dựng mô hình mạng
* **Model-1** và **Model-2** được huấn luyện **độc lập** trên tập dữ liệu **CIFAR-10** nhằm **so sánh hiệu năng**. Hai mô hình **có cùng kiến trúc**.

**Kiến trúc mô hình :**

A screenshot of a computer

AI-generated content may be incorrect.

Bảng 3 : Kiến trúc các mô hình phân loại đa lớp

Kết quả :

A screenshot of a graph

AI-generated content may be incorrect.

Bảng 4 : Mô tả độ chính xác của API keras cấp cao và API tùy chỉnh

**Nhận xét :** Tuy xây dụng custom API của TensorFlow có thể tốn thời gian và công sức nhưng Kết quả thí nghiệm cho thấy độ chính xác (accuracy) của cả hai mô hình là gần tương đương nhau. Điều này xác nhận rằng việc sử dụng custom API của TensorFlow để thiết kế các mạng nơ-ron có thể đạt được kết quả tương tự so với việc sử dụng high-level Keras API, đồng thời mang lại tính linh hoạt cao hơn.

## 4.7. Thí nghiệm 6: Sử dụng tf.summary.image API

Mục tiêu của thí nghiệm : Nhằm giúp người đọc hiểu sâu hơn về các API của module tf.summary, thí nghiệm này sử dụng API tf.summary.image để ghi kết quả confusion matrix của một mô hình mạng được huấn luyện trên tập dữ liệu CIFAR-10 và hiển thị kết quả trên images dashboard của TensorBoard.

A screen shot of a graph

AI-generated content may be incorrect.

Hình 30 : Kết quả confusion matrix được trực quan hóa trên images dashboard.

Mô tả ảnh :

* Đường chéo là thể hiện độ chính xác của dự đoán , các đường dọc thể hiện độ dự đoán sai vào những nhãn cụ thể . Thanh dọc thể hiện số sample được dự đoán vào 1 class nào đó .
* Và theo như confusion matrix :
* Class “automobile” có độ chính xác cao nhất đồng nghĩa ít bị dự đoán sai nhất.
* “Dog” là class có độ chính xác thấp nhất , và có tới tận 20% số ảnh của “dog” bị nhận xét sai là cat . Đồng nghĩa với việc , mô hình đang bị bối rồi trong việc phân biệt “dog” và “cat”. Sâu xa rất có thể là do dữ liệu chưa đủ tốt và rất có thể bị gán nhãn sai .

Nhận xét : tf.summary.image cho phép trực quan hóa hình ảnh trên Images dashboard trong quá trình huấn luyện hoặc đánh giá mô hình. Đặc biệt , API này có thể trực quan hóa confusion matrix giúp người dùng :

* + - * Giúp phát hiện lớp dự đoán kém hoặc bị thiên lệch
      * Là cơ sở để tính precision, recall, F1-score
      * Hỗ trợ cải thiện mô hình và trực quan hóa kết quả (ví dụ trên TensorBoard)

## 4.8. Thí nghiệm 7: Điều chỉnh siêu tham số với TensorBoard HParams

Mục tiêu : giới thiệu một kỹ thuật tinh chỉnh các siêu tham số sử dụng HParams dashboard do TensorFlow cung cấp. Công cụ này giúp xác định các siêu tham số phù hợp nhất cho một bài toán cụ thể.

Mô tả : Kỹ thuật này được trình bày thông qua một ví dụ chương trình, trong đó 36 mô hình mạng khác nhau được xây dựng với các tổ hợp hyperparameters khác nhau, sau đó được huấn luyện trên tập dữ liệu CIFAR-10 để phân tích và so sánh kết quả thu được.

Các hyperparameters được sử dụng để xây dựng mô hình mạng bao gồm:

A screenshot of a black screen

AI-generated content may be incorrect.

Hình 31 : Thông tin chi tiết các siêu tham số có thể được sử dụng trong các models

Dựa trên hình , ta có thể tính được tổng số model :

|  |
| --- |
| Tổng số mô hình = tổng số mọi tổ hợp của các siêu tham số rời rạc |

Model = IA \*

= 2 \* 2 \* 3 \* 3

= 36

Trong đó :

* Model : Tổng số model có thể có
* IA : Tổng số lựa chọn tăng cường ảnh (image augmentation)
* BN : Tổng số lựa chọn batch normalization
* Init : Tổng số lựa chọn khởi tạo trọng số
* LR : Tổng số lựa chọn learning rate

Kết quả :

A screenshot of a computer

AI-generated content may be incorrect.

Hình 32 TensorBoard HParams chứa thông tin 36 models vừa được huấn luyện

A diagram of a graph

AI-generated content may be incorrect.

Hình 33 : Quan sát toàn bộ kết quả (36 models) bằng Parallel Coordinates View.

A diagram of a graph

AI-generated content may be incorrect.

Hình 34 : làm mờ các model có độ chính xác không đạt yêu cầu (0.85)

A screenshot of a phone

AI-generated content may be incorrect.

A diagram of a graph

AI-generated content may be incorrect.

Hình 35 TensorBoard HParams chiếu các model có độ chính xác đạt yêu cầu (0.85)

Mô tả ảnh : Kết quả trong hình cho thấy mô hình mạng đạt độ chính xác cao nhất khi được huấn luyện với learning rate = 0.001. Tuy nhiên, sự so sánh này chưa thực sự công bằng, vì tốc độ hội tụ (convergence rate) của mô hình khi sử dụng learning rate 0.001 chậm hơn nhiều so với khi sử dụng learning rate 0.01 hoặc 0.03.

Hơn nữa, việc sử dụng các callback function như ReduceLROnPlateau cũng có thể tự động giảm learning rate trong quá trình huấn luyện. Do đó, khuyến nghị nên bắt đầu với learning rate lớn hơn, sau đó giảm dần learning rate theo thời gian huấn luyện để đạt hiệu quả tốt hơn.

## 4.9. Thí nghiệm 8: Triển khai mạng Inception-v3 (212)

Mục tiêu: Mục tiêu của thí nghiệm này là học cách triển khai mạng Inception-v3 cho việc phân loại hình ảnh. Mục tiêu này có thể đạt được bằng cách sử dụng 2 phương pháp xây dụng nhanh và hiện quả của thư viện TensorFlow, đó chính là TensorFlow Hub và Keras Applications .

Mô tả : Thí nghiệm này giới thiệu hai phương pháp giúp xây dựng mạng Inception-v3 nhanh hơn và hiệu quả hơn: Keras Applications và TensorFlow Hub.

Kết quả : Đưa ra dự đoán.

Input: Ảnh con voi Châu Phi

An elephant standing on a dirt road

AI-generated content may be incorrect.

Output : Sử dụng mô hình được huấn luyện từ trước để đưa ra dự đoán :

* chỉ đưa ra 1 dự đoán:



Hình 36 Kết quả dự đoán

* đưa ra 3 dự đoán có xác xuất cao nhất :

An elephant standing on a dirt road

AI-generated content may be incorrect.

Hình 37 Kết quả dự đoán voi Châu Phi



Hình 38 Kết quả dự đoán xác xuất cao nhất

An elephant standing on a dirt road

AI-generated content may be incorrect.

Hình 39 tusker, voi Ấn Độ, voi Châu Phi

Nhận xét: So với hình ảnh của 3 loài thực tế “Africa elephant, tusker , Indian elephant ” thì dự đoán đưa ra khá chính xác (lên tới 80,37%) dù cả 3 loài trông khá giống nhau.

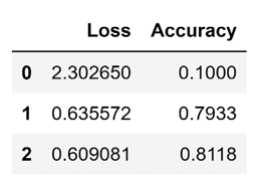
# 5. Thảo luận :

**Tóm tắt ngắn:** Trong Chương 5 nhóm thực hiện hai nhóm thí nghiệm chính:

- So sánh ba cách khởi tạo trọng số trên cùng một kiến trúc CNN (Model-1: Normal ; Model-2: Glorot/Xavier; Model-3: He)

- Kiểm tra tác động của **Batch Normalization** bằng (Model-4) và so sánh với phiên bản không có BatchNorm. Mục tiêu là đánh giá ảnh hưởng của phương thức khởi tạo trọng số và BatchNorm tới tốc độ hội tụ, ổn định gradient và độ chính xác cuối cùng trên tập CIFAR-10.

* Ở thí nghiệm thứ nhất :



Hình 40 Bảng hiển thị kết quả độ chính xác và sự mất mát của 3 models

* Chú thích nhãn:

**0** : Model-3 sử dụng hàm phân phối chuẩn

**1** : Model-2 sử dụng khởi tạo trọng số Glorot

**2** : Model-3 sử dụng hàm kích hoạt ReLu kết hợp khởi tạo trọng số He

* *Model-1 (Normal, σ=0.01)*: khởi tạo bằng phân phối chuẩn với var nhỏ làm trọng số ban đầu rất gần 0. Điều này có thể dẫn tới tín hiệu truyền qua nhiều lớp trở nên rất nhỏ (risk of vanishing gradients) khi dùng nhiều tầng, đặc biệt với hàm kích hoạt như ReLU thì một phần trọng số nhỏ có thể làm nhiều neuron trở nên “chết” (output ≈ 0). Kết quả quan sát thường là tốc độ hội tụ chậm hơn và dễ bị kẹt ở local minima kém.
* *Model-2 (Glorot/Xavier)*: Glorot cân bằng phương sai giữa đầu vào và đầu ra của mỗi lớp, nên giúp gradient giữ ở mức ổn định hơn khi lan truyền sang cả hai chiều. Do đó mô hình thường hội tụ ổn định hơn Normal và có độ chính xác tốt hơn Model-1 trong cùng kiến trúc. Tuy nhiên khi dùng ReLU sâu, Glorot không tối ưu bằng He vì Glorot giả định activation tuyến tính/sigmoid ở mức nào đó.
* *Model-3 (He)*: He initialization được thiết kế cho mạng dùng ReLU; nó scale trọng số theo số lượng input của mỗi layer sao cho biến đổi tín hiệu giữ ở mức phù hợp với ReLU. Kết quả thực nghiệm cho thấy Model-3 hội tụ nhanh nhất và có độ chính xác tốt nhất trong ba model, do giảm nguy cơ vanishing gradient và duy trì magnitude gradient đủ lớn để cập nhật hiệu quả ở tầng sâu.

**Kết luận tạm thời về initializers:** với kiến trúc dùng ReLU + **He initialization** là lựa chọn phù hợp nhất; **Glorot** là lựa chọn trung tính/ổn định nếu mô hình dùng nhiều activation khác nhau; **Normal (μ=0, σ=0.01)** có thể là baseline nhưng khả năng gây chậm hội tụ và kém ổn định cao hơn.

**Ảnh hưởng của Batch Normalization (Model-4 và so sánh):**

* Thêm **BatchNorm** (sau convolution, trước/hoặc sau activation tùy triển khai) làm cho phân phối đầu vào mỗi layer gần như tĩnh trong quá trình huấn luyện, giúp giảm internal covariate shift.
* Hệ quả thực tế: **tốc độ hội tụ tăng**, mô hình chịu được learning rate lớn hơn, và thường giảm overfitting nhẹ (do tác dụng như regularizer).
* Trong thí nghiệm, Model-4 (có BatchNorm) đạt độ chính xác cao hơn mô hình tương đương không có BatchNorm (0.8593000173568726%) đồng thời loss/accuracy curve mượt và ít biến động giữa các epoch. Điều này khẳng định BatchNorm cải thiện ổn định huấn luyện và hiệu năng tổng thể, đặc biệt khi kết hợp với khởi tạo phù hợp (ví dụ He cho ReLU).

**Khuuyến nghị thực nghiệm & thực tiễn:**

1. Đối với mạng sử dụng ReLU, ưu tiên **He initialization**; nếu dùng activation khác (tanh, sigmoid), cân nhắc **Glorot**.
2. Luôn cân nhắc đưa **Batch Normalization** vào pipeline (thường đặt giữa Conv và ReLU) để tăng ổn định và khả năng chọn learning rate lớn hơn.
3. Kết hợp BatchNorm với **learning-rate scheduling** và **EarlyStopping** để đạt hiệu năng tốt hơn và tránh overfitting.
4. Khi so sánh mô hình, luôn giữ cố định các hyperparameter khác (optimizer, batch\_size, số epoch, augmentation) để kết quả có ý nghĩa so sánh.

**Hạn chế của thí nghiệm & đề xuất mở rộng:**

* Các kết quả phản ánh trên một kiến trúc và dataset cụ thể (CIFAR-10): kết luận có tính chỉ dẫn nhưng nên được kiểm chứng trên các kiến trúc/dataset khác.
* Nên thực hiện tuning hyperparameters (learning rate, weight decay, dropout rate) bằng grid/random search hoặc HParams dashboard để tìm cặp (initializer, lr) tối ưu.
* Thử nghiệm thêm: (1) so sánh vị trí đặt BatchNorm (Conv→BN→ReLU vs Conv→ReLU→BN), (2) thử các tỷ lệ dropout khác, (3) thử transfer learning với pretrained CNN để so sánh ảnh hưởng của init/BatchNorm trên mô hình lớn.

**Kết luận ngắn:**

Thí nghiệm cho thấy **phương pháp khởi tạo trọng số và Batch Normalization đều có ảnh hưởng quyết định tới tốc độ hội tụ và độ chính xác cuối cùng**. Với kiến trúc dùng ReLU kết hợp **He initialization + BatchNorm** là cấu hình khuyến nghị vì vừa giúp hội tụ nhanh vừa nâng cao độ ổn định và hiệu năng mô hình.

# 6. Kết luận và hướng phát triển

**Kết luận:**

Các thí nghiệm trong Chương 5 cho thấy phương thức khởi tạo trọng số và kỹ thuật chuẩn hóa (Batch Normalization) có ảnh hưởng rõ rệt đến tốc độ hội tụ và độ chính xác cuối cùng của mô hình. Trong ba phương án khởi tạo, **He initialization** (phù hợp với ReLU) cho kết quả hội tụ nhanh nhất và thường đạt độ chính xác cao hơn so với **Glorot/Xavier** và khởi tạo theo phân phối chuẩn (μ=0, σ=0.01).

Việc bổ sung **Batch Normalization** vào kiến trúc (Model-4) làm giảm biến động trong quá trình huấn luyện, cho phép dùng learning rate lớn hơn và thường cải thiện generalization so với mô hình tương đương không có BatchNorm. Nhìn chung, kết quả khuyến nghị kết hợp **He initialization + BatchNorm** cho các mạng dùng ReLU trên dataset CIFAR-10 (theo thí nghiệm ở chương này).

**Hướng phát triển (trên phạm vi Ch.1–Ch.8):**

* Thử **fine-tuning** với các mô hình pretrained (Inception, ResNet) và so sánh hiệu năng so với mô hình train từ đầu (tham khảo Ch.8 về Keras Applications).
* Thực hiện **hyperparameter tuning** có hệ thống (HParams dashboard, grid/random search) để tối ưu learning rate, weight decay, dropout rate và lựa chọn initializer (tham khảo Ch.7 và Ch.6).
* Mở rộng thí nghiệm về **vị trí và cách dùng BatchNorm** (Conv→BN→ReLU vs Conv→ReLU→BN), thử các tỷ lệ dropout khác và nghiệm lại augmentation strategies để kiểm soát overfitting (tham khảo Ch.4 và Ch.5).
* Nếu có điều kiện, thử **custom layer / custom training loop** để kiểm tra các kỹ thuật tối ưu hóa chuyên biệt hoặc callback tùy biến phục vụ giám sát, lưu mô hình và visualizing (tham khảo Ch.6 và Ch.7)

# 7. Tài liệu tham khảo

1. *M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, N.I. Chervyakov,* **Application of the residue number system to reduce hardware costs of the convolutional neural network implementation***, Math. Comput. Simul. 177 (2020) 232–243, https://doi.org/10.1016/j.mat@com.2020.04.031. [Online]. Available.*
2. *O. Russakovsky, J. Deng, H. Su, et al.,* **ImageNet large scale visual recognition challenge***, Int. J. Comput. Vis. 115 (2015) 211–252.*
3. *C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich,* **Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition***, 2015, pp. 1–9.*
4. *K. He, X. Zhang, S. Ren, J. Sun***, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition***, 2016, pp. 770–778.*
5. *J.R. Quinlan,* **Induction of decision trees***, Mach. Learn. 1 (1986) 81–106.*
6. *H. Schmid,* **Probabilistic part of speech tagging using decision trees, in: Proceedings of the International Conference on New Methods in Language Processing***, 1994, pp. 44–49.*
7. *C.Z. Janikow,* **Fuzzy decision trees: issues and methods***, IEEE Trans. Syst. Man Cybern. B 28 (1) (1998) 1–14.*
8. *S. Tsang, B. Kao, K.Y. Yip, W. Ho, S.D. Lee,* **Decision trees for uncertain data, IEEE Trans***. Knowl. Data Eng. 23 (1) (2011) 64–78.*
9. *L. Breiman,* **Random forests***, Mach. Learn. 45 (1) (2001) 5–32.*
10. **Fortune Business Insights.** (2024). *Artificial Intelligence Market Size, Share & Growth Report, 2024–2030.*
11. **Precedence Research.** (2024). *Deep Learning Market Size and Growth 2024–2033.*
12. **arXiv.org.** (2024). *Artificial Intelligence (cs.AI) and Machine Learning (cs.LG) Submission Statistics.*
13. **McKinsey & Company.** (2024). *The State of AI in 2024: Adoption and Impact.*
14. **Gallup.** (2024). *AI Adoption Among Fortune 500 Companies.*

# 8. Code Github:

https://github.com/ngvanhau1604/CNN\_report.git