Sử dụng transfer learning: “Transfer learning là việc ứng dụng kỹ năng/tri thức mình học được từ vấn đề này (source domain – Ds), với ứng dụng này (source task – Ts) sang vấn đề khác (target domain -Dt) với ứng dụng khác (target task – Tt) có liên quan. Transfer learning nhằm cải thiện việc học hàm f(.) cho ứng dụng Tt trên miền Dt”

### **Convolution và Max Pooling trong CNN**

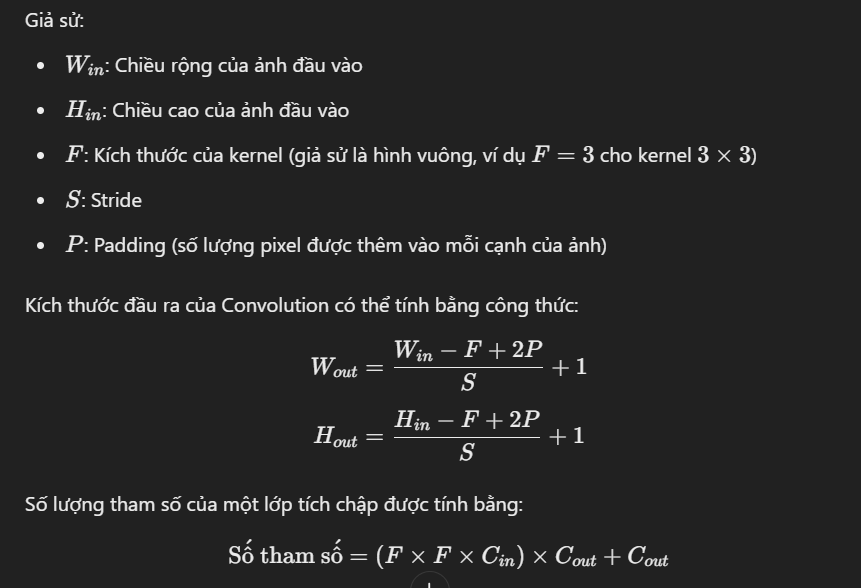
#### **1. Convolution (Tích chập)**

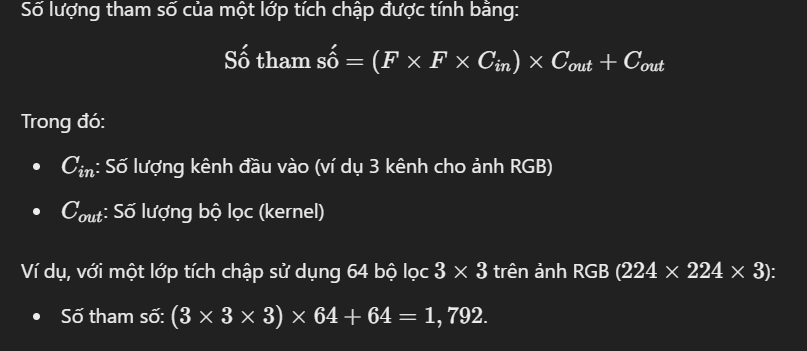
Convolution là một trong những thành phần chính của mạng nơ-ron tích chập (CNN) để trích xuất đặc trưng từ hình ảnh. Nó sử dụng các **kernel (bộ lọc)** di chuyển trên toàn bộ ảnh đầu vào để phát hiện các đặc trưng như cạnh, góc, hay các mẫu phức tạp hơn.

##### **Cách hoạt động của Convolution:**

* **Kernel (bộ lọc)**: Là một ma trận nhỏ (ví dụ 3×33 \times 33×3, 5×55 \times 55×5), chứa các tham số được học trong quá trình huấn luyện. Nó trượt qua toàn bộ ảnh đầu vào và tính toán **tích chập** giữa kernel và từng vùng nhỏ của ảnh.
* **Stride (bước nhảy)**: Quy định khoảng cách di chuyển của kernel sau mỗi lần tích chập. Stride mặc định là 1, tức là kernel di chuyển từng pixel một.
* **Padding**: Là việc thêm các giá trị 0 xung quanh biên của ảnh đầu vào để đảm bảo kích thước đầu ra của phép tích chập không bị giảm đi. Có hai loại padding:
  + **Valid padding**: Không thêm bất cứ giá trị nào, làm giảm kích thước đầu ra.
  + **Same padding**: Thêm giá trị 0 để đảm bảo kích thước đầu ra giống với kích thước đầu vào.

##### **Công thức tính kích thước đầu ra của Convolution:**

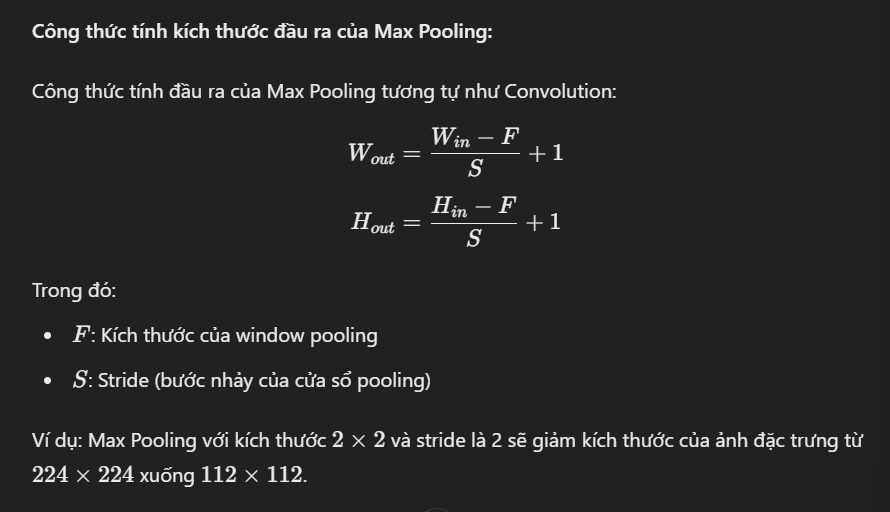




#### **2. Max Pooling (Lấy mẫu cực đại)**

Max Pooling được sử dụng để giảm kích thước không gian của ảnh đặc trưng, giúp giảm số lượng tham số và tài nguyên tính toán cần thiết, đồng thời giảm hiện tượng overfitting.

##### **Cách hoạt động của Max Pooling:**



* **Pooling window**: Tương tự như kernel trong convolution, nhưng ở đây chỉ đơn giản là lấy giá trị lớn nhất từ từng vùng của ảnh đặc trưng. Kích thước phổ biến là 2×22 \times 22×2, với stride là 2.
* Max Pooling giúp giảm kích thước ảnh đầu vào trong khi vẫn giữ lại những đặc trưng quan trọng.

##### **Công thức tính kích thước đầu ra của Max Pooling:**

### **Tóm lại**

* **Convolution** giúp trích xuất đặc trưng từ ảnh đầu vào bằng cách áp dụng các bộ lọc.
* **Max Pooling** giúp giảm kích thước của ảnh đặc trưng, giữ lại các đặc trưng quan trọng và giảm độ phức tạp của mô hình.
* Cả hai đều có công thức tính kích thước đầu ra để xác định kích thước của ảnh sau khi thực hiện mỗi bước.

### **1. VGG16 là gì?**

VGG16 là một kiến trúc mạng nơ-ron tích chập (CNN) do nhóm nghiên cứu tại Đại học Oxford phát triển. Mô hình này nổi bật với việc sử dụng các lớp tích chập kích thước nhỏ 3×33 \times 33×3 và có độ sâu lên tới 16 lớp. Nó được sử dụng phổ biến trong các tác vụ nhận dạng hình ảnh.

### **2. VGG16 có cấu trúc như thế nào?**

VGG16 có tổng cộng 16 lớp có trọng số:

* 13 lớp tích chập (convolutional layers)
* 3 lớp fully connected (lớp kết nối hoàn toàn) Ngoài ra, nó có các lớp pooling xen kẽ giữa các lớp tích chập để giảm kích thước không gian của ảnh.

### **4. Tại sao lại sử dụng kernel 3×33 \times 33×3 trong VGG16?**

Sử dụng kernel 3×3 trong VGG16 giúp giảm số lượng tham số cần học mà vẫn duy trì khả năng trích xuất các đặc trưng quan trọng từ ảnh. Các bộ lọc nhỏ như 3×3 giúp mô hình tập trung vào các vùng nhỏ của ảnh, giúp học tốt hơn các đặc trưng như cạnh, góc và các mẫu phức tạp hơn.

### **10. VGG16 có thể được sử dụng cho các tác vụ khác ngoài nhận dạng hình ảnh không?**

Ngoài nhận dạng hình ảnh, VGG16 có thể được sử dụng trong các bài toán liên quan đến thị giác máy tính khác như:

* **Phân đoạn ảnh** (Image Segmentation)
* **Phát hiện đối tượng** (Object Detection)
* **Nhận dạng khuôn mặt** (Face Recognition)

Giải thích mô hình:

trainable là một thuộc tính trong TensorFlow và Keras dùng để chỉ định liệu các tham số (trọng số và bias) của một lớp hoặc mô hình có được phép cập nhật trong quá trình huấn luyện hay không.

### **Cụ thể:**

* **trainable = True**: Các tham số của lớp hoặc mô hình sẽ được huấn luyện, tức là sẽ thay đổi dựa trên việc tối ưu hóa hàm loss trong quá trình backpropagation. Điều này giúp mô hình học và điều chỉnh trọng số để cải thiện kết quả.
* **trainable = False**: Các tham số của lớp hoặc mô hình sẽ được cố định và **không** thay đổi trong quá trình huấn luyện. Thường thì bạn sử dụng điều này khi muốn giữ nguyên các trọng số đã được huấn luyện trước đó (pre-trained) và không muốn mô hình học lại từ đầu.

### **Ví dụ:**

* Khi bạn tải mô hình **VGG16** với các trọng số được huấn luyện từ trước (pre-trained) trên bộ dữ liệu ImageNet, các trọng số này đã học được những đặc trưng chung của hình ảnh. Nếu bạn đặt trainable = False, các trọng số này sẽ không bị thay đổi trong quá trình huấn luyện và chỉ các lớp phía trên mà bạn thêm vào sẽ được huấn luyện.

### **Khi nào sử dụng trainable = False?**

* Khi bạn sử dụng **transfer learning** và chỉ muốn thêm một số lớp để tùy chỉnh mô hình cho bài toán cụ thể mà không muốn mô hình học lại từ đầu.
* Để giảm thời gian huấn luyện, vì nếu các lớp của mô hình gốc không thay đổi, bạn chỉ cần huấn luyện các lớp mới.
* Để tránh overfitting khi dữ liệu của bạn ít và bạn muốn sử dụng các trọng số đã được học từ bộ dữ liệu lớn.

**include\_top=True**: Mô hình sẽ bao gồm các lớp fully connected ở phần trên cùng (tức là phần cuối cùng của mạng nơ-ron), thường được thiết kế cho bài toán phân loại với 1000 lớp của ImageNet. Đối với VGG16, đây là các lớp Dense với tổng cộng 138,357,544 tham số, phù hợp với bài toán phân loại 1000 lớp trên ImageNet.

**include\_top=False**: Mô hình sẽ **bỏ qua** các lớp fully connected ở phần "top", chỉ giữ lại các lớp convolution để trích xuất đặc trưng. Điều này hữu ích khi bạn muốn sử dụng mạng nơ-ron đã pre-trained nhưng áp dụng cho một bài toán khác với số lớp khác (ví dụ như phân loại 10 lớp thay vì 1000 lớp).

Giải thích về batch normalization

### **1. Batch Normalization (BatchNorm)**

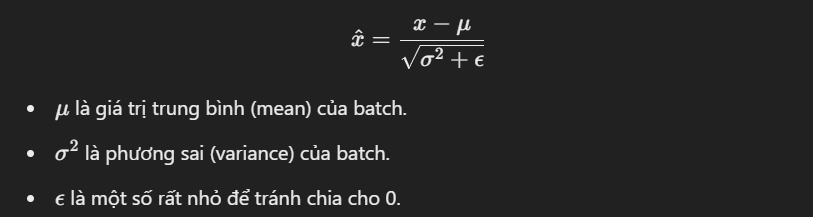
#### **Mục đích:**

Batch Normalization (viết tắt là BatchNorm) là một kỹ thuật dùng để chuẩn hóa (normalize) đầu vào của mỗi lớp trong mạng nơ-ron (hoặc có thể đầu vào của mô hình) để làm cho quá trình huấn luyện nhanh hơn và ổn định hơn. Cụ thể, nó giúp giảm thiểu vấn đề **internal covariate shift**, tức là sự thay đổi trong phân phối của các đầu vào giữa các lớp khi các tham số trong mô hình thay đổi trong quá trình huấn luyện.

#### **Cách hoạt động:**

BatchNorm thực hiện việc chuẩn hóa đầu ra của một lớp bằng cách:

1. **Tính toán giá trị trung bình (mean)** và **phương sai (variance)** của các đầu vào trong từng batch dữ liệu.
2. **Chuẩn hóa** các đầu vào trong batch đó, tức là biến đầu vào thành phân phối chuẩn với **mean bằng 0** và **phương sai bằng 1**:





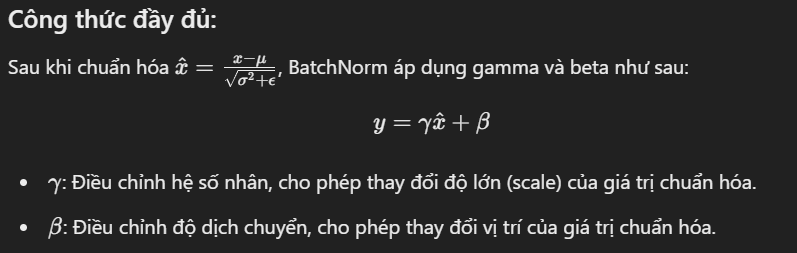
Ví dụ: nếu batch size =32=> Đúng vậy! Khi sử dụng Batch Normalization (BatchNorm), nếu batch size của bạn là 32, thì mỗi batch có 32 mẫu sẽ được chuẩn hóa riêng. Điều này có nghĩa là các giá trị trung bình (mean) và phương sai (variance) của từng batch sẽ khác nhau, và quá trình chuẩn hóa diễn ra **theo từng batch một cách độc lập**. Do đó, mỗi batch sẽ có các giá trị chuẩn hóa khác nhau

#### **Tại sao cần hệ số scale (γ) và shift (β)?**

Sau khi chuẩn hóa dữ liệu (đưa về phân phối chuẩn với mean = 0 và variance = 1), mô hình có thể bị giới hạn bởi giá trị đầu ra của lớp BatchNorm. Nếu chỉ đơn giản là chuẩn hóa, đầu ra sẽ luôn nằm trong một phạm vi rất hẹp (quanh giá trị 0 với phương sai nhỏ). Điều này có thể hạn chế khả năng của mạng nơ-ron trong việc học các biểu diễn phức tạp hơn, vì không phải lúc nào dữ liệu đầu vào của các lớp tiếp theo cũng cần có giá trị trung bình bằng 0 và phương sai bằng 1.

Do đó, sau khi chuẩn hóa, hai tham số **gamma (γ)** và **beta (β)** được thêm vào để điều chỉnh lại phân phối dữ liệu sau khi chuẩn hóa. Cụ thể:

* **Gamma (γ)**: Tham số **scale**, điều chỉnh độ lệch chuẩn của dữ liệu sau khi chuẩn hóa. Nó giúp thay đổi mức độ phân tán của dữ liệu.
* **Beta (β)**: Tham số **shift**, điều chỉnh giá trị trung bình của dữ liệu sau khi chuẩn hóa.

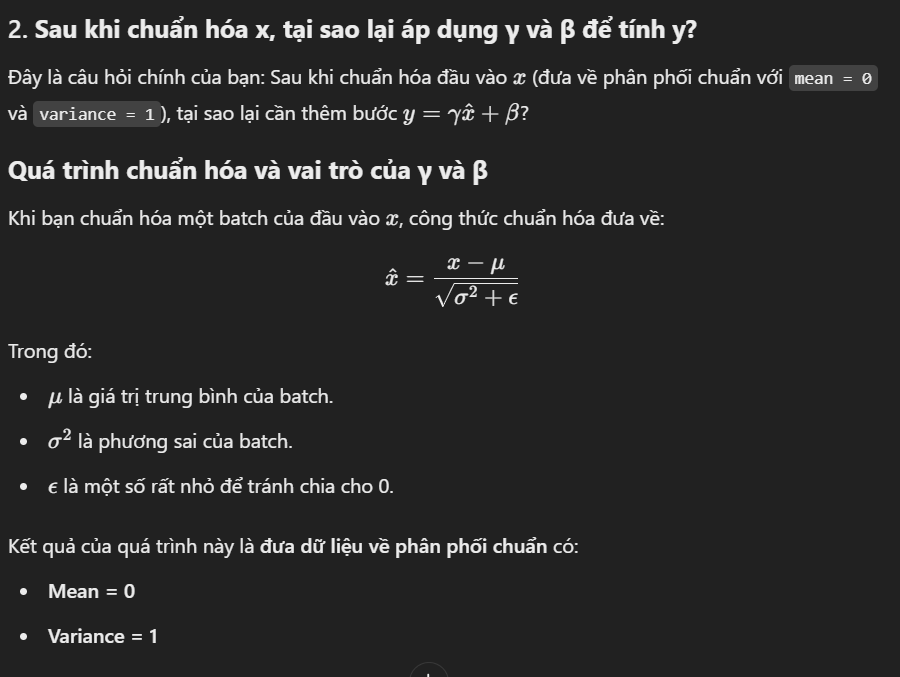


### **1. Gamma (γ) và Beta (β) từ đâu ra?**

**Gamma (γ)** và **Beta (β)** là **các tham số có thể học được** (trainable parameters) của mô hình. Tức là, chúng **không được cung cấp từ trước**, mà được mô hình **tự động học** thông qua quá trình huấn luyện. Khi bạn thêm một lớp Batch Normalization vào mô hình, TensorFlow (hoặc bất kỳ framework deep learning nào khác) sẽ tự động thêm các tham số γ và β để mô hình có thể học cách điều chỉnh phân phối của dữ liệu trong mỗi lớp.

Nói cách khác:

1. **Gamma (γ)** và **Beta (β)** được **khởi tạo ngẫu nhiên** (thường là γ = 1 và β = 0 lúc ban đầu), và sau đó được **tối ưu** (learning) trong suốt quá trình training như các tham số thông thường khác của mạng nơ-ron.
2. Trong quá trình huấn luyện, thông qua quá trình lan truyền ngược (backpropagation), các giá trị γ và β sẽ được cập nhật liên tục để tối ưu hóa quá trình chuẩn hóa.



#### **Tại sao chỉ chuẩn hóa không thôi là chưa đủ?**

Khi dữ liệu được chuẩn hóa với mean = 0 và variance = 1, tất cả các giá trị sẽ bị "nén" trong một phạm vi rất hẹp (quanh giá trị 0). Điều này có thể **giới hạn khả năng học của mô hình** vì không phải lúc nào các lớp tiếp theo cũng cần dữ liệu với mean = 0 và variance = 1.

* **Vấn đề tiềm ẩn**: Nếu bạn chỉ sử dụng giá trị chuẩn hóa x^, dữ liệu đầu vào của lớp tiếp theo sẽ luôn có giá trị trung bình là 0 và độ lệch chuẩn là 1. Điều này sẽ hạn chế mạng nơ-ron trong việc học các mối quan hệ phức tạp hơn giữa các đặc trưng.

### **Gamma (γ) và Beta (β) khắc phục điều này như thế nào?**

Để giải quyết vấn đề trên, ta sử dụng thêm các hệ số **gamma** (γ) và **beta** (β) để cho phép mô hình tự điều chỉnh phân phối của dữ liệu chuẩn hóa. Công thức đầy đủ sau khi chuẩn hóa là:

y=γx^+β

* **Gamma (γ)**: Tham số **scale**, giúp thay đổi độ lớn của các giá trị sau khi chuẩn hóa. Điều này cho phép mạng nơ-ron điều chỉnh độ phân tán của dữ liệu sau khi chuẩn hóa.
* **Beta (β)**: Tham số **shift**, giúp dịch chuyển giá trị trung bình của dữ liệu sau chuẩn hóa.

### **Lợi ích của việc sử dụng γ và β:**

1. **Gamma (γ)** cho phép mô hình tự quyết định liệu dữ liệu có cần phân tán(độ lệch chuẩn) rộng hơn hay không.
   * Nếu **γ = 1**, dữ liệu sẽ giữ nguyên độ phân tán như sau khi chuẩn hóa (giữ độ lệch chuẩn là 1).
   * Nếu **γ > 1**, dữ liệu sẽ được "mở rộng", giúp tăng độ phân tán.
   * Nếu **γ < 1**, dữ liệu sẽ bị "thu hẹp".
2. **Beta (β)** cho phép mô hình tự quyết định liệu dữ liệu có cần dịch chuyển khỏi giá trị trung bình 0 hay không.
   * Nếu **β = 0**, dữ liệu sẽ giữ nguyên giá trị trung bình là 0.
   * Nếu **β > 0**, dữ liệu sẽ dịch sang phải, có nghĩa là giá trị trung bình sẽ tăng.
   * Nếu **β < 0**, dữ liệu sẽ dịch sang trái, có nghĩa là giá trị trung bình sẽ giảm.

Như vậy, γ và β giúp tăng **sự linh hoạt** trong việc học biểu diễn của dữ liệu. Thay vì chỉ đưa dữ liệu về phân phối chuẩn, mô hình có thể tự điều chỉnh phân phối sao cho phù hợp với nhiệm vụ cụ thể.

### **3. Kết luận:**

* **BatchNorm** chuẩn hóa dữ liệu để đảm bảo rằng quá trình huấn luyện ổn định và nhanh hơn.
* **Gamma (γ)** và **Beta (β)** là tham số giúp mô hình không bị giới hạn bởi việc dữ liệu luôn phải có mean = 0 và variance = 1.
* Mô hình sẽ tự học cách điều chỉnh giá trị của **gamma** và **beta** để tạo ra các biểu diễn tối ưu nhất cho từng layer trong quá trình học.

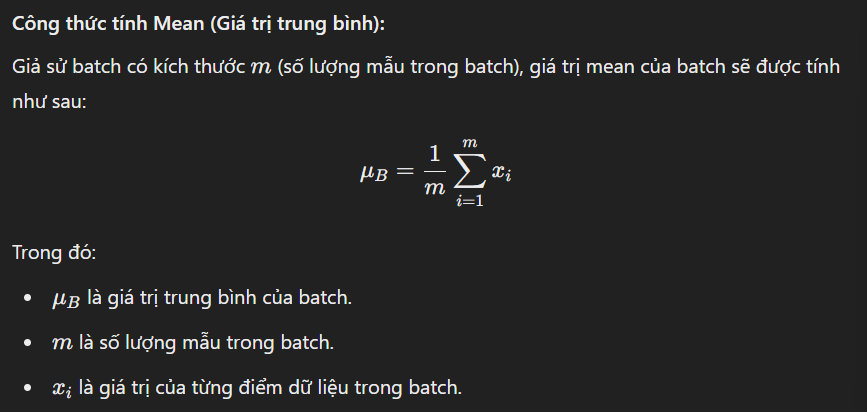
**Giá trị y của lớp batchnorm**

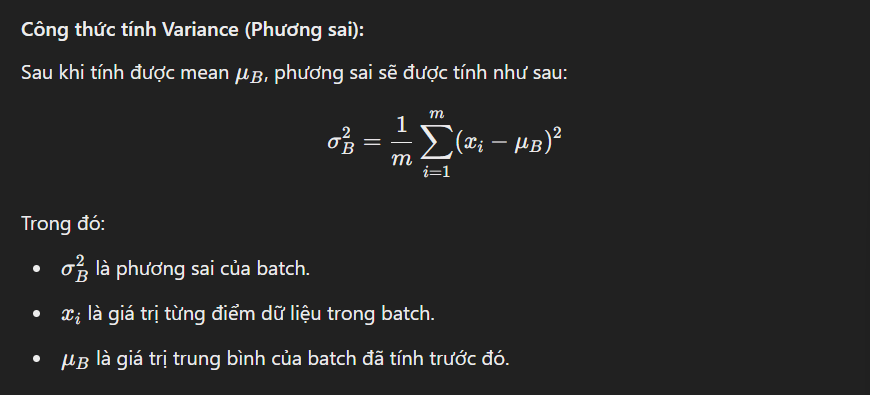
y là **kết quả đầu ra của một lớp trong mạng nơ-ron(batch norm)**, và nó sẽ được truyền qua các lớp tiếp theo trong mô hình.

### **Mục tiêu của việc tính toán yyy:**

* y giúp đảm bảo rằng dữ liệu đầu vào của các lớp tiếp theo có **phân phối ổn định**, tức là không bị quá lệch về giá trị lớn hoặc nhỏ, giúp mô hình học dễ hơn.

Tính variance và mean





### **1. Tác động của Gamma (γ):**

* **Gamma (γ)** là tham số điều chỉnh **phân tán (scale)** của dữ liệu sau khi chuẩn hóa. Nếu **γ** lớn hoặc nhỏ sẽ làm thay đổi độ rộng của phân phối dữ liệu.

#### **Tăng gamma (γ):**

* Khi tăng **gamma**, phân phối của dữ liệu sẽ trở nên **rộng hơn**, tức là các giá trị sau khi chuẩn hóa sẽ được **phóng đại**. Điều này làm cho dữ liệu có độ biến thiên cao hơn, các điểm dữ liệu sẽ cách xa giá trị trung bình nhiều hơn.
* **Tác động đến quá trình học**: Khi tăng gamma, mô hình có thể học được các đặc trưng mạnh mẽ hơn vì các giá trị kích hoạt (activation) sẽ trở nên lớn hơn. Tuy nhiên, nếu tăng gamma quá nhiều, điều này có thể dẫn đến overfitting, vì mô hình sẽ nhạy cảm hơn với các thay đổi nhỏ trong dữ liệu.

#### **Giảm gamma (γ):**

* Khi giảm **gamma**, phân phối của dữ liệu sẽ **thu hẹp**, tức là các giá trị sau khi chuẩn hóa sẽ bị **nén lại** gần với giá trị trung bình.
* **Tác động đến quá trình học**: Khi giảm gamma, mô hình có thể trở nên **ít nhạy cảm** hơn với các thay đổi nhỏ trong dữ liệu. Điều này có thể giúp mô hình ổn định hơn và tránh overfitting, nhưng nếu gamma quá nhỏ, mô hình có thể gặp khó khăn trong việc học các đặc trưng quan trọng, dẫn đến underfitting.

### **2. Tác động của Beta (β):**

* **Beta (β)** là tham số điều chỉnh **dịch chuyển (shift)** của dữ liệu sau khi chuẩn hóa. Khi thay đổi **beta**, giá trị trung bình của dữ liệu sau chuẩn hóa sẽ thay đổi theo hướng dịch sang phải hoặc trái.

#### **Tăng beta (β):**

* Khi tăng **beta**, các giá trị sau khi chuẩn hóa sẽ được dịch **sang phải**, tức là giá trị trung bình của các giá trị sau chuẩn hóa sẽ lớn hơn.
* **Tác động đến quá trình học**: Việc dịch chuyển dữ liệu sau chuẩn hóa có thể giúp mô hình học được các đặc trưng ở khoảng giá trị lớn hơn. Điều này có thể phù hợp với một số mạng nơ-ron nơi các giá trị kích hoạt cần phải lớn hơn một ngưỡng nhất định để mô hình học tốt hơn.

#### **Giảm beta (β):**

* Khi giảm **beta**, dữ liệu sau chuẩn hóa sẽ được dịch **sang trái**, tức là giá trị trung bình của các giá trị sau chuẩn hóa sẽ nhỏ hơn.
* **Tác động đến quá trình học**: Dịch chuyển giá trị trung bình xuống mức nhỏ hơn có thể phù hợp với các mạng nơi mà các giá trị nhỏ sẽ làm cho mạng ít kích hoạt hơn, từ đó có thể giúp mô hình học các đặc trưng khác nhau tùy vào việc các lớp kích hoạt sau BatchNorm sử dụng.

Dropout

**Dropout** là một kỹ thuật nhằm ngăn chặn **overfitting** trong quá trình huấn luyện mạng nơ-ron. Nó hoạt động bằng cách **bỏ qua ngẫu nhiên** một tỷ lệ phần trăm của các đơn vị (neuron) trong mạng trong mỗi lần cập nhật trọng số trong quá trình huấn luyện.

### **Cách hoạt động của Dropout:**

* Khi huấn luyện, Dropout sẽ **tạm thời vô hiệu hóa** (bỏ qua) một số lượng neuron nhất định trong mỗi layer, dựa trên tỷ lệ mà bạn đã chỉ định. Các neuron bị vô hiệu hóa sẽ **không tham gia vào việc truyền thông tin và tính toán trọng số** trong lần huấn luyện đó.

### **Ý nghĩa của tham số 0.25:**

Khi bạn truyền tham số **0.25** vào Dropout, điều này có nghĩa là **25% neuron** ở layer tương ứng sẽ bị bỏ qua ngẫu nhiên trong mỗi lần cập nhật trọng số.

### **Tại sao dùng Dropout?**

Dropout giúp mô hình **không phụ thuộc quá mức** vào bất kỳ một nhóm nhỏ các neuron nào. Việc vô hiệu hóa ngẫu nhiên một tỷ lệ neuron buộc mô hình phải tìm ra các đặc trưng khác nhau và xây dựng các liên kết mạnh mẽ giữa nhiều neuron hơn. Điều này làm cho mô hình **tổng quát hóa tốt hơn** khi áp dụng vào dữ liệu mới (không gặp phải hiện tượng overfitting).

### **Dropout trong quá trình huấn luyện và dự đoán:**

* **Trong quá trình huấn luyện**: Khi Dropout được áp dụng, mỗi lần cập nhật trọng số, một tỷ lệ neuron ngẫu nhiên sẽ bị bỏ qua (như đã nói ở trên).
* **Trong quá trình dự đoán (inference)**: Dropout sẽ **không được áp dụng**. Tức là, tất cả các neuron sẽ hoạt động như bình thường và mạng sẽ sử dụng tất cả trọng số của mình để đưa ra kết quả.

Các cách khai báo trọng số bằng phân phối :

### **1. He Normal Initialization (he\_normal)**

* **Mô tả**: Phương pháp khởi tạo He Normal được giới thiệu bởi Kaiming He và cộng sự. Đây là một cách khởi tạo trọng số cho các lớp mạng nơ-ron sử dụng phân phối chuẩn.
* **Công thức**: Các trọng số được khởi tạo bằng cách lấy mẫu từ phân phối chuẩn với phương sai là 2n\frac{2}{n}n2​, trong đó nnn là số lượng neuron đầu vào của lớp hiện tại.  
  weights∼N(0,2n)\text{weights} \sim \mathcal{N}\left(0, \frac{2}{n}\right)weights∼N(0,n2​)
* **Ưu điểm**: Thích hợp cho các hàm kích hoạt ReLU và các biến thể của nó. Giúp ngăn ngừa vấn đề gradient vanishing hoặc exploding gradients trong quá trình huấn luyện.

### **2. Glorot Normal Initialization (glorot\_normal)**

* **Mô tả**: Phương pháp khởi tạo Glorot, còn được gọi là Xavier Normal, được đề xuất bởi Xavier Glorot và Yoshua Bengio. Đây là một cách khởi tạo trọng số cho các lớp mạng nơ-ron sử dụng phân phối chuẩn.
* **Công thức**: Các trọng số được khởi tạo bằng cách lấy mẫu từ phân phối chuẩn với phương sai là 2nin+nout\frac{2}{n\_{\text{in}} + n\_{\text{out}}}nin​+nout​2​, trong đó ninn\_{\text{in}}nin​ là số lượng neuron đầu vào và noutn\_{\text{out}}nout​ là số lượng neuron đầu ra của lớp hiện tại.  
  weights∼N(0,2nin+nout)\text{weights} \sim \mathcal{N}\left(0, \frac{2}{n\_{\text{in}} + n\_{\text{out}}}\right)weights∼N(0,nin​+nout​2​)
* **Ưu điểm**: Thích hợp cho các hàm kích hoạt sigmoid và tanh. Giúp duy trì sự ổn định của gradient trong quá trình huấn luyện và làm giảm khả năng gradient vanishing hoặc exploding gradients.