Hành Trình Học "Nhìn" Của Rô-bốt: Khám Phá Mạng Nơ-ron Tích Chập (CNN)

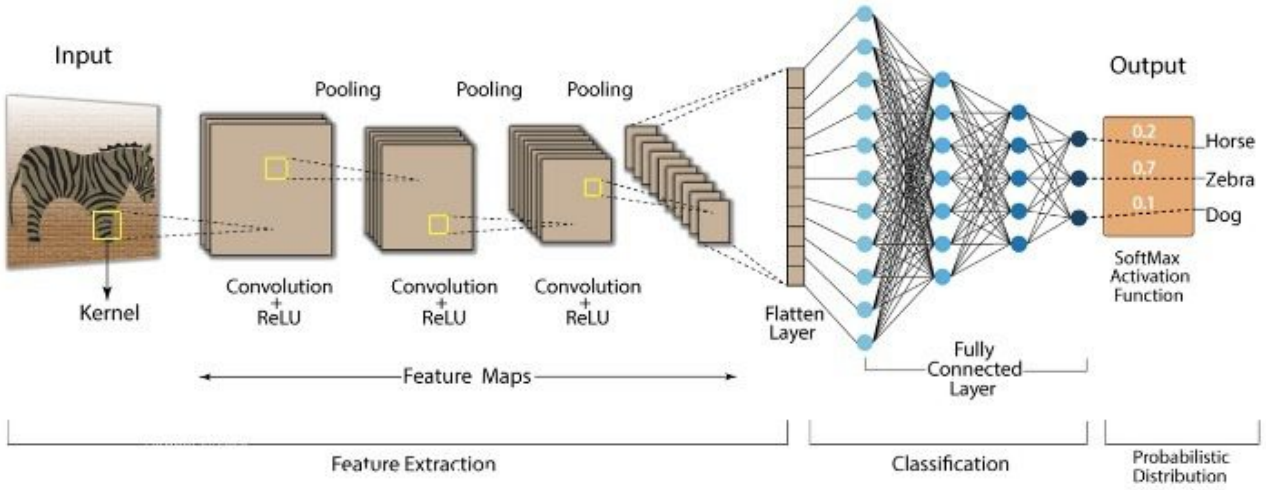
Người Kể Chuyện AI

Ngày 11 tháng 4 năm 2025

1 Lời Mở Đầu: Rô-bốt Muốn Học Nhìn

Xin chào các bạn nhỏ! Hôm nay chúng ta sẽ kể câu chuyện về một bạn rô-bốt rất thông minh. Bạn ấy muốn học cách "nhìn"và nhận biết mọi thứ xung quanh, y như con người chúng ta vậy. Ví dụ, bạn ấy muốn biết đâu là quả bóng, đâu là con mèo, đâu là cái bánh.

Để làm được điều này, các nhà khoa học đã dạy cho bạn rô-bốt một phương pháp đặc biệt gọi là **Mạng Nơ-ron Tích Chập (Convolutional Neural Network - CNN)**. Nghe tên thì hơi "ngầu"đúng không? Nhưng đừng lo, chúng ta sẽ khám phá nó qua những ví dụ thật dễ hiểu nhé! Hãy tưởng tượng bộ não của rô-bốt giống như một trò chơi xếp hình khổng lồ, và CNN là cách bạn ấy chơi trò chơi đó để hiểu được bức tranh.



Hình 1: Sơ đồ tổng quan về cách CNN hoạt động - Giống như các bước rô-bốt học nhìn.

2 Bước 1: Nhìn Qua Kính Lúp Đặc Biệt (Convolution - Conv2D)

2.1 Ý tưởng như một đứa trẻ 5 tuổi

Đầu tiên, để nhận biết một vật, ví dụ như con mèo trong bức ảnh, rô-bốt không nhìn toàn bộ bức ảnh cùng một lúc. Thay vào đó, bạn ấy dùng một chiếc "kính lúp thần kỳ"nhỏ xíu. Chiếc kính lúp này không phóng to hình ảnh, mà nó giúp rô-bốt tìm ra những **đặc điểm** (features) quan trọng.

Hãy tưởng tượng bức ảnh giống như một tờ giấy kẻ ô vuông, mỗi ô là một màu (pixel). Còn kính lúp là một tờ giấy nhỏ hơn cũng có các ô vuông, trên đó vẽ sẵn những hình thù đơn giản như đường thẳng, góc cong, chấm tròn... Rô-bốt sẽ đặt chiếc kính lúp này lên từng vùng nhỏ của bức ảnh, rồi di chuyển nó khắp nơi (đây gọi là **tích chập - convolution**).

Hình 2: Kính lúp (màu vàng) trượt trên bức ảnh (màu xanh lá) để tìm đặc điểm. Phép tính xảy ra ở mỗi vị trí.

Mỗi lần đặt kính lúp xuống, rô-bốt sẽ so sánh hình vẽ trên kính lúp với vùng ảnh bên dưới. Nếu chúng khớp nhau (ví dụ, kính lúp vẽ đường cong giống tai mèo, và vùng ảnh cũng có đường cong đó), rô-bốt sẽ ghi lại một điểm số cao vào một tờ giấy nháp mới. Nếu không khớp,

điểm số sẽ thấp. Tờ giấy nháp mới này gọi là **bản đồ đặc trưng (feature map)**, nó cho biết những đặc điểm mà kính lúp tìm thấy nằm ở đâu trong ảnh gốc.

2.2 Giải thích sâu hơn và Toán học

Lớp thực hiện nhiệm vụ này gọi là **Conv2D** (Convolution 2 Chiều - vì ảnh có chiều rộng và chiều cao). "Kính lúp"thực chất là một bộ lọc (filter) hay còn gọi là **kernel**. Kernel này là một ma trận nhỏ chứa các con số gọi là **trọng số (weights)**. Các trọng số này được học trong quá trình huấn luyện, để kernel biết cách nhận ra các đặc điểm hữu ích (như cạnh, góc, vân...).

Phép tích chập (Convolution) được thực hiện như sau: Kernel trượt trên ảnh đầu vào (Input Image - I). Tại mỗi vị trí, các giá trị trong kernel (K) sẽ được nhân tương ứng với các giá trị pixel trong vùng ảnh mà kernel đang che phủ, sau đó cộng tất cả kết quả lại.

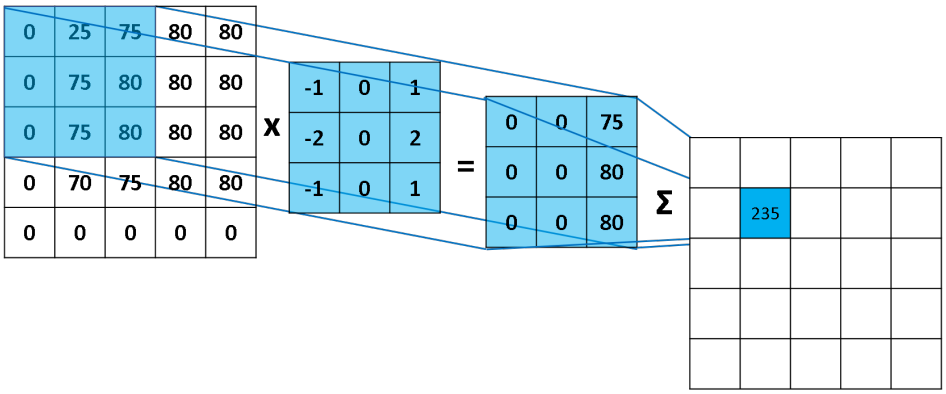
$$\text{Output}(i, j) = \sum_m \sum_n \text{Input}(i + m, j + n) \times \text{Kernel}(m, n)$$

Ở đây:

- $\text{Output}(i, j)$ là giá trị tại vị trí (i, j) trên bản đồ đặc trưng.
- $\text{Input}(i + m, j + n)$ là giá trị pixel của ảnh đầu vào tại vị trí tương ứng.
- $\text{Kernel}(m, n)$ là giá trị (trọng số) tại vị trí (m, n) trong kernel.
- Phép tổng $\sum$ là cộng tất cả các kết quả nhân lại.

Tại sao lại dùng phép tính này? Phép nhân và cộng này giúp "khuếch đại"tín hiệu ở những nơi mà vùng ảnh giống với mẫu hình trong kernel. Nếu một vùng ảnh có các giá trị pixel khớp với các trọng số cao trong kernel, kết quả phép nhân và tổng sẽ lớn, cho thấy đặc điểm đó xuất hiện rõ ràng.

Rô-bốt không chỉ dùng một kính lúp đâu! Nó có cả một bộ sưu tập kính lúp khác nhau, mỗi cái tìm một loại đặc điểm riêng (một cái tìm tai, cái tìm mắt, cái tìm ria mép...). Mỗi kính lúp sẽ tạo ra một bản đồ đặc trưng riêng.



Hình 3: Nhiều bộ lọc (kernels) khác nhau được áp dụng lên ảnh gốc, tạo ra nhiều bản đồ đặc trưng (feature maps).

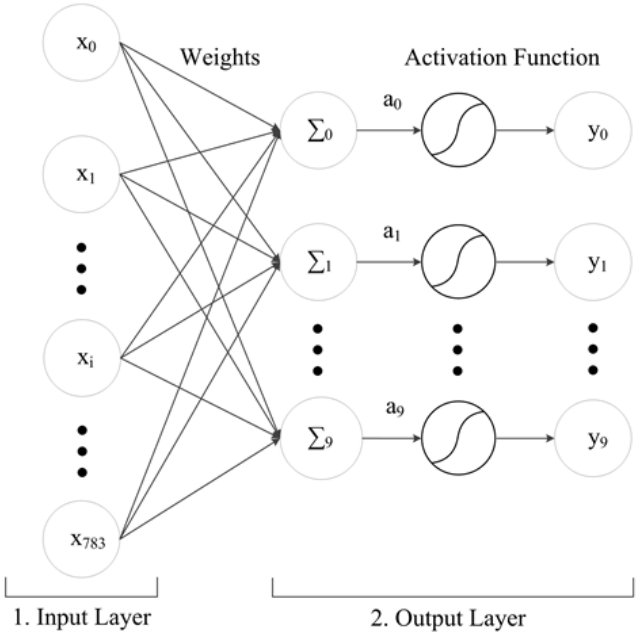
3 Bước 2: Quyết Định Điều Gì Quan Trọng (Activation Function - ReLU)

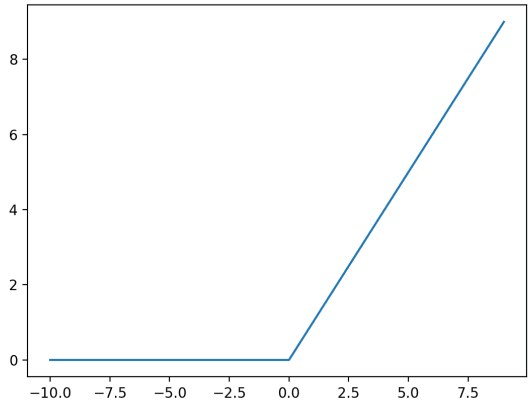
3.1 Ý tưởng như một đứa trẻ 5 tuổi

Sau khi dùng kính lúp và ghi lại điểm số vào các bản đồ đặc trưng, rô-bốt cần quyết định xem thông tin nào thực sự quan trọng. Giống như khi bạn chơi trò chơi, có những lúc bạn tìm thấy thứ gì đó nhưng nó không hữu ích lắm.

Rô-bốt sử dụng một quy tắc đơn giản gọi là **ReLU** (Rectified Linear Unit - Đơn vị Tuyến tính Chỉnh lưu). Quy tắc này giống như một người gác cổng:

- Nếu điểm số (giá trị trên bản đồ đặc trưng) là **dương** (lớn hơn 0), nghĩa là rô-bốt thấy đặc điểm đó khá rõ ràng, người gác cổng sẽ nói: "Ok, quan trọng đấy, giữ lại!"và giữ nguyên điểm số đó.
- Nếu điểm số là **âm hoặc bằng 0**, nghĩa là đặc điểm đó không rõ hoặc không có, người gác cổng sẽ nói: "Không quan trọng lắm, bỏ qua đi!"và đổi điểm số đó thành 0.





Hình 4: Đồ thị hàm ReLU (màu xanh dương). Giá trị âm thành 0, giá trị dương giữ nguyên.

Điều này giúp rô-bốt tập trung vào những tín hiệu mạnh mẽ nhất và loại bỏ những nhiễu không cần thiết.

3.2 Giải thích sâu hơn và Toán học

Hàm kích hoạt (Activation Function) như ReLU được áp dụng sau mỗi lớp tích chập (hoặc lớp kết nối đầy đủ). Mục đích chính của nó là đưa yếu tố **phi tuyến tính (non-linearity)** vào mạng.

Thế giới thực rất phức tạp và mối quan hệ giữa các điểm ảnh để tạo thành một vật thể (như con mèo) không phải lúc nào cũng là đường thẳng (tuyến tính). Nếu không có hàm kích hoạt phi tuyến, dù mạng có bao nhiêu lớp đi nữa, nó cũng chỉ tương đương với một lớp tuyến tính duy nhất và không thể học được các mối quan hệ phức tạp.

Hàm ReLU được định nghĩa rất đơn giản:

$$\text{ReLU}(x) = \max(0, x)$$

Nghĩa là:

- Nếu $x > 0$, thì $\text{ReLU}(x) = x$
- Nếu $x \leq 0$, thì $\text{ReLU}(x) = 0$

Tại sao lại là ReLU?

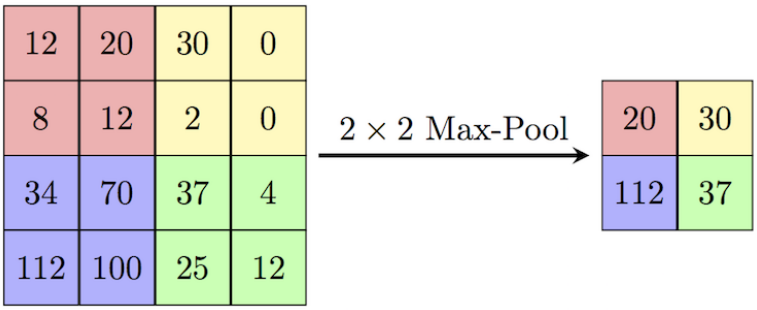
- **Tính toán đơn giản:** Chỉ cần so sánh với số 0, rất nhanh cho máy tính.
- **Tránh vấn đề Vanishing Gradient:** Trong các mạng sâu, các hàm kích hoạt cũ như Sigmoid hay Tanh có thể làm cho đạo hàm (gradient - thông tin để học) trở nên rất nhỏ khi đi qua nhiều lớp, khiến việc học bị chậm hoặc dừng lại. ReLU với đạo hàm bằng 1 cho các giá trị dương giúp giảm bớt vấn đề này.
- **Tạo ra sự thưa thớt (Sparsity):** Việc biến nhiều giá trị thành 0 giúp mạng trở nên "thưa"hơn, có thể hiệu quả hơn về mặt tính toán và đôi khi giúp khái quát hóa tốt hơn.

4 Bước 3: Tóm Tắt Thông Tin (Pooling - Max Pooling)

4.1 Ý tưởng như một đứa trẻ 5 tuổi

Sau khi rô-bốt đã có các bản đồ đặc trưng (đã qua "gác cổng"ReLU), trên đó đánh dấu những chỗ quan trọng. Nhưng các bản đồ này vẫn còn khá chi tiết và lớn. Rô-bốt cần làm cho chúng gọn gàng hơn, giống như bạn tóm tắt một câu chuyện dài thành một vài ý chính.

Bạn ấy sử dụng một phương pháp gọi là **Pooling**, và loại phổ biến nhất là **Max Pooling**. Nó giống như việc chia bản đồ đặc trưng thành các ô vuông nhỏ (ví dụ, ô 2x2). Trong mỗi ô vuông nhỏ đó, rô-bốt chỉ giữ lại con số **lớn nhất** (max) và bỏ đi các con số còn lại.



Hình 5: Max Pooling: Chia thành các vùng (ví dụ 2x2), giữ lại giá trị lớn nhất trong mỗi vùng.

Ví dụ, trong một ô 2x2 có các số 5, 1, 8, 2, rô-bốt chỉ giữ lại số 8. Làm như vậy cho tất cả các ô vuông nhỏ, rô-bốt sẽ có một bản đồ đặc trưng mới nhỏ gọn hơn rất nhiều, nhưng vẫn giữ được những thông tin quan trọng nhất (những tín hiệu mạnh nhất).

4.2 Giải thích sâu hơn và Toán học

Lớp Pooling, đặc biệt là Max Pooling, thường được đặt sau lớp ReLU. Mục tiêu chính của nó là **giảm kích thước không gian (chiều rộng và chiều cao)** của các bản đồ đặc trưng, từ đó:

- **Giảm số lượng tham số và tính toán** trong mạng, giúp mạng chạy nhanh hơn và đỡ tốn bộ nhớ hơn.
- **Tăng khả năng chống lại sự thay đổi nhỏ về vị trí** của đặc điểm (translation invariance). Nếu tai mèo hơi dịch chuyển một chút trong ô 2x2, giá trị lớn nhất có thể vẫn được giữ lại, giúp mạng nhận ra tai mèo dù vị trí không hoàn toàn chính xác.

Phép toán Max Pooling rất đơn giản: Chia bản đồ đặc trưng thành các vùng không chồng chéo (hoặc có chồng chéo) và lấy giá trị cực đại trong mỗi vùng đó.

$$\text{Output}(i,j) = \max_{(m,n)\in \text{Region}(i,j)} \text{Input}(m,n)$$

Trong đó $\text{Region}(i,j)$ là tập hợp các vị trí (m,n) trong vùng tương ứng với vị trí (i,j) của bản đồ đầu ra.

Tại sao lại hoạt động? Ý tưởng là giá trị lớn nhất trong một vùng nhỏ thường đại diện cho sự hiện diện mạnh mẽ nhất của một đặc điểm nào đó mà bộ lọc (kernel) đã phát hiện ra. Bằng cách giữ lại giá trị lớn nhất, chúng ta giữ lại thông tin quan trọng nhất về sự hiện diện của đặc điểm, đồng thời loại bỏ thông tin về vị trí chính xác của nó trong vùng đó, làm cho mạng trở nên linh hoạt hơn.

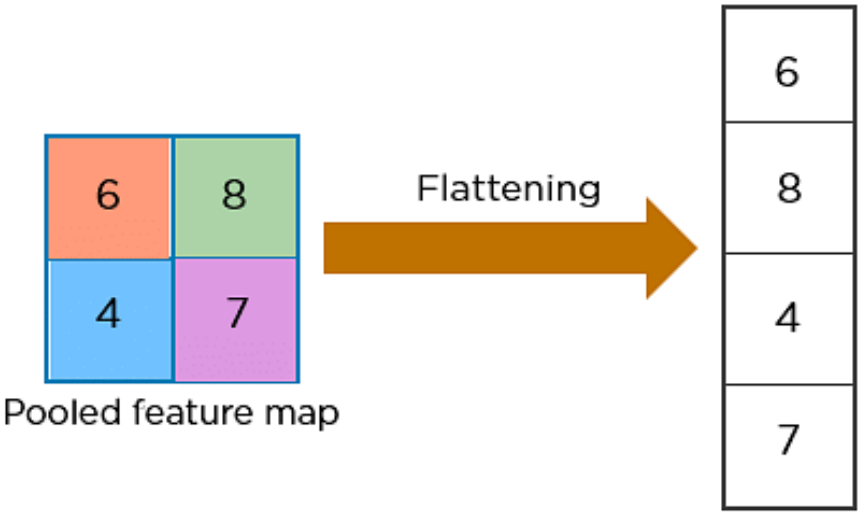
Các bước Convolution → ReLU → Pooling này thường được lặp lại nhiều lần. Mỗi lần lặp, các bộ lọc (kernels) có thể học được các đặc điểm phức tạp hơn dựa trên các đặc điểm đơn giản đã tìm thấy ở lớp trước. Ví dụ, lớp đầu tìm cạnh, góc; lớp sau kết hợp cạnh, góc để tìm hình vuông, hình tròn; lớp sau nữa kết hợp chúng để tìm mắt, mũi...

5 Bước 4: Xếp Hàng Tất Cả Thông Tin (Flattening)

5.1 Ý tưởng như một đứa trẻ 5 tuổi

Sau nhiều lần dùng kính lúp, qua gác cổng và tóm tắt, rô-bốt giờ có một chồng các bản đồ đặc trưng nhỏ gọn nhưng chứa đầy thông tin quan trọng. Bây giờ, để đưa ra quyết định cuối cùng (đây là con mèo hay con chó?), rô-bốt cần nhìn vào tất cả thông tin này cùng một lúc.

Hãy tưởng tượng các bản đồ đặc trưng giống như những trang giấy nhỏ. Rô-bốt sẽ lấy tất cả các trang giấy này và **trải phẳng (flatten)** chúng ra, nối chúng lại thành một dải giấy thật dài duy nhất. Dải giấy này chứa tất cả các con số quan trọng mà rô-bốt đã tìm thấy.



Hình 6: Lớp Flattening: Biến các bản đồ đặc trưng 2D thành một vector 1D duy nhất.

5.2 Giải thích sâu hơn

Lớp Flattening thực hiện một nhiệm vụ đơn giản nhưng cần thiết: chuyển đổi dữ liệu từ dạng đa chiều (ví dụ: chiều cao x chiều rộng x số kênh/bản đồ đặc trưng) thành dạng một chiều (một vector dài).

Ví dụ, nếu đầu ra của lớp Pooling cuối cùng là một tập hợp 64 bản đồ đặc trưng, mỗi cái có kích thước 7x7, thì lớp Flattening sẽ biến nó thành một vector có độ dài là $7 \times 7 \times 64 = 3136$.

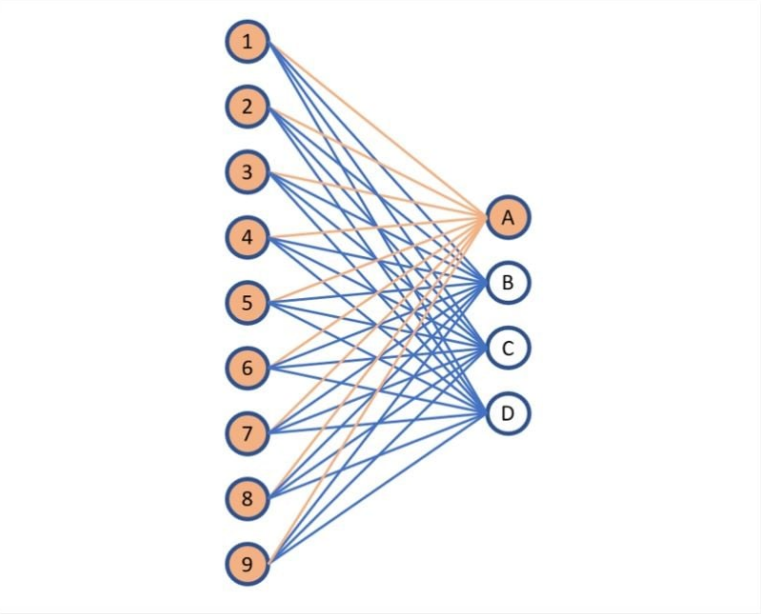
Tại sao cần Flattening? Các lớp tiếp theo trong mạng CNN thường là các lớp **Kết nối Đầy đủ (Fully Connected - FC)** hay còn gọi là lớp Dense. Các lớp này hoạt động trên dữ liệu đầu vào dạng vector một chiều. Do đó, Flattening là bước chuyển tiếp cần thiết để kết nối phần trích xuất đặc trưng (Convolution, ReLU, Pooling) với phần phân loại (Fully Connected).

6 Bước 5: Kết Nối Mọi Thứ Lại (Fully Connected Layer)

6.1 Ý tưởng như một đứa trẻ 5 tuổi

Bây giờ rô-bốt có một dải thông tin dài ngoằng từ bước Flattening. Tiếp theo, bạn ấy cần kết nối tất cả các mẩu thông tin này lại với nhau để đưa ra dự đoán. Giống như một thám tử nhìn vào tất cả manh mối (lông, tiếng kêu meo meo, hình dáng tai...) và suy luận: "Aha, tất cả những thứ này kết hợp lại giống một con mèo!"

Lớp **Kết nối Đầy đủ (Fully Connected - FC)** làm điều này. Hãy tưởng tượng mỗi con số trên dải giấy dài được nối với một vài "bóng đèn"ở lớp tiếp theo. Mỗi kết nối có một "độ mạnh"khác nhau (gọi là trọng số). Nếu một manh mối (con số) rất quan trọng để nhận ra con mèo, kết nối từ nó đến bóng đèn "Mèo"sẽ rất mạnh.



Hình 7: Lớp Fully Connected: Mỗi nơ-ron ở lớp trước được kết nối với mọi nơ-ron ở lớp sau.

Các bóng đèn này sẽ cộng tất cả tín hiệu từ các kết nối gửi đến (nhân giá trị manh mối với độ mạnh kết nối rồi cộng lại). Bóng đèn nào nhận được tín hiệu tổng cộng mạnh nhất sẽ sáng lên, cho biết rô-bốt nghĩ rằng đó là vật thể tương ứng. Có thể có nhiều lớp bóng đèn nối tiếp nhau để suy luận phức tạp hơn.

6.2 Giải thích sâu hơn và Toán học

Trong lớp Fully Connected (còn gọi là lớp Dense), mỗi nơ-ron (node) nhận đầu vào từ **tất cả** các nơ-ron ở lớp trước đó và kết nối đến **tất cả** các nơ-ron ở lớp tiếp theo (trừ lớp cuối cùng).

Phép toán tại mỗi nơ-ron trong lớp FC bao gồm: 1. Tính tổng có trọng số của tất cả các đầu vào từ lớp trước. 2. Cộng thêm một giá trị gọi là **thiên vị (bias)**. 3. Đưa kết quả qua một hàm kích hoạt (thường là ReLU cho các lớp ẩn FC, và một hàm khác như Softmax cho lớp cuối cùng).

$$\text{Output}_{\text{neuron } j} = \text{Activation}\left(\sum_i (\text{Input}_i \times \text{Weight}_{ij}) + \text{Bias}_j\right)$$

Ở đây:

- Input_i là đầu ra của nơ-ron thứ i ở lớp trước.
- Weight_{ij} là trọng số của kết nối từ nơ-ron i (lớp trước) đến nơ-ron j (lớp hiện tại).
- Bias_j là giá trị thiên vị của nơ-ron j.
- Activation là hàm kích hoạt được áp dụng.

Tại sao cần lớp FC? Sau khi các lớp tích chập và pooling đã trích xuất các đặc trưng không gian (như hình dạng, kết cấu) và làm cho chúng bất biến với vị trí, lớp FC có nhiệm vụ kết hợp các đặc trưng cấp cao này lại để thực hiện nhiệm vụ phân loại cuối cùng. Nó học cách "đọc"các đặc trưng đã được trích xuất và quyết định xem chúng tương ứng với lớp đối tượng nào (mèo, chó, ô tô...).

7 Bước 6: Đưa Ra Dự Đoán Cuối Cùng (Softmax)

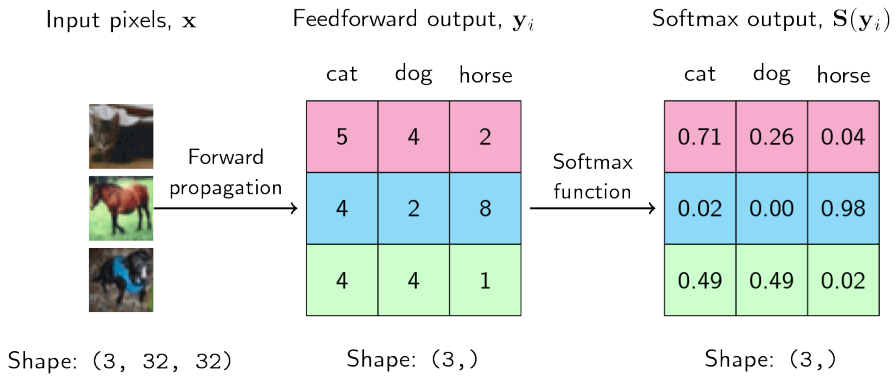
7.1 Ý tưởng như một đứa trẻ 5 tuổi

Sau khi các "bóng đèn"ở lớp cuối cùng đã tính toán xong, rô-bốt cần đưa ra câu trả lời cuối cùng. Ví dụ, nếu rô-bốt đang học cách phân biệt giữa "Mèo", "Chó", và "Vịt", lớp cuối cùng sẽ có 3 bóng đèn tương ứng.

Tuy nhiên, thay vì chỉ nói chắc nịch "Đây là Mèo!", rô-bốt sẽ thông minh hơn. Nó dùng một phép thuật gọi là **Softmax** để biến điểm số của các bóng đèn thành **xác suất**. Giống như rô-bốt nói:

- "Mình khá chắc đây là Mèo, khoảng 80%!"
- "Có một chút giống Chó, khoảng 15%."
- "Còn giống Vịt thì rất ít, chỉ 5% thôi."

Tổng các phần trăm này luôn luôn bằng 100% (hay 1). Bóng đèn nào có xác suất cao nhất sẽ là dự đoán cuối cùng của rô-bốt.



Hình 8: Softmax biến điểm số thô (logits) thành xác suất cho mỗi lớp (Mèo, Chó, Vịt). Tổng xác suất bằng 1.

7.2 Giải thích sâu hơn và Toán học

Softmax là một hàm kích hoạt đặc biệt thường được sử dụng ở lớp đầu ra (output layer) của mạng nơ-ron cho các bài toán **phân loại đa lớp (multi-class classification)**. Nó nhận vào một vector các giá trị số thực (gọi là logits - điểm số thô từ lớp FC trước đó) và chuyển đổi chúng thành một vector các xác suất.

Công thức của hàm Softmax cho đầu ra thứ i (tương ứng với lớp i) trong N lớp là:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Ở đây:

- z_i là điểm số (logit) đầu vào cho lớp thứ i.
- e^{z_i} là hàm mũ cơ số e của z_i. Hàm mũ giúp:
  - Đảm bảo tất cả các giá trị đầu ra đều dương (vì xác suất không thể âm).

– Khuếch đại sự khác biệt giữa các điểm số (điểm số cao hơn sẽ có e^{z_i} lớn hơn rất nhiều so với điểm số thấp hơn).

- $\sum_{j=1}^{N} e^{z_j}$ là tổng của tất cả các giá trị e^{z_j} cho tất cả N lớp. Việc chia cho tổng này đảm bảo rằng tất cả các xác suất đầu ra cộng lại bằng 1.

Tại sao dùng Softmax? Nó cung cấp một cách diễn giải kết quả đầu ra của mạng rất tự nhiên dưới dạng xác suất cho mỗi lớp có thể có. Điều này không chỉ cho chúng ta biết lớp nào được dự đoán có khả năng cao nhất mà còn cho biết "độ chắc chắn"của dự đoán đó so với các lớp khác.

8 Bước 7: Học Từ Lỗi Sai (Loss Function - Cross-Entropy)

8.1 Ý tưởng như một đứa trẻ 5 tuổi

Sau khi rô-bốt đoán xong ("Mình nghĩ 80% là Mèo!"), chúng ta cần cho bạn ấy biết bạn ấy đoán đúng hay sai, và sai nhiều hay ít. Giống như khi bạn chơi đoán đồ vật, người lớn sẽ nói "Đúng rồi!"hoặc "Sai rồi, đó là con chó cơ!".

Máy tính dùng một thứ gọi là **Hàm Mất Mát (Loss Function)** để đo xem dự đoán của rô-bốt "xa"với câu trả lời đúng đến mức nào. Đối với bài toán phân loại như thế này, người ta thường dùng hàm **Cross-Entropy Loss**.

Hãy tưởng tượng:

- Nếu câu trả lời đúng là "Mèo"(tức là 100% Mèo, 0% Chó, 0% Vịt).
- Rô-bốt đoán: 80% Mèo, 15% Chó, 5% Vịt.
- Hàm Cross-Entropy sẽ tính toán và thấy rằng dự đoán này khá tốt, nên "độ sai"(loss) sẽ nhỏ.
- Nếu rô-bốt đoán: 10% Mèo, 70% Chó, 20% Vịt.
- Hàm Cross-Entropy sẽ thấy dự đoán này sai nhiều, nên "độ sai"sẽ lớn. Giống như rô-bốt bị "phạt"nhiều hơn khi đoán sai nhiều.

Mục tiêu của rô-bốt là học cách điều chỉnh các kính lúp (kernels) và các kết nối (weights) của mình sao cho tổng "độ sai"này càng nhỏ càng tốt qua nhiều lần đoán.

8.2 Giải thích sâu hơn và Toán học

Hàm mất mát (Loss Function) định lượng sự khác biệt giữa kết quả dự đoán của mô hình ($\hat{y}$) và giá trị thực tế (y). Nó cung cấp một tín hiệu để hướng dẫn quá trình học của mô hình. **Categorical Cross-Entropy** thường được sử dụng cùng với Softmax trong phân loại đa lớp.

Công thức của Cross-Entropy Loss cho một mẫu dữ liệu là:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Ở đây:

- N là số lượng lớp.

- y_i là giá trị thực tế cho lớp i. Trong phân loại, đây thường là mã hóa **one-hot**. Ví dụ, nếu có 3 lớp (Mèo, Chó, Vịt) và mẫu thực tế là Mèo, thì $y = [1, 0, 0]$ (tức $y_1 = 1, y_2 = 0, y_3 = 0$).
- $\hat{y}_i$ là xác suất dự đoán bởi mô hình (đầu ra của Softmax) cho lớp i. Ví dụ, $\hat{y} = [0.8, 0.15, 0.05]$.
- log là logarit tự nhiên.

Tại sao lại là Cross-Entropy?

- **Đo lường sự khác biệt phân phối xác suất:** Nó đo lường khoảng cách giữa phân phối xác suất thực tế (one-hot y) và phân phối xác suất dự đoán ($\hat{y}$).
- **Phạt nặng các dự đoán sai và tự tin:** Hãy xem xét trường hợp $y_i = 1$ (đây là lớp đúng). Thành phần trong tổng sẽ là $-1 \times \log(\hat{y}_i)$.
  - Nếu mô hình dự đoán đúng và tự tin ($\hat{y}_i$ gần 1), thì $\log(\hat{y}_i)$ gần 0, và loss gần 0 (phạt ít).
  - Nếu mô hình dự đoán sai và tự tin ($\hat{y}_i$ gần 0), thì $\log(\hat{y}_i)$ tiến về âm vô cùng, và loss sẽ rất lớn (phạt nặng).
- **Kết hợp tốt với Softmax về mặt đạo hàm:** Khi tính toán đạo hàm của loss theo các trọng số (cần cho việc cập nhật trọng số), công thức trở nên đơn giản và ổn định về mặt số học.

Tổng loss trên toàn bộ tập dữ liệu huấn luyện thường là trung bình của loss trên từng mẫu.

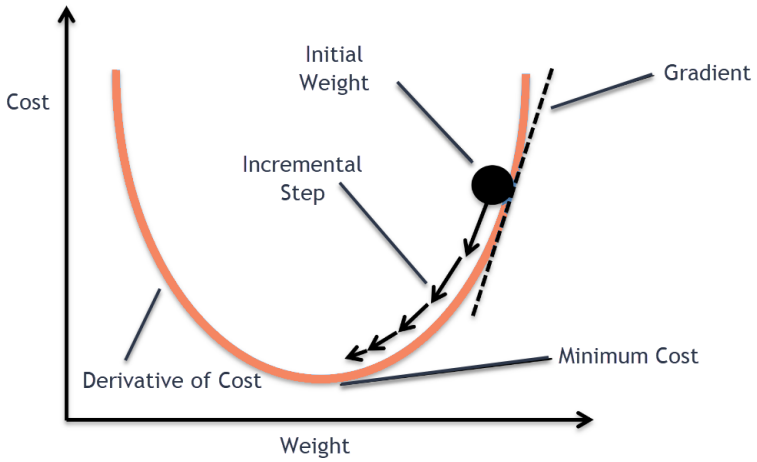
9 Bước 8: Cách Rô-bốt Học Giỏi Hơn (Optimizer - SGD)

9.1 Ý tưởng như một đứa trẻ 5 tuổi

Bây giờ rô-bốt biết mình đoán sai bao nhiêu (nhờ Hàm Mất Mát). Làm thế nào để bạn ấy sửa sai và đoán tốt hơn lần sau? Bạn ấy cần điều chỉnh những chiếc kính lúp (kernels) và độ mạnh các kết nối (weights) trong bộ não của mình.

Bạn ấy dùng một phương pháp gọi là **Stochastic Gradient Descent (SGD)** - Hạ Dốc Gradient Ngẫu Nhiên. Nghe phức tạp nhỉ? Hãy tưởng tượng rô-bốt đang đứng trên một ngọn đồi mù sương (ngọn đồi này tượng trưng cho "độ sai loss). Đỉnh đồi là nơi sai nhiều nhất, chân đồi là nơi sai ít nhất (mục tiêu cần đến). Vì mù sương, rô-bốt không thấy đường xuống chân đồi.

Bạn ấy làm thế nào? 1. Bạn ấy cảm nhận xem dốc nghiêng nhất ở đâu ngay dưới chân mình (đây gọi là tính **Gradient** - đạo hàm của hàm mất mát). Gradient chỉ hướng đi lên dốc nhất. 2. Bạn ấy bước một bước nhỏ theo hướng **ngược lại** với hướng dốc nhất (đi xuống dốc). 3. Bạn ấy lặp lại bước 1 và 2, từng bước nhỏ một, dần dần đi xuống chân đồi.



Hình 9: Gradient Descent: Đi từng bước nhỏ xuống dốc để tìm điểm có giá trị (loss) thấp nhất.

Chữ **Stochastic (Ngẫu nhiên)** ở đây có nghĩa là thay vì xem xét tất cả các bức ảnh cùng một lúc để quyết định bước đi (sẽ rất chậm), rô-bốt chỉ nhìn vào một vài bức ảnh ngẫu nhiên (một **batch** nhỏ) mỗi lần để ước lượng độ dốc và thực hiện một bước đi. Điều này giúp bạn ấy học nhanh hơn nhiều!

9.2 Giải thích sâu hơn và Toán học

Tối ưu hóa (Optimization) là quá trình điều chỉnh các tham số của mô hình (trọng số và bias) để giảm thiểu hàm mất mát. Gradient Descent là thuật toán tối ưu hóa nền tảng.

Ý tưởng cơ bản: 1. Tính toán **gradient** (đạo hàm riêng) của hàm mất mát L theo từng tham số θ (trọng số hoặc bias) của mô hình: $\nabla_\theta L$. Gradient là một vector chỉ hướng và tốc độ thay đổi lớn nhất của hàm mất mát. 2. Cập nhật tham số theo hướng ngược lại của gradient:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \nabla_\theta L$$

Ở đây:

- θ_{new} là giá trị mới của tham số.
- θ_{old} là giá trị cũ của tham số.
- η (eta) là **tốc độ học (learning rate)**: một siêu tham số (hyperparameter) nhỏ, dương, kiểm soát kích thước của mỗi bước cập nhật. Nếu η quá lớn, có thể bị "vọt"qua điểm tối ưu. Nếu quá nhỏ, việc học sẽ rất chậm.
- $\nabla_\theta L$ là gradient của hàm mất mát theo tham số θ. Dấu trừ $(-)$ đảm bảo chúng ta di chuyển theo hướng làm giảm loss.

Tại sao lại là "Stochastic"?

- **Batch Gradient Descent (chuẩn):** Tính gradient dựa trên toàn bộ tập dữ liệu huấn luyện. Chính xác nhưng rất chậm và tốn bộ nhớ với dữ liệu lớn.
- **Stochastic Gradient Descent (SGD):** Tính gradient và cập nhật tham số cho từng mẫu dữ liệu một. Nhanh hơn nhiều nhưng cập nhật rất "nhiễu"(dao động mạnh).

- **Mini-batch Gradient Descent (phổ biến nhất, thường gọi tắt là SGD):** Tính gradient và cập nhật tham số dựa trên một nhóm nhỏ (mini-batch) các mẫu dữ liệu (ví dụ: 32, 64, 128 mẫu). Đây là sự cân bằng tốt giữa tốc độ của SGD và sự ổn định của Batch GD.

SGD và các biến thể của nó (như Adam, RMSprop - thêm các cơ chế như momentum, adaptive learning rates) là động cơ chính giúp các mạng nơ-ron học hỏi từ dữ liệu. Quá trình tính gradient ngược từ lớp cuối cùng về các lớp đầu tiên được thực hiện hiệu quả bằng thuật toán **Lan truyền ngược (Backpropagation)**.

10 Kết Luận: Rô-bốt Đã Biết Nhìn!

Vậy là chúng ta đã cùng bạn rô-bốt trải qua hành trình học "nhìn"bằng Mạng Nơ-ron Tích chập (CNN): 1. Dùng kính lúp đặc biệt (Conv2D) để tìm đặc điểm. 2. Quyết định đặc điểm nào quan trọng (ReLU). 3. Tóm tắt thông tin cho gọn (Max Pooling). 4. (Lặp lại 1-2-3 nhiều lần để tìm đặc điểm phức tạp hơn). 5. Trải phẳng tất cả thông tin (Flattening). 6. Kết nối các thông tin lại để suy luận (Fully Connected). 7. Đưa ra dự đoán xác suất (Softmax). 8. Đo xem dự đoán sai bao nhiêu (Cross-Entropy Loss). 9. Học cách sửa sai bằng cách đi xuống dốc (SGD).

Bằng cách lặp đi lặp lại quá trình này với rất nhiều hình ảnh ví dụ (hàng nghìn, hàng triệu tấm ảnh!), rô-bốt dần dần học được cách điều chỉnh các trọng số trong kernel và các lớp FC để nhận biết đồ vật ngày càng chính xác hơn. Thật kỳ diệu phải không nào? CNN chính là một trong những công nghệ cốt lõi đằng sau rất nhiều ứng dụng thú vị như nhận dạng khuôn mặt trên điện thoại, xe tự lái, phân tích ảnh y tế và nhiều hơn nữa!