BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC SAO ĐỎ**

\*\*\*\*\*

**BÁO CÁO BÀI TẬP LỚN**

**TÊN ĐỀ TÀI: THIẾT KẾ MẠNG UNET PHÁT HIỆN ĐIỂM BẤT THƯỜNG CỦA NÃO TRÊN ẢNH MRI**

**Học phần: Lập trình Python**

**Sinh viên thực hiện: Trương Văn Tuấn**

**Lớp: DK10 – CNTT**

**Khoa: Công nghệ thông tin**

**Năm 2022**

**MỤC LỤC**

[**MỞ ĐẦU**](#_Toc73958105)

[**CHƯƠNG I. CƠ SỞ LÝ THUYẾT** 1](#_Toc73958115)

[**1.1.Tổng quang về mạng Neural** 1](#_Toc73958116)

[1.1.1.Khái niệm 1](#_Toc73958117)

[1.1.2.Các thành phần cơ bản 2](#_Toc73958118)

[1.1.3.Kiến trúc mạng 3](#_Toc73958118)

[1.1.3.1..Phân loại mạng theo kiểu kết nối các neuron 3](#_Toc73958118)

[1.1.3.2..Phân loại mạng theo số lớp neuron 4](#_Toc73958118)

[1.1.4.Huấn luyện mạng 5](#_Toc73958118)

[1.1.5.Mang noron tích chập 6](#_Toc73958118)

[**1.2.Mạng U-NET** 12](#_Toc73958116)

[1.2.1.Giới thiệu về mạng U-NET 12](#_Toc73958117)

[1.2.2.Ý tưởng đằng sau U-NET 13](#_Toc73958117)

[1.2.3.Kiến trúc U-NET 13](#_Toc73958117)

[1.2.4.Tính toán loss. IoU và DSC 14](#_Toc73958117)

[**CHƯƠNG 2.THIẾT KẾ CHƯƠNG TRÌNH** 17](#_Toc73958130)

[**2.1. Quy trình** 17](#_Toc73958131)

[**2.2. Cơ sở dữ liệu** 17](#_Toc73958132)

[**2.3.Thiết kế chương trình** 18](#_Toc73958133)

[2.3.1.Chuẩn bị dữ liệu 18](#_Toc73958133)

[2.3.2.Trực quan hóa dữ liệu 19](#_Toc73958133)

[2.3.3.Định cấu hình trình tạo dữ liệu 19](#_Toc73958133)

[2.3.4.Thiết kế mô hình U-NET 20](#_Toc73958133)

[2.3.5.Đào tạo mô hình 22](#_Toc73958133)

[2.3.6.Xây dựng giao diện 22](#_Toc73958133)

[2.3.7.Đánh giá mô hình đào tạo 23](#_Toc73958133)

[**2.4. Kết luận**](#_Toc73958151) **25**

**2.5.Tài liệu tham khảo 26**

DANH MỤC HÌNH ẢNH

[Hình 1.Cấu trúc của tế bào neuron sinh học 1](#_Toc73958856)

[Hình 2. Mô hình của một neuron nhân tạo được gán nhãn 2](#_Toc73958857)

[Hình 3.Ví dụ mạng neuron có 1 lớp ẩn 3](#_Toc73958858)

[Hình 4.. Mạng neural truyền thẳng nhiều lớp 4](#_Toc73958859)

[Hình 5. Mạng hồi quy 4](#_Toc73958860)

[Hình 6. Phân loại mạng neural 4](#_Toc73958860)

[Hình 7. Kiến trúc truyền thống của một mạng CNN 6](#_Toc73958860)

[Hình 8. Ví dụ mô hình CNN 6](#_Toc73958860)

[Hình 9. Minh họa Convolution 7](#_Toc73958860)

[Hình 10. Minh họa tầng kết nối đầy đủ 7](#_Toc73958860)

[Hình 11.Minh họa mạng CNN 7](#_Toc73958861)

[Hình 12.Minh họa các bộ lọc có kích thước khác nhau 8](#_Toc73958861)

[Hình 13.Minh họa độ trượt S 8](#_Toc73958861)

[Hình 14.Minh họa tính tương thích của tham số 10](#_Toc73958861)

[Hình 15.Minh họa tính trường thụ cảm tầng 11](#_Toc73958861)

[Hình 16.Minh họa cấu trúc U-NET 13](#_Toc73958861)

[Hình 17.Ví dụ phát hiện biển báo dừng 15](#_Toc73958861)

[Hình 18.Công thức tính IoU 15](#_Toc73958861)

[Hình 19.Công thức tính DSC 16](#_Toc73958861)

[Hình 20.Bộ dữ liệu hình ảnh MRI não 17](#_Toc73958861)

[Hình 21. Biểu đồ đánh giá quá trình đào tạo mô hình 24](#_Toc73958861)

[Hình 22.Kết quả dự đoán trên tập dữ liệu 25](#_Toc73958861)

[Hình 23.Giao diện cuối cùng 25](#_Toc73958861)

**MỞ ĐẦU**

Các ứng dụng của mô hình học sâu và thị giác máy tính trong thời kỳ hiện đại đang phát triển nhảy vọt. Thị giác máy tính là một trong những lĩnh vực trí tuệ nhân tạo, nơi chúng tôi đào tạo các mô hình của mình để diễn giải các hình ảnh trực quan trong đời thực. Với sự trợ giúp của các kiến ​​trúc học sâu như U-Net và CANet, có thể đạt được kết quả chất lượng cao trên bộ dữ liệu thị giác máy tính để thực hiện các tác vụ phức tạp. Mặc dù thị giác máy tính là một lĩnh vực vô cùng phức tạp với rất nhiều thứ để cung cấp và rất nhiều loại vấn đề khác nhau, độc đáo cần giải quyết, nhưng trọng tâm của chúng tôi trong vài bài viết tiếp theo sẽ là về hai kiến ​​trúc, đó là U-Net và CANet, được thiết kế để giải quyết nhiệm vụ của phân đoạn ảnh.

Nhiệm vụ trong phân đoạn hình ảnh là lấy một hình ảnh và chia nó thành nhiều đoạn nhỏ hơn. Những phân đoạn này hoặc nhiều phân đoạn được tạo ra này sẽ giúp tính toán các tác vụ phân đoạn hình ảnh. Đối với các tác vụ phân đoạn hình ảnh, một yêu cầu thiết yếu khác là sử dụng mặt nạ. Với sự trợ giúp của mặt nạ, về cơ bản là một hình ảnh nhị phân bao gồm các giá trị 0 hoặc khác 0, chúng ta có thể thu được kết quả mong muốn cần thiết cho nhiệm vụ phân đoạn. Một khi chúng tôi mô tả các thành phần thiết yếu nhất của hình ảnh thu được trong quá trình phân đoạn hình ảnh với sự trợ giúp của hình ảnh và mặt nạ tương ứng của chúng, chúng tôi có thể đạt được vô số nhiệm vụ trong tương lai với chúng.

Một số ứng dụng quan trọng nhất của phân đoạn hình ảnh bao gồm thị giác máy, phát hiện đối tượng, phân đoạn hình ảnh y tế, thị giác máy, nhận dạng khuôn mặt, v.v.. TensorFlow và Keras với các khung học sâu nó sẽ sử dụng để xây dựng kiến ​​trúc U-Net.

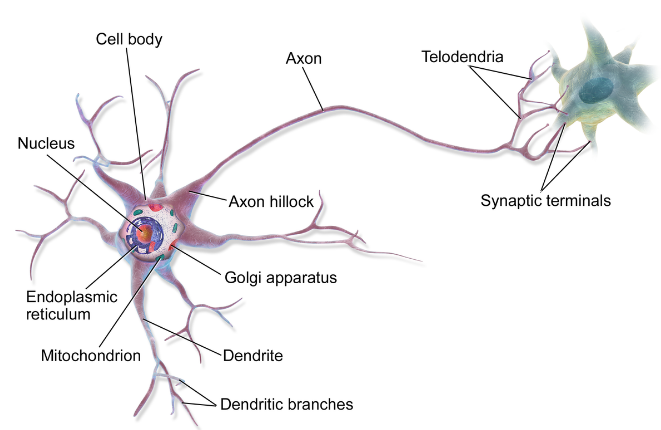
CHƯƠNG I. CƠ SỞ LÝ THUYẾT

1. **Tổng quan về mạng Neural**

### **1.3.1. Khái niệm**

Mạng neural (Neural network) hay một số tác giả khác dịch là mạng nơ-ron hoặc mạng nơron. Khi nói một nơ-ron (hoặc nơron) đơn lẻ, từ tiếng anh là neuron. Vậy nên trong giáo trình này, khi nói về một mạng nơ-ron, tác giả sẽ sử dụng thuật ngữ là mạng neural, khi nói về một nơ-ron đơn lẻ, thuật ngữ sử dụng là neuron.

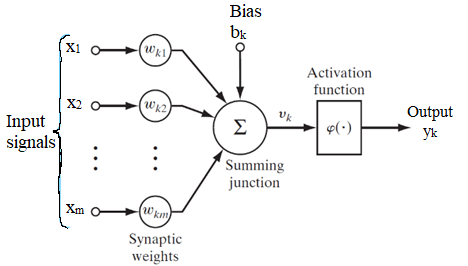
Những ý tưởng xây dựng các mô hình mạng neural nhân tạo bắt nguồn từ việc khám phá ra các cơ chế hoạt động đơn giản của mạng neural sinh học (biological neural network).



1. Cấu trúc của tế bào neuron sinh học

Các nhà nghiên cứu đã tìm cách chuyển đổi những hiểu biết về cách thức hoạt động của các tế bào thần kinh sinh học thành các mô hình mạng neural nhân tạo (Artificial Neural Network) có thể hoạt động được trên máy tính.

Mô hình của một neuron đơn lẻ, được xem như đơn vị xử lý thông tin cơ bản của một mạng neural. Các neuron này được sử dụng để xây dựng thành các mạng neural có kiến trúc phức tạp sẽ được trình bày ở các phần sau.



1. Mô hình của một neuron nhân tạo được gán nhãn k

***Các đặc trưng cơ bản của một mạng neural:***

* Một tập các đơn vị xử lý (các neuron nhân tạo).
* Trạng thái kích hoạt hay đầu ra của đơn vị xử lý.
* Liên kết giữa các đơn vị. Xét tổng quát, mỗi liên kết được định nghĩa bởi một trọng số wjk cho ta biết hiệu ứng của đơn vị j có trên đơn vị k.
* Mỗi luật lan truyền quyết định cách tính tín hiệu của từng đơn vị từ đầu vào của nó.
* Một hàm kích hoạt hay hàm chuyển (activation function, transfer function) xác định mức độ kích hoạt khác dựa trên mức độ kích hoạt hiện tại.
* Một đơn vị điều chỉnh (độ lệch) (bias, offset) của mỗi đơn vị.
* Phương pháp thu thập thông tin (luật học – learning rule).
* Môi trường hệ thống có thể hoạt động.

### **1.3.2. Các thành phần cơ bản**

Mỗi neuron (nút) là một đơn vị xử lý thông tin của mạng neural, là yếu tố cơ bản để cấu tạo thành mạng neural

Một mạng neural gồm 3 thành phần cơ bản sau:

* Đơn vị xử lý

Còn được gọi là một neuron hay một nút (node), thực hiện một công việc rất đơn giản: Nhận tín hiệu vào từ các đơn vị phía trước hay một nguồn bên ngoài và sử dụng chúng để tính tín hiệu ra sẽ được lan truyền sang các đơn vị khác.

* Hàm kết hợp

Mỗi một đơn vị trong một mạng kết hợp các giá trị đưa vào nó thông qua các liên kết với các đơn vị khác, sinh ra một giá trị gọi là net input. Hàm thực hiện nhiệm vụ này gọi là hàm kết hợp (combination function), được định nghĩa bởi một luật lan truyền   
cụ thể.

* Hàm kích hoạt

Phần lớn các đơn vị trong mạng neural chuyển net input bằng cách sử dụng một hàm vô hướng gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị (unit’s activation). Loại trừ khả năng đơn vị đó thuộc lớp ra, giá trị kích hoạt được đưa vào một hay nhiều đơn vị khác. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp (squashing).

### **1.3.3. Kiến trúc mạng**

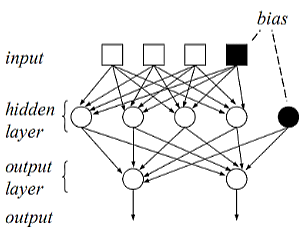
Kiến trúc của một mạng neural được xác định bởi:

* Số lượng các tín hiệu đầu vào và đầu ra.
* Số lượng các lớp
* Số lượng các neuron trong mỗi lớp
* Số lượng các trọng số (các liên kết) đối với mỗi neuron.
* Cách thức các neuron (trong một lớp, giữa các lớp) liên kết với nhau.
* Những neuron nào nhận các tín hiệu điều chỉnh lỗi.

Một mạng neural cần phải có:

* Một lớp đầu vào (input layer)
* Một lớp đầu ra (output layer)
* Không, một, hoặc nhiều lớp ẩn (hidden layer(s))

Ví dụ:



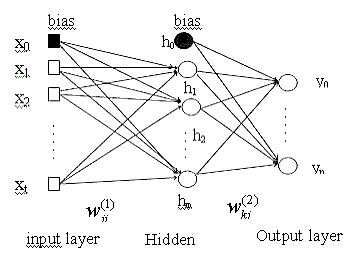
1. Ví dụ mạng neural có 1 lớp ẩn

Ví dụ về mạng neural có một lớp ẩn như hình 1.15 mạng neural có:

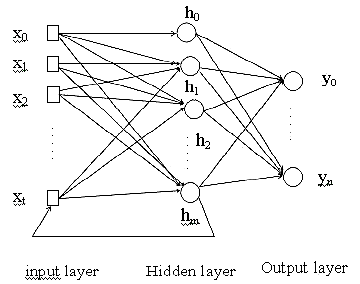
* Đầu vào: 3 tín hiệu
* Đầu ra: 2 giá trị
* Tổng cộng có 6 neuron: 4 neuron ở lớp ẩn, 2 neuron ở đầu ra.

*1.3.3.1. Phân loại mạng theo kiểu kết nối các neuron*

Dựa theo kiểu kết nối, các mạng neural truyền thẳng (feedforward Neural Network) và mạng hồi quy (recurrent Neural Network). Trong mạng truyền thẳng, các kết nối đi theo một hướng nhất định, không tạo thành chu trình. Ngược lại, các mạng hồi quy cho phép các kết nối neuron tạo thành chu trình, với đỉnh là các neuron và cung là các kết nối giữa chúng. Các neuron nhận tín hiệu vào gọi là neuron vào, các neuron đưa thông tin ra gọi là neuron ra, các neuron còn lại gọi là neuron ẩn.



1. Mạng neural truyền thẳng nhiều lớp h

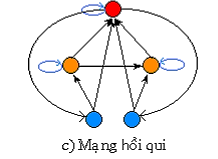
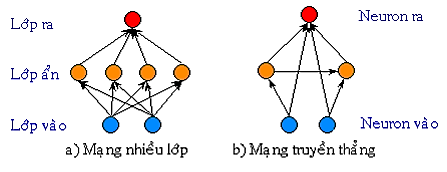


1. Mạng hồi quy i

*1.3.3.2. Phân loại theo số lớp neuron*

Các neuron trong mạng có thể được tổ chức thành các lớp theo nguyên tắc các neuron ở lớp này, chỉ được nối với các neuron ở lớp khác, không cho phép kết nối giữa các neuron trên cùng lớp, hoặc từ neuron lớp dưới lên neuron lớp trên, cũng không cho phép kết nối nhảy qua 1 lớp. Lớp nhận tín hiệu vào gọi là lớp vào, lớp đưa thông tin ra gọi là lớp ra, các lớp ở giữa gọi là lớp ẩn.

Thông thường lớp vào không tham gia quá trình tính toán của mạng neural nên khi tính số lớp người ta không kể lớp vào.



1. Phân loại mạng neural

### **1.3.4. Huấn luyện mạng**

Chức năng của một mạng nơron được quyết định bởi các nhân tố như: hình trạng mạng (số lớp, số đơn vị trên mỗi tầng, và cách mà các lớp được liên kết với nhau) và các trọng số của các liên kết bên trong mạng.

Hình trạng của mạng thường là cố định, và các trọng số được quyết định bởi một thuật toán huấn luyện (training algorithm). Tiến trình điều chỉnh các trọng số để mạng “nhận biết” được quan hệ giữa đầu vào và đích mong muốn được gọi là học (learning) hay huấn luyện (training).

Rất nhiều thuật toán học đã được phát minh để tìm ra tập trọng số tối ưu làm giải pháp cho các bài toán. Các thuật toán đó có thể chia làm ba nhóm chính:

* Học có giám sát (Supervised learning).

Là giải thuật điều chỉnh các trọng số kết nối dựa vào sự khác biệt giữa ngõ ra thực tế của mạng (actual network output) và ngõ ra mong muốn (target hoặc desired network output) ứng với một tập tín hiệu vào. Học giám sát đòi hỏi phải có một teacher hay supervisor cung cấp ngõ ra mong muốn, vì thế người ta gọi là học có thầy hay học có giám sát.

* Học không giám sát (Unsupervised Learning).

Trong phương pháp học không giám sát, không đòi hỏi tập ngõ ra mong muốn, vì thế người ta gọi là học không thầy hay học không giám sát. Trong quá trình huấn luyện, chỉ có tập dữ liệu vào được đưa vào mạng, mạng tự điều chỉnh theo nguyên tắc gộp các mẫu dữ liệu vào có đặc điểm tương tự thành từng nhóm.

Mạng neural không được giám sát được huấn luyện bằng cách để mạng neural liên tục tự điều chỉnh theo các đầu vào mới. Chúng được sử dụng để rút ra các suy luận từ các tập dữ liệu bao gồm dữ liệu đầu vào mà không có các phản hồi được gắn nhãn.

* Học tăng cường (reinforcement learning).

Học tăng cường là một giải thuật đặc biệt của học có giám sát. Thay vì phải do thầy cung cấp ngõ ra mong muốn thì giải thuật này sẽ nhờ một “chuyên gia” (critic) ước lượng ngõ ra tốt nhất ứng với một ngõ vào cho trước. Tiêu biểu cho học tăng cường là giải thuật di truyền (Genetic Algorithm - GA)

Học tăng cường là một loại kỹ thuật học máy trong đó tác nhân máy tính học cách thực hiện một tác vụ thông qua các tương tác thử và lỗi lặp đi lặp lại với một môi trường động. Cách tiếp cận học tập này cho phép người đại diện đưa ra một loạt quyết định nhằm tối đa hóa số liệu thưởng cho nhiệm vụ mà không cần sự can thiệp của con người và không được lập trình rõ ràng để đạt được nhiệm vụ.

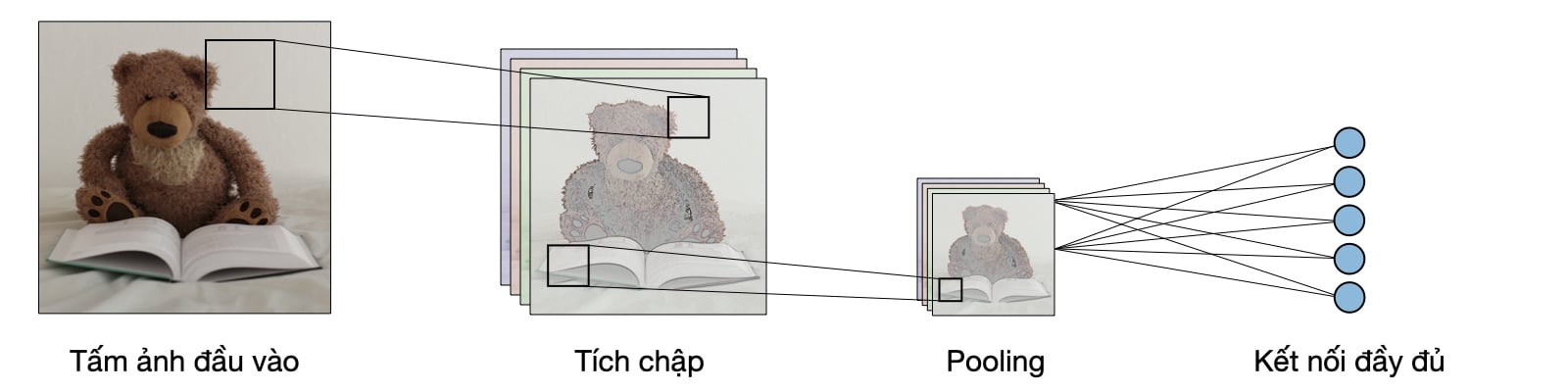
Các chương trình AI được huấn luyện với học tập củng cố đánh bại người chơi con người trong các trò chơi trên bàn cờ như cờ vây và cờ vua, cũng như trò chơi điện tử. Mặc dù học tăng cường không phải là một khái niệm mới, nhưng tiến bộ gần đây trong học sâu và sức mạnh tính toán đã giúp nó có thể đạt được một số kết quả đáng chú ý trong lĩnh vực trí tuệ nhân tạo.

### **1.3.5. Mạng neural tích chập**

*a, Cách thức hoạt động của CNN*

Một mạng neural tích chập có thể có hàng chục hoặc hàng trăm lớp mà mỗi lớp học để phát hiện các đặc điểm khác nhau của hình ảnh.

CNN bao gồm một lớp đầu vào, một lớp đầu ra và nhiều lớp ẩn ở giữa. Các lớp này thực hiện các hoạt động làm thay đổi dữ liệu với mục đích tìm hiểu các đặc trưng cụ thể cho dữ liệu. Một số lớp phổ biến nhất là: Convolution, ReLU, Pooling, ReLU.

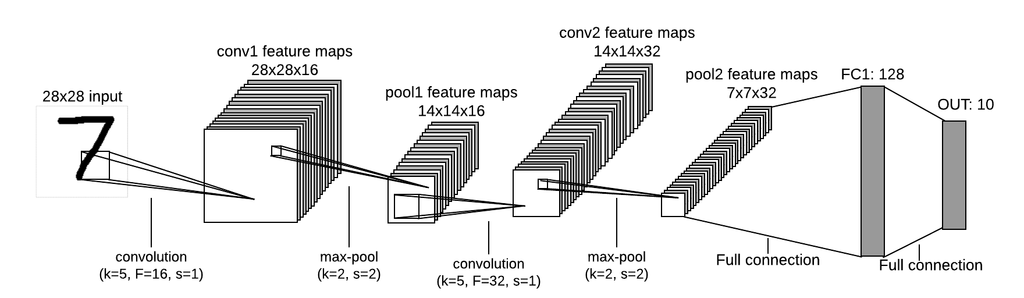


1. Kiến trúc truyền thống của một mạng CNN

*b, Mô hình tổng quát của CNN*

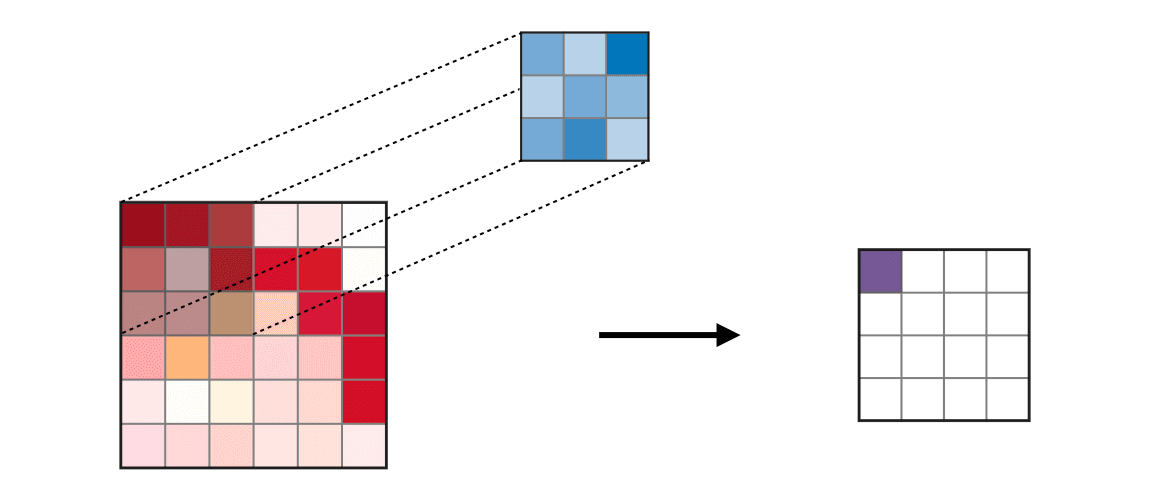
Mô hình tổng quát của một mạng CNN:

Input image 🡪 Convolutional layer (Conv) + Pooling layer (Pool) 🡪 Fully connected layer (FC) 🡪 Output.



1. Ví dụ mô hình CNN

Convolution: Đặt các hình ảnh đầu vào thông qua một tập hợp các bộ lọc tích chập, mỗi bộ lọc trong số đó sẽ kích hoạt một số đặc trưng nhất định từ hình ảnh.

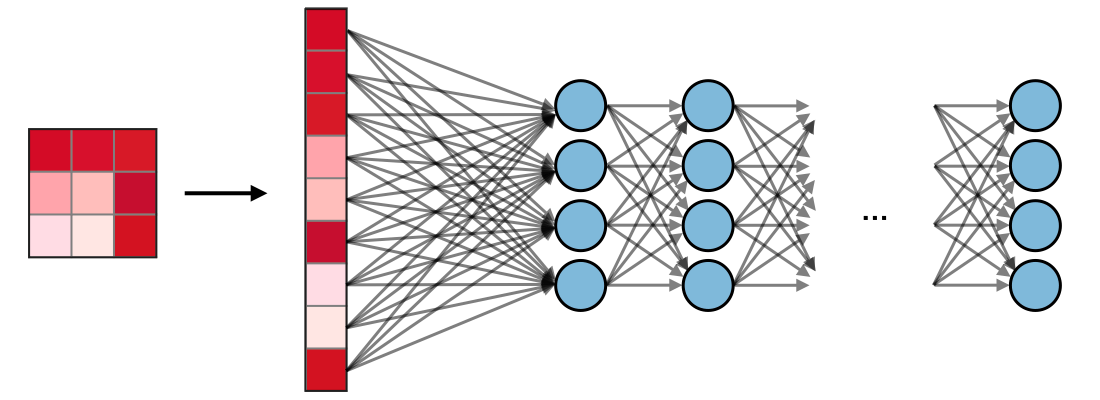


1. Minh họa Convolution

ReLU: Cho phép huấn luyện nhanh hơn và hiệu quả hơn bằng cách ánh xạ các giá trị âm về 0 và duy trì các giá trị dương. Điều này đôi khi được gọi là kích hoạt , bởi vì chỉ các đặc trưng đã được kích hoạt mới được chuyển sang lớp tiếp theo.

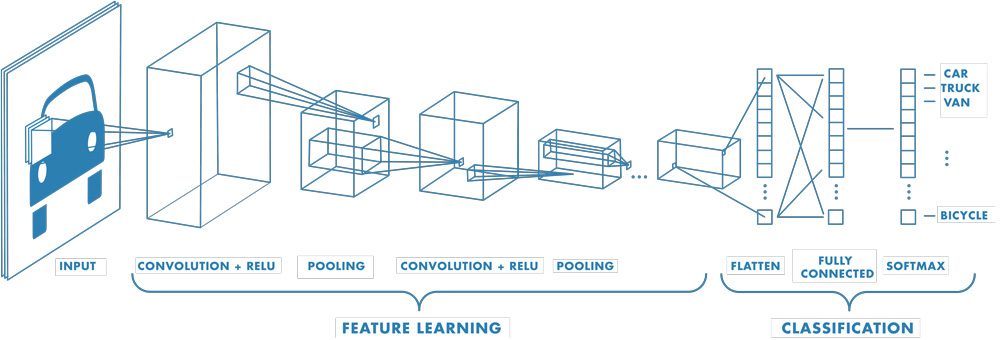
Pooling: Đơn giản hóa đầu ra bằng cách thực hiện lấy mẫu xuống phi tuyến, giảm số lượng các tham số mà mạng cần tìm hiểu.

đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.



1. Minh họa tầng kết nối đầy đủ

Các thao tác này được lặp lại trên hàng chục hoặc hàng trăm lớp, với mỗi lớp học để xác định các đặc trưng khác nhau.



1. Minh họa mạng CNN

Hình 1.24 là ví dụ về một mạng có nhiều lớp chập. Các bộ lọc được áp dụng cho mỗi hình ảnh huấn luyện ở các độ phân giải khác nhau và đầu ra của mỗi hình ảnh biến đổi được sử dụng làm đầu vào cho lớp tiếp theo.

**Trọng số và độ lệch chia sẻ**

Giống như một mạng neural truyền thống, CNN có trọng số và độ lệch. Mô hình học các giá trị này trong quá trình huấn luyện và nó liên tục cập nhật chúng với mỗi ví dụ huấn luyện mới. Tuy nhiên, trong trường hợp của CNN, trọng số và độ lệch là giống nhau đối với tất cả các neuron ẩn trong một lớp nhất định.

Điều này có nghĩa là tất cả các neuron ẩn đang phát hiện cùng một đặc điểm, chẳng hạn như một cạnh hoặc một đốm màu, ở các vùng khác nhau của hình ảnh. Điều này làm cho mạng có khả năng dịch các đối tượng trong một hình ảnh. Ví dụ: một mạng được huấn luyện để nhận dạng ô tô sẽ có thể làm như vậy ở bất kỳ nơi nào có ô tô trong hình ảnh.

**Các lớp phân loại**

Sau khi tìm hiểu các đặc trưng trong nhiều lớp, kiến ​​trúc của CNN chuyển sang phân loại.

Lớp tiếp theo đến lớp cuối cùng là lớp được kết nối đầy đủ tạo ra một vector có K chiều trong đó K là số lớp mà mạng có thể dự đoán. Vector này chứa các xác suất cho mỗi lớp của bất kỳ hình ảnh nào đang được phân loại.

Lớp cuối cùng của kiến ​​trúc CNN sử dụng một lớp phân loại như softmax để cung cấp đầu ra phân loại.

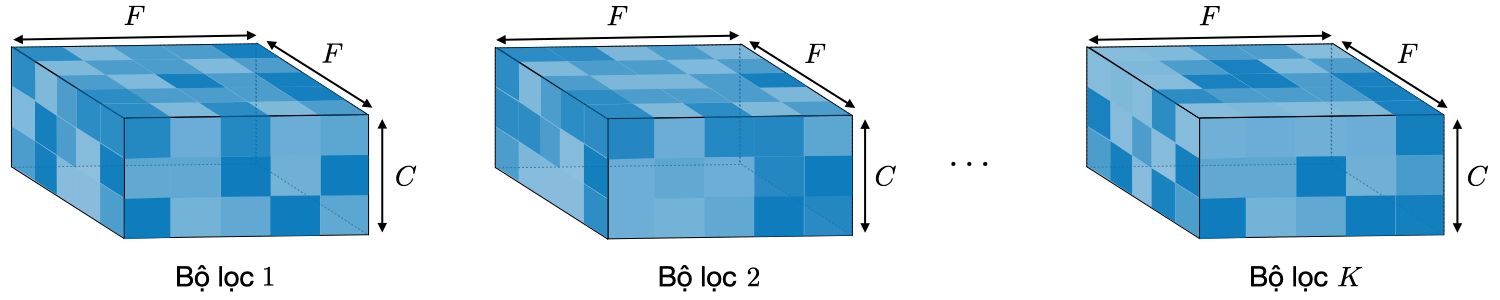
*c, Điều chỉnh các siêu tham số*

* + *Các siêu tham số của bộ lọc*

Tầng tích chập chứa các bộ lọc mà rất quan trọng cho ta khi biết ý nghĩa đằng sau các siêu tham số của chúng.

**- Các chiều của một bộ lọc**

Một bộ lọc kích thước FxF áp dụng lên đầu vào chứa C kênh (channels) thì có kích thước tổng thể là F×*F*×*C* thực hiện phép tích chập trên đầu vào kích thước  *I*×*I*×*C* và cho ra một feature map (hay còn gọi là activation map) có kích thước *O*×*O*×1.

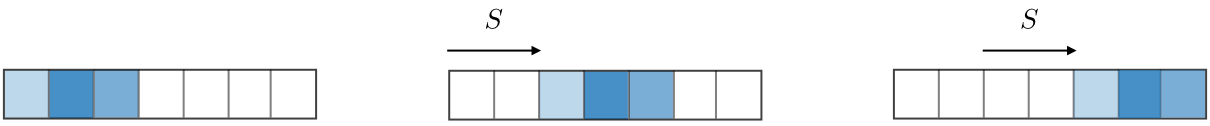


1. Minh họa các bộ lọc có kích thước khác nhau

*Lưu ý: Việc áp dụng K bộ lọc có kích thước F×F cho ra* một *feature map có kích thước O O×O×K.*

**- Stride**

Đối với phép tích chập hoặc phép pooling, độ trượt *S* ký hiệu số pixel mà cửa sổ sẽ di chuyển sau mỗi lần thực hiện phép tính.



1. Minh họa độ trượt S

**- Zero-padding**

Zero-padding là tên gọi của quá trình thêm *P* số không vào các biên của đầu vào. Giá trị này có thể được lựa chọn thủ công hoặc một cách tự động bằng một trong ba những phương pháp mô tả bên dưới:

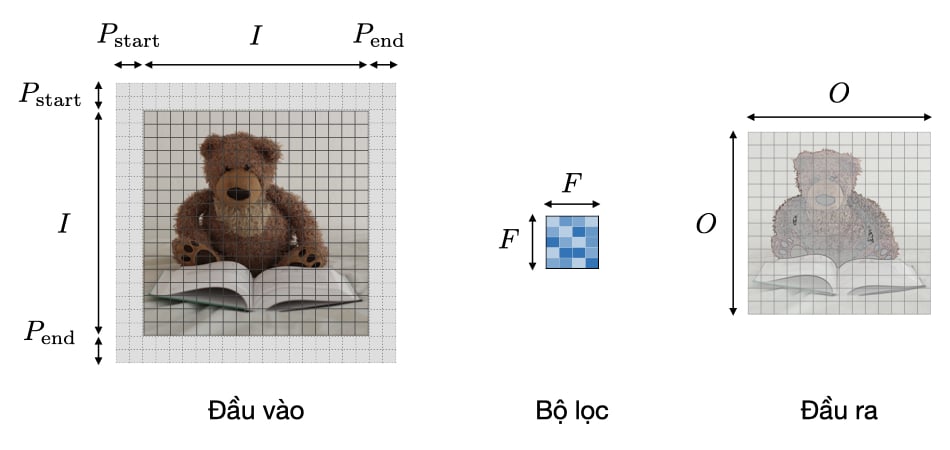
1. So sánh các phương pháp Zero-padding

| **Phương pháp** | **Valid** | **Same** | **Full** |
| --- | --- | --- | --- |
| **Giá trị** | P = 0 |  | Pend = F-1 |
| **Minh họa** | Padding valid | Padding same | Padding full |
| **Mục đích** | • Không sử dụng padding • Bỏ phép tích chập cuối nếu số chiều không khớp | • Sử dụng padding để làm cho feature map có kích thước  • Kích thước đầu ra thuận lợi về mặt toán học • Còn được gọi là 'half' padding | • Padding tối đa sao cho các phép tích chập có thể được sử dụng tại các rìa của đầu vào  • Bộ lọc 'thấy' được đầu vào từ đầu đến cuối |

* + *Điều chỉnh siêu tham số*

**- Tính tương thích của tham số trong tầng tích chập**

Bằng cách ký hiệu I là độ dài kích thước đầu vào, F là độ dài của bộ lọc, P là số lượng zero padding, S là độ trượt, ta có thể tính được độ dài *O* của feature map theo một chiều bằng công thức:



1. Minh họa tính tương thích của tham số

*Lưu ý:* Trong *một số trường hợp, Pstart=Pend≜P, ta có thể thay thế  Pstart+Pend bằng $2P trong công thức trên.*

**- Hiểu về độ phức tạp của mô hình**

Để đánh giá độ phức tạp của một mô hình, cách hữu hiệu là xác định số tham số mà mô hình đó sẽ có. Trong một tầng của mạng neural tích chập, nó sẽ được tính toán như sau:

1. So sánh độ phức tạp của mô hình

|  | **CONV** | **POOL** | **FC** |
| --- | --- | --- | --- |
| Minh họa | https://stanford.edu/~shervine/teaching/cs-230/illustrations/table-conv.png?79f617dcf0ac221ddfaf21694f6e08ad | https://stanford.edu/~shervine/teaching/cs-230/illustrations/table-pool.png?e8528df02bafea0840916a83482e42e9 | https://stanford.edu/~shervine/teaching/cs-230/illustrations/table-fc.png?0074d2fdaa632e724022c13e94a49a22 |
| Kích thước đầu vào | I×*I*×*C* | *I*×*I*×*C* | *N*in​ |
| Kích thước đầu ra | *O*×*O*×*K* | *O*×*O*×*C* | *N*out​ |
| Số lượng tham số | (*F*×*F*×*C*+1)⋅*K* | 0 | (Nin +1)x Nout |
| Lưu ý | • Một tham số bias với mỗi bộ lọc  • Trong đa số trường hợp, S < F  • Một lựa chọn phổ biến cho K là 2C | • Phép pooling được áp dụng lên từng kênh (channel-wise)  • Trong đa số trường hợp, S = F | • Đầu vào được làm phẳng  • Mỗi neuron có một tham số bias  • Số neuron trong một tầng FC phụ thuộc vào ràng buộc kết cấu |

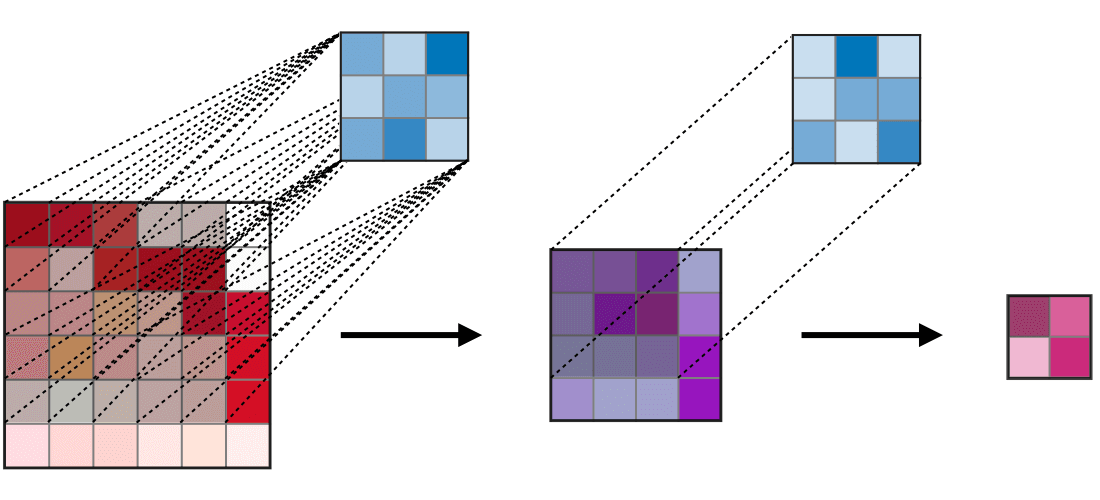
**- Trường thụ cảm**

Trường thụ cảm (receptive field) tại tầng k là vùng được ký hiệu Rk×Rk của đầu vào mà những pixel của activation map thứ k có thể "nhìn thấy".

Gọi Fj là kích thước bộ lọc của tầng j và *Si* là giá trị độ trượt của tầng i và để thuận tiện, ta mặc định *S*0=1, trường thụ cảm của tầng k được tính toán bằng công thức:

(2.1)

*Trong ví dụ hình 1.27, ta có  F1=F2=3 và S1=S2=1, nên cho ra được R2=1+2⋅1+2⋅1=5.*



1. Minh họa tính trường thụ cảm tầng k

*d, Các hàm kích hoạt thường gặp*

* + *Rectified Linear Unit*

Tầng rectified linear unit (ReLU) là một hàm kích hoạt g*g* được sử dụng trên tất cả các thành phần. Mục đích của nó là tăng tính phi tuyến tính cho mạng. Những biến thể khác của ReLU được tổng hợp ở bảng dưới:

1. Biến thể của ReLU

| **ReLU** | **Leaky ReLU** | **ELU** |
| --- | --- | --- |
| *g*(*z*)=max(0,*z*) | *g*(*z*)=max(*ϵz*,*z*) với *ϵ*≪1 | *g*(*z*)=max(*α*(*ez*−1),*z*) với *α*≪1 |
| ReLU | Leaky ReLU | ELU |
| • Độ phức tạp phi tuyến tính có thể thông dịch được về mặt sinh học | • Gán vấn đề ReLU chết cho những giá trị âm | • Khả vi tại mọi nơi |

*e, Softmax*

Bước softmax có thể được coi là một hàm logistic tổng quát lấy đầu vào là một vector chứa các giá trị *x*∈R*n*và cho ra là một vector gồm các xác suất *p*∈R*n* thông qua một hàm softmax ở cuối kiến trúc

Huấn luyện mạng phức tạp từ đầu hay sử dụng các mạng được huấn luyện trước để nhanh chóng học các nhiệm vụ mới.

Tạo mạng sâu mới cho các nhiệm vụ phân loại và hồi quy hình ảnh bằng cách xác định kiến trúc mạng và huấn luyện mạng từ đầu. Việc tinh chỉnh mạng phân loại hình ảnh được huấn luyện trước bằng cách học chuyển tiếp thường nhanh hơn và dễ dàng hơn nhiều so với việc huấn luyện từ đầu. Sử dụng mạng sâu được huấn luyện trước cho phép nhanh chóng tìm hiểu các tác vụ mới mà không cần xác định và huấn luyện mạng mới, có hàng triệu hình ảnh hoặc có GPU mạnh mẽ.

Sau khi xác định kiến trúc mạng, ta phải xác định các tham số huấn luyện bằng cách sử dụng hàm trainingOptions. Sau đó, có thể huấn luyện mạng bằng cách sử dụng trainNetwork. Sử dụng mạng được huấn luyện để dự đoán nhãn lớp hoặc phản hồi số.

Ta có thể huấn luyện mạng neural phức hợp trên một CPU, một GPU, nhiều CPU hoặc GPU hoặc song song trên một cụm hoặc trong đám mây. Chỉ định môi trường thực thi bằng cách sử dụng chức năng trainingOptions.

1. **Mạng UNET**
   1. **Giới thiệu về mạng unet**

Kiến trúc U-Net, được xuất bản lần đầu tiên vào năm 2015, đã là một cuộc cách mạng trong lĩnh vực học sâu. Kiến trúc đã giành chiến thắng trong Hội nghị chuyên đề quốc tế về thách thức theo dõi tế bào Y sinh (ISBI) năm 2015 ở nhiều hạng mục với tỷ lệ chênh lệch lớn. Một số công trình của họ bao gồm phân đoạn cấu trúc tế bào thần kinh trong các ngăn xếp hiển vi điện tử và hình ảnh hiển vi ánh sáng truyền qua.

Với kiến ​​trúc U-Net này, việc phân đoạn hình ảnh có kích thước 512X512 có thể được tính toán bằng GPU hiện đại trong một khoảng thời gian nhỏ. Đã có nhiều biến thể và sửa đổi của kiến ​​trúc này do thành công phi thường của nó. Một số trong số chúng bao gồm LadderNet, U-Net với sự chú ý, U-Net tích hợp lặp lại và dư (R2-UNet) và U-Net với các khối còn lại hoặc các khối có kết nối dày đặc.

Mặc dù U-Net là một thành tựu đáng kể trong lĩnh vực học sâu, nhưng điều cần thiết không kém là phải hiểu các phương pháp trước đây đã được sử dụng để giải quyết loại nhiệm vụ tương tự như vậy. Một trong những ví dụ chính cuối cùng là phương pháp cửa sổ trượt, đã chiến thắng thách thức phân đoạn EM tại ISBI vào năm 2012 với tỷ suất lợi nhuận lớn. Cách tiếp cận cửa sổ trượt có thể tạo ra một loạt các bản vá mẫu ngoài tập dữ liệu đào tạo ban đầu.

Kết quả này là do nó đã sử dụng phương pháp thiết lập mạng kiến ​​trúc cửa sổ trượt bằng cách tạo nhãn lớp của mỗi pixel thành các đơn vị riêng biệt bằng cách cung cấp một vùng cục bộ (bản vá) xung quanh pixel đó. Một thành tựu khác của kiến ​​trúc này là nó có thể bản địa hóa khá dễ dàng trên bất kỳ tập dữ liệu đào tạo nào cho các nhiệm vụ tương ứng.

Tuy nhiên, cách tiếp cận cửa sổ trượt có hai nhược điểm chính mà kiến ​​trúc U-Net đã khắc phục. Vì mỗi pixel được xem xét riêng biệt, nên các bản vá kết quả chúng tôi chồng chéo lên nhau rất nhiều. Do đó, rất nhiều dự phòng tổng thể đã được tạo ra. Một hạn chế khác là quy trình đào tạo tổng thể khá chậm và tiêu tốn nhiều thời gian và nguồn lực. Tính khả thi của hoạt động của mạng bị nghi ngờ do những lý do sau.

U-Net là một kiến ​​trúc thanh lịch giải quyết hầu hết các vấn đề xảy ra. Nó sử dụng khái niệm mạng tích tụ đầy đủ cho cách tiếp cận này. Mục đích của U-Net là nắm bắt cả các đặc điểm của ngữ cảnh cũng như bản địa hóa. Quá trình này được hoàn thành thành công bởi kiểu kiến ​​trúc được xây dựng. Ý tưởng chính của việc triển khai là sử dụng các lớp hợp đồng kế tiếp, được theo sau bởi các toán tử upsampling để đạt được kết quả đầu ra có độ phân giải cao hơn trên các hình ảnh đầu vào.

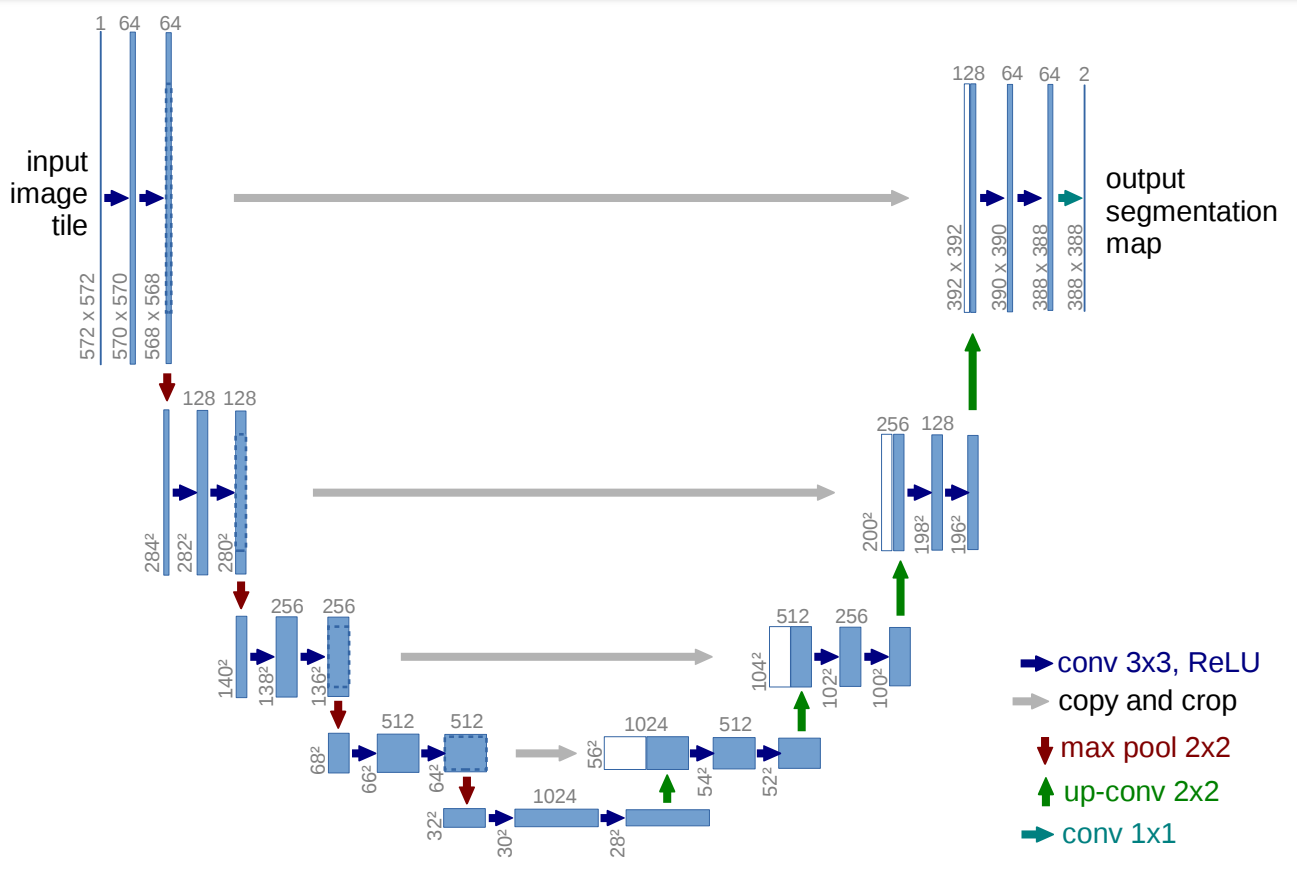
* 1. **Ý tưởng đằng sau UNET**

Ý tưởng chính đằng sau CNN là học feature mapping của hình ảnh và dùng nó để tạo ra feature mapping mức cao hơn. Điều này hoạt động tốt trong các bài toán phân loại vì hình ảnh được chuyển đổi thành một vector sử dụng thêm để phân loại. Nhưng trong phân vùng hình ảnh, chúng ta không chỉ cần chuyển đổi feature map thành một vector mà còn tái tạo lại một hình ảnh từ vector này. Đây là một tác vụ khó. Toàn bộ ý tưởng của UNet được xoay quanh vấn đề này.

Khi chuyển đổi một hình ảnh thành một vector, chúng ta đã học feature mapping của hình ảnh, vậy tại sao không sử dụng cùng một mapping để chuyển đổi nó thành hình ảnh. Đây là công thức đằng sau UNet. Sử dụng các feature map tương tự được sử dụng khi tạo vector để mở rộng vector thành hình ảnh được phân vùng. Điều này sẽ bảo vệ tính toàn vẹn cấu trúc của hình ảnh, làm giảm sự biến dạng rất nhiều. Hãy tìm hiểu về kiến trúc một cách ngắn gọn hơn.

* 1. **Kiến trúc UNET**

Hình dạng của kiến ​​trúc được hình thành có dạng chữ 'U' và do đó có tên như sau. Chỉ cần nhìn vào cấu trúc và vô số yếu tố liên quan đến quá trình xây dựng kiến ​​trúc này, chúng ta có thể hiểu rằng mạng được xây dựng là một mạng phức hợp đầy đủ. Họ đã không sử dụng bất kỳ lớp nào khác như dày đặc hoặc làm phẳng hoặc các lớp tương tự khác. Biểu diễn trực quan cho thấy một đường dẫn hợp đồng ban đầu theo sau là một đường dẫn mở rộng.



1. Cấu trúc UNET

Kiến trúc cho thấy rằng một hình ảnh đầu vào được chuyển qua mô hình và sau đó nó được theo sau bởi một vài lớp phức hợp với chức năng kích hoạt ReLU. Chúng ta có thể nhận thấy rằng kích thước hình ảnh đang giảm từ 572X572 xuống 570X570 và cuối cùng là 568X568. Lý do cho việc giảm này là do họ đã sử dụng các chập không được đệm (được định nghĩa là các chập là "hợp lệ"), dẫn đến giảm kích thước tổng thể. Ngoài các khối Convolution, chúng ta cũng nhận thấy rằng chúng ta có một khối mã hóa ở phía bên trái, tiếp theo là khối bộ giải mã ở phía bên phải.

Kiến trúc này gồm 3 phần: contraction (encoder), bottleneck và expansion (decoder). Phần contraction được tạo bởi nhiều khối. Mỗi khối gồm hai lớp CNN có kernel size là 3×3, theo sau bởi một lớp 2×2 max pooling. Số lượng kernel hay feature map tăng gấp đôi sau mỗi khối, do đó kiến trúc này có thể học những đặc trưng phức tạp một cách hiệu quả. Lớp ở vị trí thấp nhất làm trung gian giữa các lớp ở phần contraction và phần expansion. Nó dùng 2 lớp 3×3 CNN, theo sau bởi 1 với 2×2 up convolution.

Nhưng trái tim của kiến trúc này nằm ở phần expansion. Tương tự như contraction, nó cũng bao gồm nhiều khối. Mỗi khối gồm hai lớp 3×3 CNN theo sau bởi một lớp upsampling. Sau mỗi khối số lượng feature map được giảm đi một nửa để đảm bảo tính đối xứng. Tuy nhiên đầu vào của mỗi khối expansion sẽ được nối thêm feature map của khối contraction tương ứng. Hành động này sẽ đảm bảo các được trưng được học trong quá trình contracting ảnh sẽ được dùng để tái tạo nó. Số lượng khối expansion bằng với số lượng khối contraction. Sau cùng, kết quả sẽ được đưa qua một lớp 3×3 CNN khác với số feature map bằng với số vùng mong muốn.

Khối mã hóa có sự giảm kích thước hình ảnh liên tục với sự trợ giúp của các lớp tổng hợp tối đa của bước 2. Chúng tôi cũng có các lớp tích tụ lặp lại với số lượng bộ lọc ngày càng tăng trong kiến ​​trúc bộ mã hóa. Khi chúng tôi tiếp cận khía cạnh bộ giải mã, chúng tôi nhận thấy số lượng bộ lọc trong các lớp phức hợp bắt đầu giảm cùng với việc lấy mẫu tăng dần ở các lớp sau lên đến đầu. Chúng tôi cũng nhận thấy rằng việc sử dụng các kết nối bỏ qua kết nối các đầu ra trước đó với các lớp trong khối bộ giải mã.

Kết nối bỏ qua này là một khái niệm quan trọng để bảo toàn sự mất mát từ các lớp trước để chúng phản ánh mạnh mẽ hơn về các giá trị tổng thể. Chúng cũng được khoa học chứng minh là tạo ra kết quả tốt hơn và dẫn đến sự hội tụ mô hình nhanh hơn. Trong khối tích chập cuối cùng, chúng ta có một vài lớp tích chập theo sau là lớp tích chập cuối cùng. Lớp này có bộ lọc 2 với chức năng thích hợp để hiển thị kết quả đầu ra. Lớp cuối cùng này có thể được thay đổi tùy theo mục đích mong muốn của dự án mà bạn đang cố gắng thực hiện.

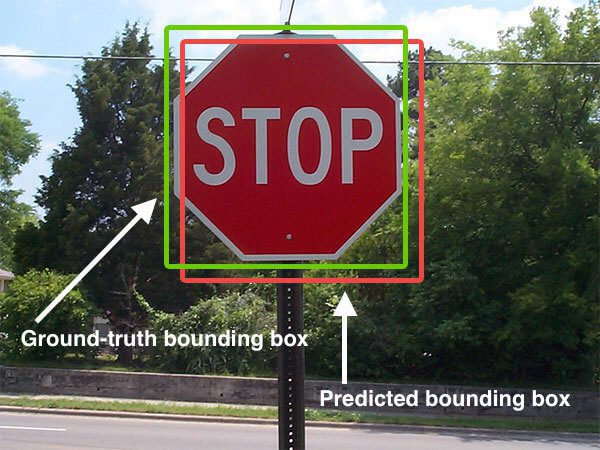
* 1. **Tính toán loss, IoU và DSC**

1. **Intersection over Union** (**IoU)**

Intersection over Union là một số liệu đánh giá được sử dụng để đo độ chính xác của bộ phát hiện đối tượng trên một tập dữ liệu cụ thể.

Intersection over Union chỉ đơn giản là một thước đo đánh giá . Bất kỳ thuật toán nào cung cấp các hộp giới hạn dự đoán dưới dạng đầu ra đều có thể được đánh giá bằng IoU.

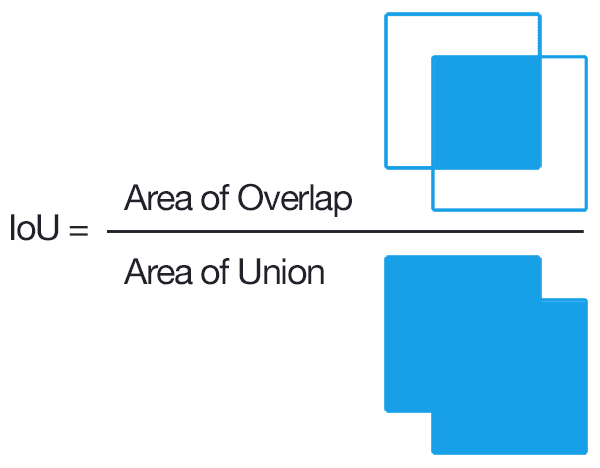
Dưới đây, một ví dụ trực quan về hộp giới hạn sự thật cơ bản so với hộp giới hạn dự đoán:



1. Ví dụ về phát hiện biển báo dừng

Hộp giới hạn dự đoán được vẽ bằng màu đỏ trong khi hộp giới hạn chân lý nền (nghĩalà được gắn nhãn bằng tay) được vẽ bằng màu xanh lục.

Do đó, Giao điểm tính toán qua Union có thể được xác định thông qua:



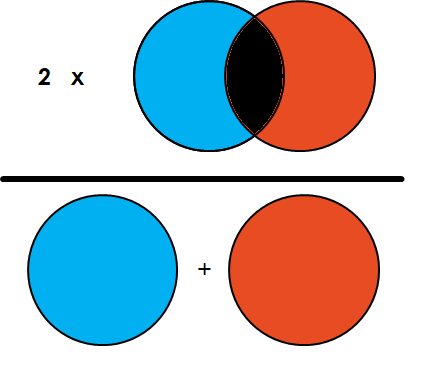
1. Công thức tình IoU

IoU là diện tích trùng lặp giữa phân đoạn được dự đoán và sự thật cơ bản chia cho diện tích hợp nhất giữa phân đoạn được dự đoán và sự thật cơ bản , như được hiển thị trên hình bên trái. Chỉ số này nằm trong khoảng từ 0–1 (0–100%) với 0 biểu thị không có chồng chéo và 1 biểu thị phân đoạn chồng chéo hoàn hảo.

Đối với phân đoạn nhị phân (hai lớp) hoặc nhiều lớp , IoU trung bình của hình ảnh được tính bằng cách lấy IoU của mỗi lớp và lấy trung bình của chúng . (Nó được triển khai hơi khác trong mã).

1. **Dice Coefficient**

Hệ số Xúc xắc rất giống với IoU. Chúng có tương quan thuận, có nghĩa là nếu một người nói mô hình A tốt hơn mô hình B trong việc phân đoạn một hình ảnh, thì người kia cũng sẽ nói như vậy. Giống như IoU, cả hai đều nằm trong khoảng từ 0 đến 1, với 1 biểu thị sự giống nhau lớn nhất giữa dự đoán và sự thật.



1. Công thức tính DSC
2. **Loss trong UNET**

Loại loss nên được dùng trong phân vùng ảnh. Nó đã được định nghĩa đơn giản trong paper: “The energy function is computed by a pixel-wise soft-max over the final feature map combined with the cross-entropy loss function“.

Unet sử dụng một cơ chế tính trọng số loss cho từng pixel sao cho trọng số cao hơn ở viền của các vùng chứa đối tượng. Cơ chế loss có trọng số này giúp cho mô hình Unet phân vùng các tế bào trong ảnh y sinh theo kiểu không liên tục sao cho các tế bào riêng lẻ có thể được xác định trong ảnh phân vùng nhị phân.

Trước hết, pixel-wise softmax được áp dụng trên hình ảnh kết quả, tiếp theo là hàm cross-entropy. Vì vậy, chúng ta đang phân loại từng pixel thành một trong số các lớp. Ý tưởng là khi phân vùng, mọi pixel đều phải nằm trong một số lớp. Vì vậy, chúng ta chỉ chuyển một bài toán phân vùng thành một vài toán phân loại đa lớp và nó cho kết quả rất tốt so với các hàm loss truyền thống.

# **CHƯƠNG 2. THIẾT KẾ CHƯƠNG TRÌNH**

## 2.1. Quy trình

Bước 1: Thu thập và gán nhãn dữ liệu

Bước 2: Trực quan hóa dữ liệu

Bước 3: Định cấu hình trình tạo dữ liệu

Bước 4: Xây dựng kiến ​​trúc U-Net:

Bước 5: Đào tạo mô hình

Bước 6: Đánh giá kết quả

## 2.2. Cơ sở dữ liệu

Bộ dữ liệu này chứa hình ảnh MRI não cùng với mặt nạ phân đoạn bất thường flair thủ công. Các hình ảnh đã được lấy từ kho lưu trữ hình ảnh ung thư (TCIA). Chúng tương ứng với 110 bệnh nhân được bao gồm trong bộ sưu tập Glioma cấp thấp hơn của ung thư (TCGA) với ít nhất là trình tự phục hồi đảo ngược (Flair) và dữ liệu bộ gen có sẵn.

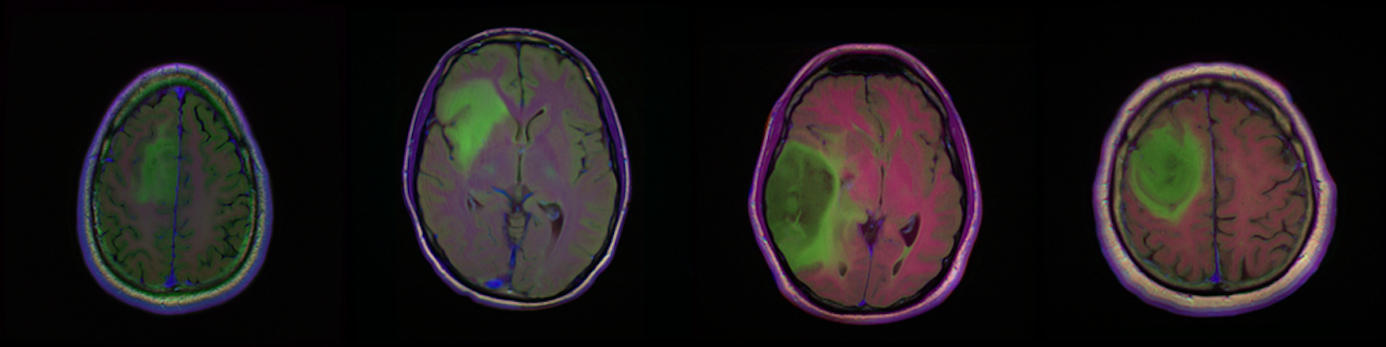
Bộ dữ liệu này chứa hình ảnh MR não cùng với mặt nạ phân đoạn bất thường FLAIR thủ công.

Các hình ảnh được lấy từ Cơ quan Lưu trữ Hình ảnh Ung thư (TCIA). Chúng tương ứng với 110 bệnh nhân được đưa vào bộ sưu tập u thần kinh đệm cấp thấp hơn của The Cancer Genome Atlas (TCGA) với ít nhất trình tự phục hồi đảo ngược giảm độc tố chất lỏng (FLAIR) và dữ liệu cụm gen có sẵn.

Các cụm gen khối u và dữ liệu bệnh nhân được cung cấp trong tệp `data.csv`. Tất cả hình ảnh được cung cấp ở định dạng `.tif` với 3 kênh trên mỗi hình ảnh.

Đối với 101 trường hợp, 3 chuỗi có sẵn, tức là trước tương phản, FLAIR, sau tương phản (theo thứ tự kênh này). Đối với 9 trường hợp, trình tự sau tương phản bị thiếu và đối với 6 trường hợp, thiếu trình tự tương phản trước.

Các trình tự bị thiếu được thay thế bằng trình tự FLAIR để làm cho tất cả hình ảnh thành 3 kênh. Mặt nạ là hình ảnh nhị phân, 1 kênh. Họ phân đoạn bất thường FLAIR có trong trình tự FLAIR (có sẵn cho mọi trường hợp).



1. *Bộ dữ liệu hình ảnh MRI não*

Bộ dữ liệu gồm hơn 2800 ảnh và mask kèm theo file csv lưu trữ thông tin dùng để gán nhãn dữ liệu tự động hoặc thủ công. Tất cả dữ liệu dùng để huấn luyện với định dạng .tif.

## 2.3. Thiết kế chương trình

1. ***Chuẩn bị tập dữ liệu***

Bộ dữ liệu được thu thập Tập dữ liệu được tổ chức thành 110 thư mục được đặt tên theo ID trường hợp chứa thông tin về tổ chức nguồn.

Mỗi thư mục chứa hình ảnh MR với quy ước đặt tên sau:

`TCGA\_ <bệnh viện- mã> \_ <bệnh nhân- điều trị> \_ <slice-number> .tif`

Các mặt nạ tương ứng có hậu tố `\_mask`.

Đánh nhãn ảnh lỗi:

def draw\_defect(file, labels, w, h):

# Lấy file id

file\_id = int(file.replace(".png", ""))

# Lấy nhãn của file

label = labels[file\_id - 1]

# Tách các thành phần trong nhãn

label = label.replace("\t", "").replace(" ", " ").replace(" ", " ").replace("\n", "")

label\_array = label.split(" ")

# Vẽ hình ellipse

major, minor, angle, x\_pos, y\_pos = float(label\_array[1]), float(label\_array[2]), float(label\_array[3]), float(

label\_array[4]), float(label\_array[5])

rr, cc = ellipse(y\_pos, x\_pos, r\_radius=minor, c\_radius=major, rotation=-angle)

# Tạo ảnh màu đen

mask\_image = np.zeros((w, h), dtype=np.uint8)

try:

# Gán các điểm thuộc hình ellipse thành 1

mask\_image[rr, cc] = 1

except:

# Nếu lỗi chỉ gán các điểm trong ảnh

rr\_n = [min(511, rr[i]) for i in rr]

cc\_n = [min(511, cc[i]) for i in cc]

mask\_image[rr\_n, cc\_n] = 1

# mask\_image = Image.fromarray(mask\_image)

# Chuyển thành ảnh

mask\_image = np.array(mask\_image, dtype=np.uint8)

mask\_image = Image.fromarray(mask\_image)

return mask\_image

1. ***Trực quan hóa dữ liệu***

Thu thập và xử lý trước dữ liệu cho dự án, bước tiếp theo sẽ là xem xét sơ lược về tập dữ liệu. Hãy để chúng tôi phân tích tập dữ liệu bằng cách hiển thị cả hình ảnh và đầu ra được phân đoạn tương ứng của nó. Đầu ra được phân đoạn với mặt nạ này thường được gọi là chú thích sự thật. Khối mã được viết như sau:

rows, cols = 3, 3

fig = plt.figure(figsize=(10, 10))  # tạo biểu đồ

for i in range(1, rows \* cols + 1):

    fig.add\_subplot(rows, cols, i)  # tạo thêm sub-plot vị trí tiếp

    img\_path = train\_files[i]  # lấy path dữ liệu

    msk\_path = mask\_files[i]

    img = cv2.imread(img\_path)  # đọc ảnh

    img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

    msk = cv2.imread(msk\_path)

    plt.imshow(img)

    plt.imshow(msk, alpha=0.4)

plt.show()

1. ***Định cấu hình trình tạo dữ liệu***
   * Thay vì sử dụng trình tạo dữ liệu để chuẩn bị dữ liệu, mô hình sẽ sử dụng hoạt động trình tự từ khung học sâu . Bên cạnh đó thì data augmentation cũng được sử dụng để tăng lượng ảnh cho training set. Sau đó, xây dựng hàm cuối cùng sẽ tạo ra các lô dữ liệu.

def train\_generator(data\_frame, batch\_size, aug\_dict,

                    image\_color\_mode="rgb",

                    mask\_color\_mode="grayscale",

                    image\_save\_prefix="image",

                    mask\_save\_prefix="mask",

                    save\_to\_dir=None,

                    target\_size=(256, 256),

                    seed=1):

    image\_datagen = ImageDataGenerator(\*\*aug\_dict)

    mask\_datagen = ImageDataGenerator(\*\*aug\_dict)

    image\_generator = image\_datagen.flow\_from\_dataframe(

        data\_frame,

        x\_col="filename",

        class\_mode=None,

        color\_mode=image\_color\_mode,

        target\_size=target\_size,

        batch\_size=batch\_size,

        save\_to\_dir=save\_to\_dir,

        save\_prefix=image\_save\_prefix,

        seed=seed)

    mask\_generator = mask\_datagen.flow\_from\_dataframe(

        data\_frame,

        x\_col="mask",

        class\_mode=None,

        color\_mode=mask\_color\_mode,

        target\_size=target\_size,

        batch\_size=batch\_size,

        save\_to\_dir=save\_to\_dir,

        save\_prefix=mask\_save\_prefix,

        seed=seed)

    train\_gen = zip(image\_generator, mask\_generator)

    for (img, mask) in train\_gen:

        img, mask = adjust\_data(img, mask)

        yield (img, mask)

* Trong bước tiếp theo, chúng tôi sẽ xác định sự phân chia giữa dữ liệu đào tạo và xác nhận tương ứng. Cả hai thực thể dữ liệu này phải được xem riêng biệt để mô hình không bị xem qua dữ liệu thử nghiệm. Trong tập hợp xác thực sẽ thực hiện thao tác xáo trộn tùy chọn sẽ trộn tất cả các hình ảnh trong tập dữ liệu và có thể lấy các mẫu ngẫu nhiên cho cả hình ảnh xác thực.

df = pd.DataFrame(data={"filename": train\_files, 'mask': mask\_files})

df\_train, df\_test = train\_test\_split(df, test\_size=0.1)

df\_train, df\_val = train\_test\_split(df\_train, test\_size=0.2)

* Ngoài ra thì ảnh cũng được scale về [0,1] và mask cũng đưa về 0 và 1 với bài toán segmentation chỉ gồm 1 lớp và background

def adjust\_data(img, mask):

    img = img / 255

    mask = mask / 255

    mask[mask > 0.5] = 1

    mask[mask <= 0.5] = 0

    return (img, mask)

1. ***Thiết kế mô hình U-Net***

Sau khi nhập các thư viện cần thiết, có thể tiếp tục xây dựng kiến ​​trúc U-Net. Xác định tất cả các tham số và giá trị tương ứng theo thứ tự và tiếp tục quá trình cho đến khi đạt đến phần cuối hoặc có một vài khối lặp đi lặp lại. Mô hình sử dụng ba khối lặp lại như được hiển thị trong biểu diễn kiến ​​trúc, đó là khối hoạt động tích chập, khối mã hóa và khối giải mã. Khối hoạt động tích chập được sử dụng để thực hiện hoạt động chính của việc lấy các tham số đầu vào đã nhập và xử lý một lớp kép các phép toán tích chập. Dử dụng giá trị của ‘padding same’ để duy trì các hình dạng kích thước. Các lớp phức hợp này được theo sau cùng với lớp chuẩn hóa(BatchNormalization). Cuối cùng, một lớp kích hoạt ReLU được thêm vào và nút đầu ra là hàm kích hoạt sigmoid.

conv1 = Conv2D(64, (3, 3), padding='same')(inputs)

    bn1 = Activation('relu')(conv1)

    conv1 = Conv2D(64, (3, 3), padding='same')(bn1)

    bn1 = BatchNormalization(axis=3)(conv1)

    bn1 = Activation('relu')(bn1)

    pool1 = MaxPooling2D(pool\_size=(2, 2))(bn1)

    conv2 = Conv2D(128, (3, 3), padding='same')(pool1)

    bn2 = Activation('relu')(conv2)

    conv2 = Conv2D(128, (3, 3), padding='same')(bn2)

    bn2 = BatchNormalization(axis=3)(conv2)

    bn2 = Activation('relu')(bn2)

    pool2 = MaxPooling2D(pool\_size=(2, 2))(bn2)

    conv3 = Conv2D(256, (3, 3), padding='same')(pool2)

    bn3 = Activation('relu')(conv3)

    conv3 = Conv2D(256, (3, 3), padding='same')(bn3)

    bn3 = BatchNormalization(axis=3)(conv3)

    bn3 = Activation('relu')(bn3)

    pool3 = MaxPooling2D(pool\_size=(2, 2))(bn3)

    conv4 = Conv2D(512, (3, 3), padding='same')(pool3)

    bn4 = Activation('relu')(conv4)

    conv4 = Conv2D(512, (3, 3), padding='same')(bn4)

    bn4 = BatchNormalization(axis=3)(conv4)

    bn4 = Activation('relu')(bn4)

    pool4 = MaxPooling2D(pool\_size=(2, 2))(bn4)

    conv5 = Conv2D(1024, (3, 3), padding='same')(pool4)

    bn5 = Activation('relu')(conv5)

    conv5 = Conv2D(1024, (3, 3), padding='same')(bn5)

    bn5 = BatchNormalization(axis=3)(conv5)

    bn5 = Activation('relu')(bn5)

    # decoder

    up6 = concatenate([Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(bn5), conv4], axis=3)

    conv6 = Conv2D(512, (3, 3), padding='same')(up6)

    bn6 = Activation('relu')(conv6)

    conv6 = Conv2D(512, (3, 3), padding='same')(bn6)

    bn6 = BatchNormalization(axis=3)(conv6)

    bn6 = Activation('relu')(bn6)

    up7 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(bn6), conv3], axis=3)

    conv7 = Conv2D(256, (3, 3), padding='same')(up7)

    bn7 = Activation('relu')(conv7)

    conv7 = Conv2D(256, (3, 3), padding='same')(bn7)

    bn7 = BatchNormalization(axis=3)(conv7)

    bn7 = Activation('relu')(bn7)

    up8 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(bn7), conv2], axis=3)

    conv8 = Conv2D(128, (3, 3), padding='same')(up8)

    bn8 = Activation('relu')(conv8)

    conv8 = Conv2D(128, (3, 3), padding='same')(bn8)

    bn8 = BatchNormalization(axis=3)(conv8)

    bn8 = Activation('relu')(bn8)

    up9 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(bn8), conv1], axis=3)

    conv9 = Conv2D(64, (3, 3), padding='same')(up9)

    bn9 = Activation('relu')(conv9)

    conv9 = Conv2D(64, (3, 3), padding='same')(bn9)

    bn9 = BatchNormalization(axis=3)(conv9)

    bn9 = Activation('relu')(bn9)

    conv10 = Conv2D(1, (1, 1), activation='sigmoid')(bn9)

1. ***Đào tạo mô hình***

Biên dịch và đào tạo mô hình để xem hiệu suất của nó trên dữ liệu. Tạo một hàm checkpoint để kiểm soát để lưu mô hình để có thể đưa ra dự đoán tốt nhất. Sử dụng độ đo lường hệ số tương đương, và iou và loss của hệ số tương đương. EPOCHS = 100, BATCH\_SIZE = 32, learning\_rate = 1e-4

opt = tf.keras.optimizers.Adam(learning\_rate)

model.compile(optimizer=opt, loss=dice\_coef\_loss, metrics=["binary\_accuracy", iou, dice\_coef])

callbacks = [ModelCheckpoint('unet\_brain\_mri\_seg\_coppy.hdf5', monitor='val\_iou', verbose=1, save\_best\_only=True, mode='max')]

model.fit(train\_gen,

steps\_per\_epoch=len(df\_train) / BATCH\_SIZE,

epochs=EPOCHS,

callbacks=callbacks,

validation\_data = test\_gener,

validation\_steps=len(df\_val) / BATCH\_SIZE)

1. ***Xây dựng giao diện***

root = tk.Tk()

root.title("GUI KẾT QUẢ PREDICT")

root.geometry("950x450")

def openfile():

filetypes = (

('All files', '\*.\*'),('text files', '\*.txt'))

filename = filedialog.askopenfilename(

title='Open a file',

initialdir='/',

filetypes=filetypes)

showinfo(

title=' FILE ĐÃ CHỌN',

message=filename

)

filemask = filename.replace(".","\_mask.")

img = cv2.imread(filename)

img = cv2.resize(img, (im\_height, im\_width))

img = img / 255

img = img[np.newaxis, :, :, :]

pred = model.predict(img)

fig = plt.figure(figsize=(7, 4))

plt.subplot(1, 3, 1)

plt.imshow(np.squeeze(img))

plt.title(' ẢNH MRI GỐC')

plt.subplot(1, 3, 2)

plt.imshow(np.squeeze(cv2.imread(filemask)))

plt.title('ẢNH MRI Mask')

plt.subplot(1, 3, 3)

plt.imshow(np.squeeze(pred) > .5)

plt.title('VÙNG LỖI DỰ ĐOÁN')

canvas = FigureCanvasTkAgg(fig) # creating the Tkinter canvas containing the Matplotlib figure

canvas.draw()

canvas.get\_tk\_widget().place(x=20, y=80)

plot\_lable =Label(root, text = "\_\_\_\_\_MÔ HÌNH NHẬN DẠNG ĐIỂM BẤT THƯỜNG TRÊN NÃO QUA HÌNH ẢNH CHỤP MRI\_\_\_\_\_").place(x=100, y=20)

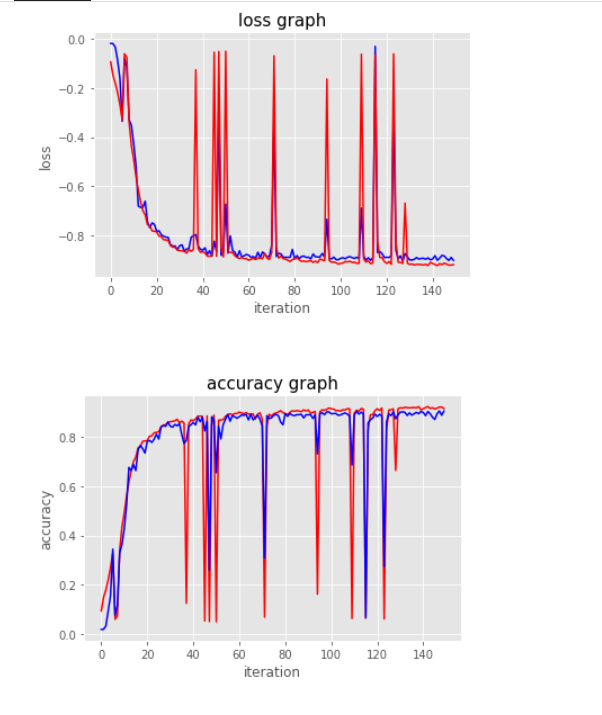
plot\_button = tk.Button( command=openfile, height=2, width=10, text="CHỌN ẢNH").place(x=120, y=40)

root.mainloop()

1. ***Đánh giá mô hình đào tạo***

Hàm loss được đo bằng val\_loss và loss có xu hướng giảm.

Hàm accuracy được đo bằng IoU và val\_IoU xu hướng tăng dần.



1. Biểu đồ đánh giá quá trình đào tạo mô hình

* Load file weights có trọng số tốt nhất tiến hành dự đoán trên tập dữ liệu test

model = load\_model('unet\_brain\_mri\_seg.hdf5', custom\_objects={'dice\_coef\_loss': dice\_coef\_loss, 'iou': iou, 'dice\_coef': dice\_coef})

* Đánh giá mô hình trên dữ liệu thử nghiệm thông qua evaluate():

test\_gen = train\_generator(df\_test, BATCH\_SIZE,

dict(),

target\_size=(im\_height, im\_width))

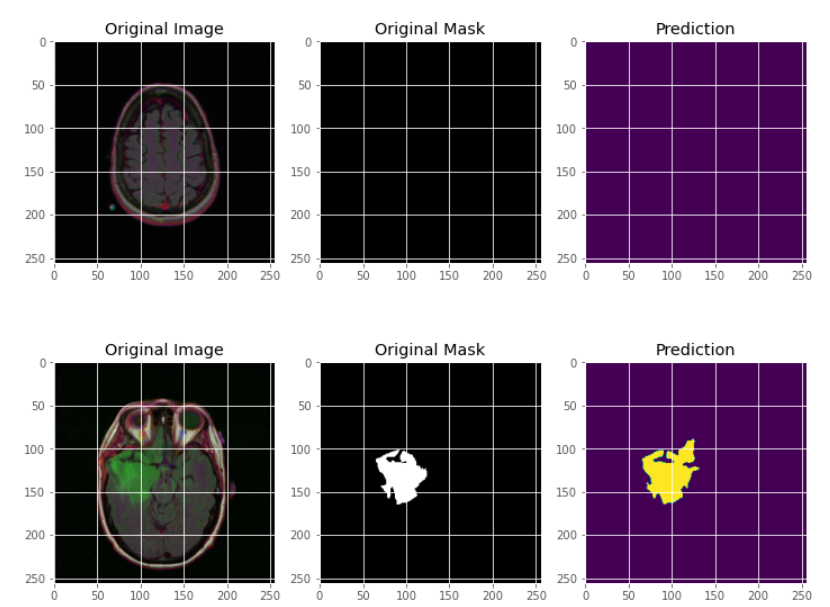
results = model.evaluate(test\_gen, steps=len(df\_test) / BATCH\_SIZE)

print("Test lost: ",results[0])

print("Test IOU: ",results[1])

print("Test Dice Coefficent: ",results[2])

* Hình ảnh dự đoán trên tập test

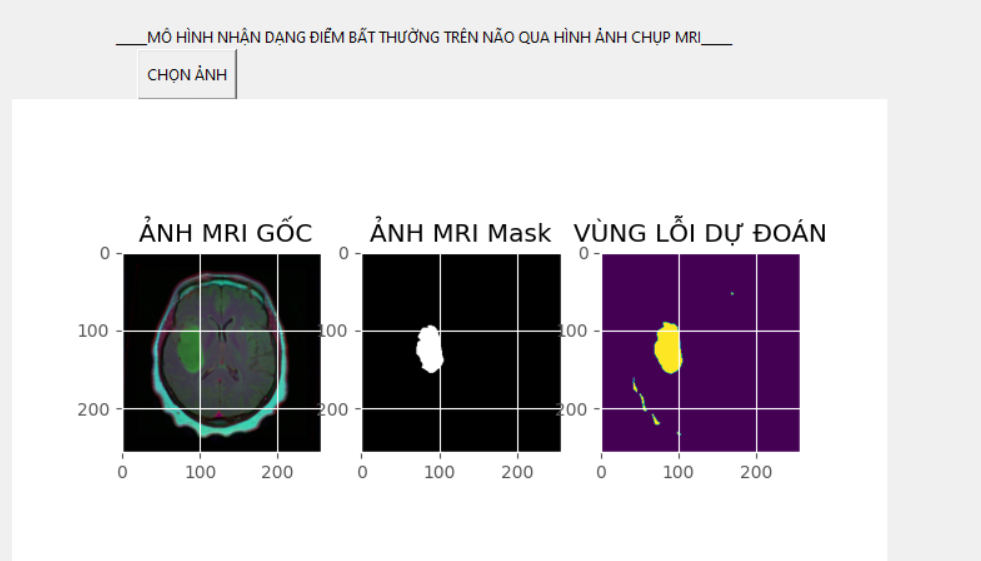


1. Kết quả dự đoán trên tập dữ liệu

## 2.3. Kết luận

Mô hình Unet sau khi huấn luyện qua 150 epochs trong vòng 4 giờ thu được kết quả như sau:

* Loss: -0.91.
* Binary\_accuracy: 0.99
* IOU đạt mực tốt nhất: 0.8608.
* Dice Coefficent đạt mức tốt nhất: 0.9248.



1. *Giao diện cuối cùng*

Kiến trúc U-Net là một trong những dấu mốc quan trọng và mang tính cách mạng trong lĩnh vực học sâu. Trong khi tài liệu nghiên cứu ban đầu giới thiệu kiến ​​trúc U-Net là để giải quyết nhiệm vụ Phân đoạn Hình ảnh Y sinh, nó không chỉ giới hạn ở ứng dụng đơn lẻ này. Mô hình có thể và vẫn có thể giải quyết các vấn đề phức tạp nhất trong học sâu. Mặc dù một số yếu tố trong kiến ​​trúc ban đầu đã lỗi thời.

Mô hình nhận dạng và khoanh vùng lỗi trên ảnh MRI mang lại hiệu quả khá tốt khoanh vùng đước hầu hết các vùng có chứa khối u trên ảnh, nhưng vẫn còn chưa đạt đến mức khoanh vùng dữ liệu có khối u chính xác tuyệt đối do qua đi nhiều lớp nên lượng dữ liệu trên ảnh đã bị mất đi 1 một phần để cải thiện hiệu xuất mô hình thì giải pháp là sử dụng. LadderNet, U-Net with attention và convolutional U-Net (R2-UNet), và các mạng tương tự khác được lấy thành công từ Mô hình U-Net ban đầu. Ngoài ra có thể sử dụng các mô hình mạng được train sẵn trong phần encoder ví dụ như VGG, Alexnet, Resnet,.... Dùng các pre\_train model để khởi tạo tham số cho phần encoder. Còn đối với phần decoder ta cũng có thể xây dựng linh hoạt ngoài việc concat đối xứng, có thể làm những cách khác nhau như Upsample luôn hoặc add đối xứng các layer lại với nhau.

## 2.4. Tài liệu tham khảo

[1] Olaf Ronneberger, Philipp Fischer, Thomas Brox: U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015; arXiv:1505.04597.

[2] Howard Jeremy. Fastai - Dynamic U-Net. https://www.youtube.com/watch?v=0frKXR-2PBY. Accessed 2 September 2019.