ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

DƯƠNG MINH TÂM – 18521367

LÊ TRƯỜNG THỊNH - 18521438

BÁO CÁO ĐỒ ÁN MÔN HỌC

HIỆN THỰC MẠNG NƠ-RON TÍCH CHẬP TRÊN FPGA SỬ DỤNG KIẾN TRÚC VGG16

Implementation of Convolutional Neural Network on FPGA using VGG16 architecture

TP. HỒ CHÍ MINH, 2021

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

DƯƠNG MINH TÂM – 18521367

LÊ TRƯỜNG THỊNH - 18521438

BÁO CÁO ĐỒ ÁN MÔN HỌC

HIỆN THỰC MẠNG NƠ-RON TÍCH CHẬP TRÊN FPGA SỬ DỤNG KIẾN TRÚC VGG16

Implementation of Convolutional Neural Network on FPGA using VGG16 architecture

GIẢNG VIÊN HƯỚNG DẪN

ThS. Trương Văn Cương

TP. HỒ CHÍ MINH, 2021

MỤC LỤC

DANH MỤC HÌNH

Chương 1. GIỚI THIỆU TỔNG QUAN

1.1. Giới thiệu

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao và áp dụng vào thực tiễn như hiện nay. Ví dụ như Facebook, Google, … đã đưa vào sản phẩm của mình chức năng nhận diện khuôn mặt người dùng. CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh.

Tuy nhiên, do số lượng tham số cần tính toán trong CNN khá lớn đòi hỏi tài nguyên với tốc độ xử lý cao, người ta thường chọn GPU là phần cứng để huấn luyện và hiện thực mạng. Mặc dù vậy, GPU lại có những điểm hạn chế như: tiêu tốn nhiều năng lượng sử dụng, chi phí cao và tốc độ xử lý chậm hơn so với FPGA. FPGA có khả năng cung cấp hàng ngàn đơn vị bộ nhớ cho việc tính toán giúp tăng tốc độ xử lý, giảm chi phí tính toán. Bên cạnh đó, FPGA còn có khả năng tái lập trình, cho phép tối ưu về mặt thông lượng và sử dụng ít năng lượng hơn. Chính nhưng điểm nổi bật trên khiến cho FPGA trở nên thích hợp trong các ứng dụng nhúng. Với sự phát triển của FPGA như ngày nay và sự hỗ trợ nhiệt tình về phần mềm, mã nguồn mở từ các nhà sản xuất, chúng ta có thể dễ dàng xây dựng mô hình CNN trên FPGA cho các ứng dụng xử lý thời gian thực.

1.2. Mục tiêu của đề tài

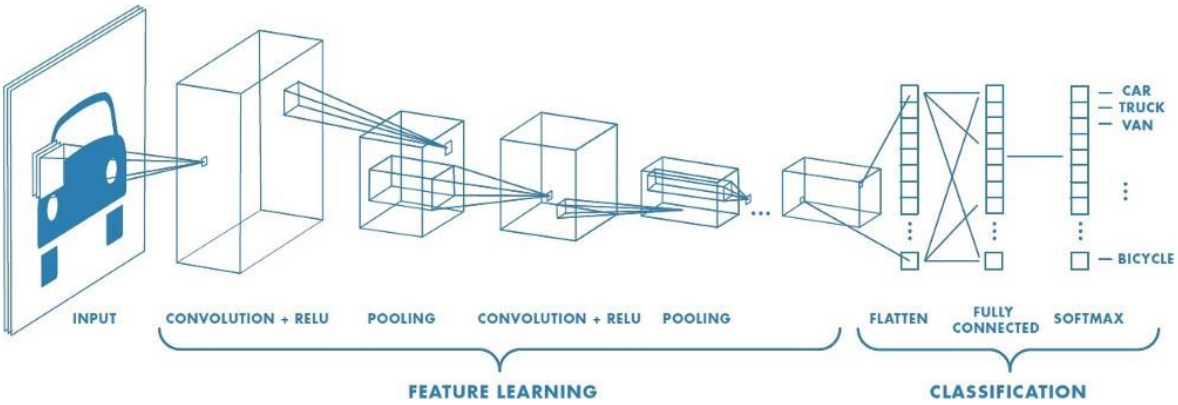
Hiện thực mạng VGG16 với input là một bức ảnh 224 x 224 x 3 output là xác xuất phần trăm chính xác của các class, ứng dụng trong việc nhận dạng phân biệt chó và mèo.

Chương 2. TÌM HIỂU TỔNG QUAN

2.1. Convolutional Neural Network

Kiến trúc cơ bản trong một mạng CNN bao gồm: lớp tích chập (Convolutional); lớp kích hoạt phi tuyến ReLU (Rectified Linear Unit); lớp lấy mẫu (Pooling); lớp kết nối đầy đủ (Fully connected) được thay đổi về số lượng và cách sắp xếp để tạo ra các mô hình huấn luyện phù hợp cho từng bài toán khác nhau.

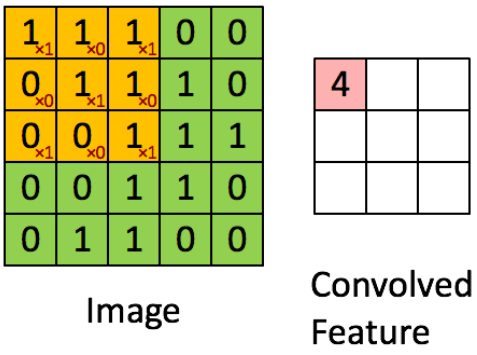
CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh. Để tìm hiểu tại sao thuật toán này được sử dụng rộng rãi cho việc nhận dạng (detection)



Hình 2.1: Kiến trúc cơ bản của một mạng CNN

2.1.1. Convolutional Layer – Lớp tích chập

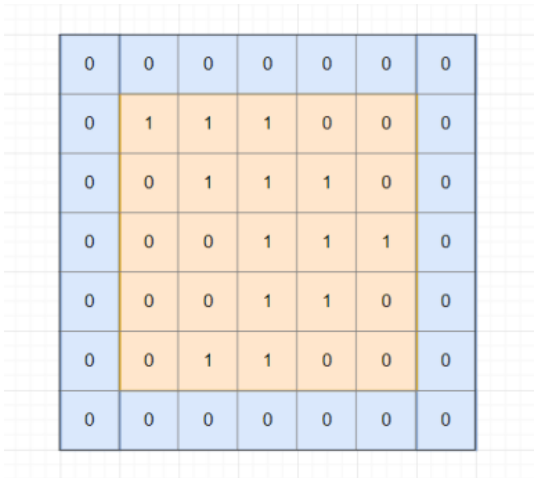
Convolution hay tích chập là nhận ma trận Filter với ma trận ảnh. Kết quả được một ma trận gọi là Convoled feature, kết quả của lớp này sẽ được truyền sang lớp tiếp theo.



Hình 2.2: Tích chập 2 chiều với stride = 1 và padding = 0

Để tính tích chập cần quan tâm đến hai thông số là Stride và Padding.

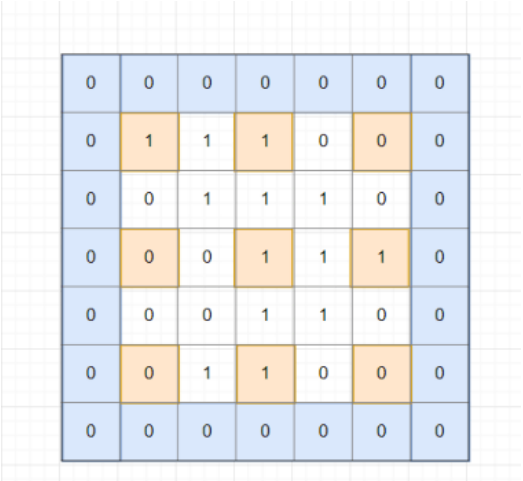
Mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Để ma trận Y thu được có kích thước bằng ma trận X => Thêm giá trị 0 ở viền ngoài ma trận X. Phép tính này gọi là convolution với **padding=1**. Padding=k nghĩa là thêm k vector 0 vào mỗi phía của ma trận.



Hình 2.3: Convolution với padding = 1

Kernel sẽ được trượt qua ảnh, với mỗi lần trượt ta sẽ tính kết quả tại vị trí đầu ra. Số bước ở mỗi lần trượt kernel được gọi là Stride. Thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1

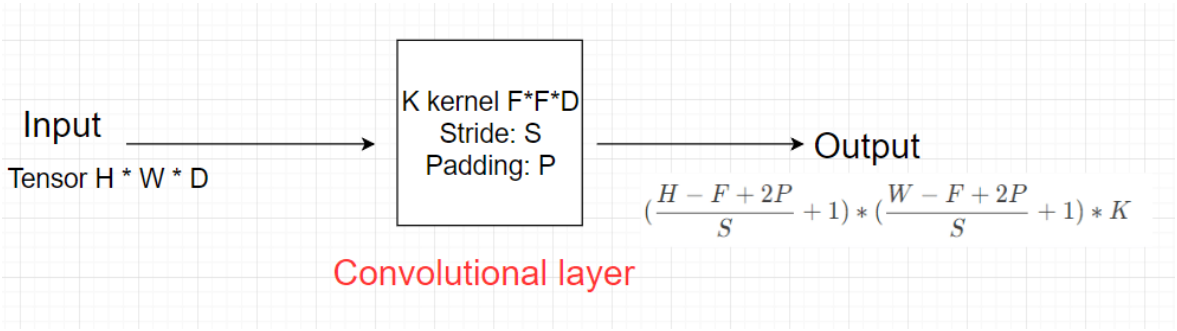
Tuy nhiên nếu **stride=k** (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử $x_{1+i*k,1+j*k}$. Ví dụ k = 2.



Hình 2.4: Convolution với stride = 2 và padding = 1

Convolutional layer tổng quát

VD input của 1 convolutional layer tổng quát là tensor kích thước H * W * D. Kernel có kích thước F * F * D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.



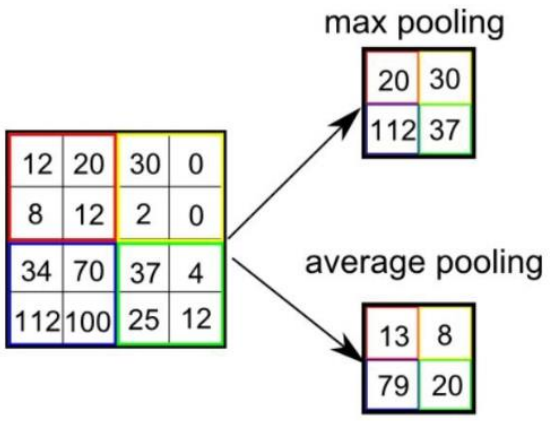
Hình 2.5: Convolutional layer áp dụng K kernel

2.1.2. Pooling Layer – Lớp tổng hợp

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

Hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên .

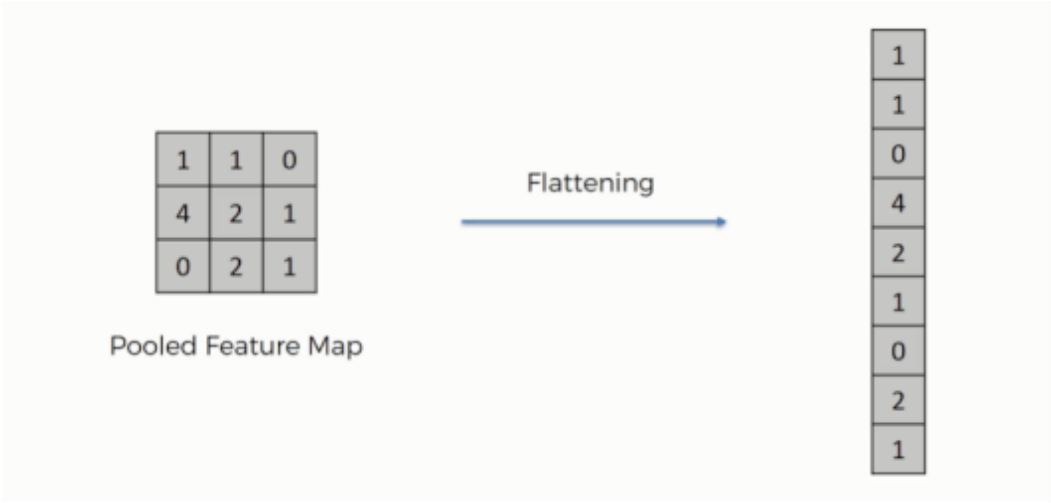
Có hai loại Pooling là Max Pooling (trả về giá trị lớn nhất trong cửa sổ trượt) và Average Pooling (trả về giá trị trung bình trong cửa sổ trượt).



Hình 2.6: Lớp Pooling khi Stride bằng 2

2.1.3. Fully Connected Layer – Lớp kết nối đầy đủ

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,…) thì tensor của output của layer cuối cùng, kích thước H*W*D, sẽ được chuyển về 1 vector kích thước (H*W*D)



Hình 2.7: Đầu vào của lớp sẽ được "làm phẳng" thành ma trận một chiều.

2.1.4. Các hàm kích hoạt phi tuyến tính ReLU

Lớp kích hoạt phi tuyến ReLU: được xây dựng để đảm bảo tính phi tuyến của mô hình huấn luyện sau khi đã thực hiện một loạt các phép tính toán tuyến tính qua các lớp tích chập. Lớp kích hoạt phi tuyến sử dụng các hàm kích hoạt phi tuyến như ReLU để giới hạn phạm vi biên độ cho phép của giá trị đầu ra. Hàm ReLU được chọn do cài đặt đơn giản, tốc độ xử lý nhanh mà vẫn đảm bảo được tính toán hiệu quả. Phép tính toán của hàm ReLU chỉ đơn giản là chuyển tất cả các giá trị âm thành giá trị 0.

$$f(x) = max(0, x)$$

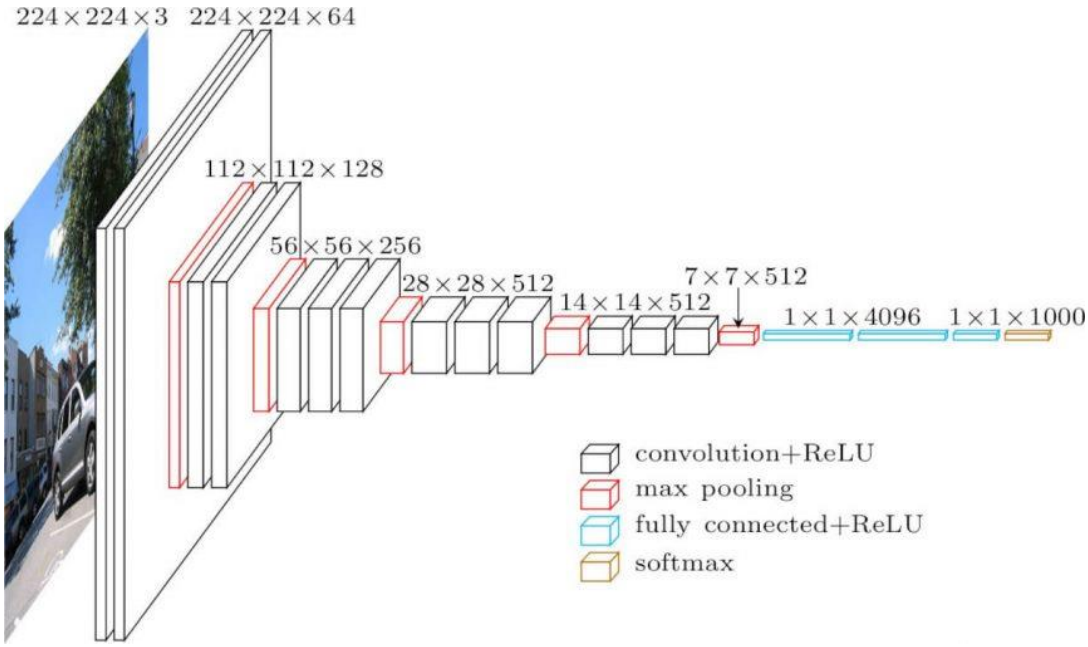
Lớp ReLU được áp dụng ngay phía sau lớp tích chập, với đầu ra là một ảnh mới có kích thước giống với ảnh đầu vào, các giá trị điểm ảnh cũng hoàn toàn tương tự, trừ các giá trị âm đã bị loại bỏ.

2.2. VGG16 Network

2.2.1. Nguồn gốc

VGG16 là một mô hình mạng nơ-ron tích chập được đề xuất bởi K. Simonyan và A. Zisserman từ Đại học Oxford trong bài báo **"Very Deep Convolutional Networks for Large-Scale Image Recognition"**. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Đây là một trong những mô hình nổi tiếng được nộp cho ILSVRC-2014. Nó được cải tiến hơn AlexNet bằng cách thay thế các kernel-sized filters. VGG16 đã được train trong nhiều tuần và sử dụng NVIDIA Titan Black GPU's.

2.2.2. Kiến trúc



Phân tích:

- Convolutional layer: kích thước 3*3, padding=1, stride=1. Mặc định sẽ là stride=1 và padding = 1 để cho output cùng width và height với input.
- Pool/2: max pooling layer với size 2*2
- 3*3 conv, 64: thì 64 là số kernel áp dụng trong layer đấy, hay depth của output của layer đấy.
- Càng các convolutional layer sau thì kích thước width, height càng giảm nhưng depth càng tăng.
- Sau khá nhiều convolutional layer và pooling layer thì dữ liệu được flatten và cho vào fully connected layer.

Kết quả:

VGG-16 là một trong những kiến trúc hoạt động tốt nhất trong thử thách ILSVRC năm 2014, đứng thứ nhất trong nhiệm vụ phân loại với lỗi phân loại top 5 là *7,32%* (chỉ sau GoogLeNet với lỗi phân loại *6,66%*). Nó cũng là người chiến thắng trong nhiệm vụ bản địa hóa với *25,32%* lỗi bản địa hóa.

Những thách thức của VGG 16:

Nó rất chậm để đào tạo (mô hình VGG ban đầu được đào tạo trên GPU Nvidia Titan trong 2-3 tuần).

Kích thước của trọng lượng imageNet được đào tạo VGG-16 là *528* MB. Vì vậy, nó chiếm khá nhiều dung lượng ổ đĩa và băng thông khiến nó hoạt động kém hiệu quả.

Lớp 1 (Tích chập):
- Số bộ lọc: 64
- Kích thước bộ lọc: 3 x 3 x 64
- Bộ nhớ: 224 x 224 x 64 = 3,2M
- Số lượng tham số: (3 x 3 x 3) x 64 = 1.728

*** Lớp 2** (Tích chập):
- Đầu vào: 224 x 224 x 64
- Số bộ lọc: 64
- Kích thước bộ lọc: 3 x 3 x 64
- Bộ nhớ: 224 x 224 x 64 = 3,2M
- Số lượng tham số: (3 x 3 x 64) x 64 = 36.864

*** Lớp chuyển tiếp sang lớp 3** (Lấy mẫu):
- Size = (2,2)

- Stride = 2
- Padding = 0
- Bộ nhớ: 112 x 112 x 64 = 800K

Kích thước đầu ra của dữ liệu giảm 1/2, từ (224 x 224 x 3)
xuống (112 x 112 x 3), và chiều sâu được giữ nguyên

Lớp 4 (Tích chập):
- Đầu vào: 112 x 112 x 3
- Số bộ lọc: 128
- Kích thước bộ lọc: 3 x 3 x 128
- Bộ nhớ: 112 x 112 x 128 = 1,6M
- Số lượng tham số: (3 x 3 x 128) x 128 = 147.456

*** Lớp chuyển tiếp sang lớp 5** (Lấy mẫu):
- Size = (2,2)
- Stride = 2
- Padding = 0
- Bộ nhớ: 56 x 56 x 128 = 400K

Kích thước đầu ra của dữ liệu giảm 1/2, từ (112 x 112 x 3)
xuống (56 x 56 x 3), và chiều sâu được giữ nguyên

***Lớp 5** (Tích chập):
- Đầu vào: 56 x 56 x 3
- Số bộ lọc: 256
- Kích thước bộ lọc: 3 x 3 x 256
- Bộ nhớ: 56 x 56 x 256 = 800K
- Số lượng tham số: (3 x 3 x 128) x 256 = 294.912

***Lớp 6** (Tích chập):
- Đầu vào: 56 x 56 x 3
- Số bộ lọc: 256
- Kích thước bộ lọc: 3 x 3 x 256
- Bộ nhớ: 56 x 56 x 256 = 800K
- Số lượng tham số: (3 x 3 x 256) x 256 = 589.824

*** Lớp 7** (Tích chập):
- Đầu vào: 56 x 56 x 3
- Số bộ lọc: 256
- Kích thước bộ lọc: 3 x 3 x 256
- Bộ nhớ: 56 x 56 x 256 = 800K
- Số lượng tham số: (3 x 3 x 256) x 256 = 589.824

*** Lớp chuyển tiếp sang lớp 8** (Lấy mẫu):
- Size = (2,2)
- Stride = 2
- Padding = 0
- Bộ nhớ: 28 x 28 x 256 = 200K

Kích thước đầu ra của dữ liệu giảm 1/2, từ (56 x 56 x 3) xuống
(28 x 28 x 3), và chiều sâu được giữ nguyên

*** Lớp 8** (Tích chập):
- Đầu vào: 28 x 28 x 3
- Số bộ lọc: 512

- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 28 x 28 x 512 = 400K
- Số lượng tham số: (3 x 3 x 256) x 512 = 1.179.648

*** Lớp 9** (Tích chập):
- Đầu vào: 28 x 28 x 3
- Số bộ lọc: 512
- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 28 x 28 x 512 = 400K
- Số lượng tham số: (3 x 3 x 512) x 512 = 2.359.296

*** Lớp 10** (Tích chập):
- Đầu vào: 28 x 28 x 3
- Số bộ lọc: 512
- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 28 x 28 x 512 = 400K
- Số lượng tham số: (3 x 3 x 512) x 512 = 2.359.296

*** Lớp chuyển tiếp sang lớp 11** (Lấy mẫu):
- Size = (2,2)
- Stride = 2
- Padding = 0
- Bộ nhớ: 14 x 4 x 512 = 100K

Kích thước đầu ra của dữ liệu giảm 1/2, từ (28 x 28 x 3) xuống (14 x 14 x 3), và chiều sâu được giữ nguyên

*** Lớp 11** (Tích chập):
- Đầu vào: 14 x 14 x 3
- Số bộ lọc: 512
- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 14 x 14 x 512 = 100K
- Số lượng tham số: (3 x 3 x 512) x 512 = 2.359.296

*** Lớp 12** (Tích chập):
- Đầu vào: 14 x 14 x 3
- Số bộ lọc: 512
- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 14 x 14 x 512 = 100K
- Số lượng tham số: (3 x 3 x 512) x 512 = 2.359.296

*** Lớp 13** (Tích chập):
- Đầu vào: 14 x 14 x 3
- Số bộ lọc: 512
- Kích thước bộ lọc: 3 x 3 x 512
- Bộ nhớ: 14 x 14 x 512 = 100K
- Số lượng tham số: (3 x 3 x 512) x 512 = 2.359.296

*** Lớp chuyển tiếp sang lớp 14** (Lấy mẫu):
- Size = (2,2)
- Stride = 2
- Padding = 0
- Bộ nhớ: 7 x 7 x 512 = 25K

Kích thước đầu ra của dữ liệu giảm 1/2, từ (14 x 14 x 3) xuống (7 x 7 x 3), và chiều sâu được giữ

nguyên

*** Lớp 14** (Kết nối đầy đủ):

- Đầu vào: 1 x 1 x 4.096
- Bộ nhớ: 4.096K
- Số lượng tham số: 7 x 7 x 512 x 4.096 = 102.760.448

*** Lớp 15** (Kết nối đầy đủ):

- Đầu vào: 1 x 1 x 4.096
- Bộ nhớ: 4.096K
- Số lượng tham số: 4.096 x 4.096 = 16.777.216

*** Lớp 16** (Kết nối đầy đủ):

- Đầu vào: 1 x 1 x 4.096
- Bộ nhớ: 1.000K
- Số lượng tham số: 4.096 x 1.000 = 4.096.000

Chương 3. THIẾT KẾ TỔNG QUAN

Chương 4. THIẾT KẾ CHI TIẾT

Chương 5. MÔ PHỎNG THIẾT KẾ

Chương 6. KẾT LUẬN VÀ ĐÁNH GIÁ

Chương 7. TÀI LIỆU THAM KHẢO

[1] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Karen Simonyan ∗ & Andrew Zisserman + Visual Geometry Group, Department of Engineering Science, University of Oxford {karen,az}@robots.ox.ac.uk