ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN MÔN HỌC**

**NHẬN DẠNG – CS338**

**NHẬN DIỆN TRÁI CÂY VÀ RAU CỦ**

LỚP CS338.L21

Giảng viên hướng dẫn: **ThS. Đỗ Văn Tiến**

Nhóm sinh viên thực hiện đề tài:

1. Phan Gia Huy 18520068

2. Nguyễn Quyết Thắng 18520152

*TP. Hồ Chí Minh, tháng 07 năm 2021*

**MỤC LỤC**

[I. GIỚI THIỆU 2](#_Toc77284775)

II. KIẾN THỨC TÌM HIỂU 2

[III. CÁC BƯỚC THỰC HIỆN 6](#_Toc77284777)

[IV. KẾT QUẢ 10](#_Toc77284778)

[V. PHÂN CÔNG NHIỆM VỤ 10](#_Toc77284779)

[VI. TÀI LIỆU THAM KHẢO 11](#_Toc77284780)

[VII. LINK GITHUB 11](#_Toc77284781)

VIII. LINK DEMO 11

# GIỚI THIỆU

Hiện nay, ở nước ta nói riêng và ở các nước đang phát triển có nền nông nghiệp là một trong các ngành sản xuất chủ yếu, quá trình thu hoạch, phân loại và đánh giá chất lượng các loại nông sản, đặc biệt là hoa quả, chủ yếu còn phải thực hiện bằng tay. Đây là công việc tốn nhiều thời gian, công sức của con người, là rào cản đối với mở rộng phát triển quy mô sản xuất nông nghiệp. Vì vậy nếu áp dụng những kiến thức Deeplearning để xây dựng một ứng dụng nhận diện trái cây sẽ giúp cho bà con nông dân đỡ vất vả hơn.

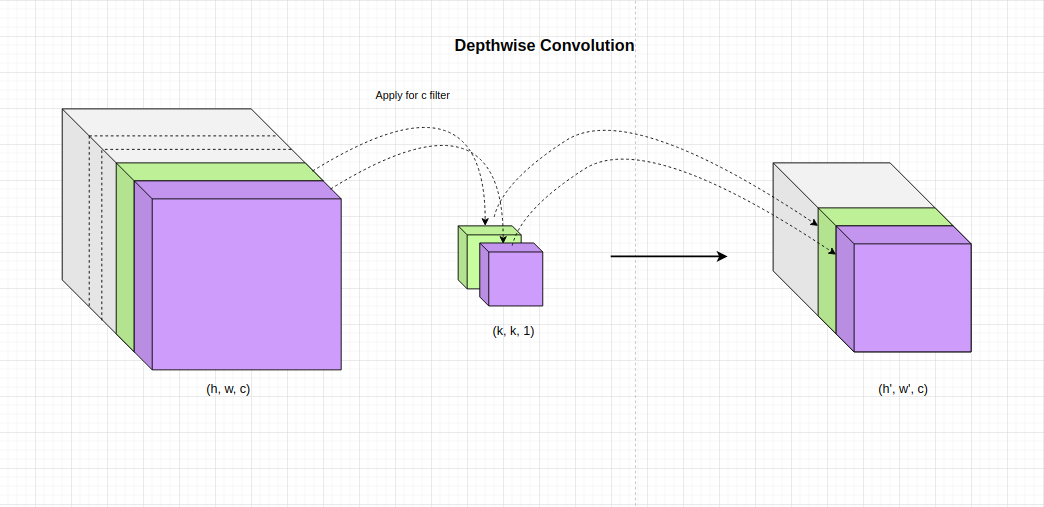
1. **KIẾN THỨC TÌM HIỂU**

MobileNet là mô hình được phát triển bởi đội ngũ Google, mô hình sử dụng cách tính tích chập mang tên DSC(Depthwise Separable Convolution) nhằm giảm kích thước mô hình và giảm dộ phức tạp tính toán. Do đó, MobileNet thường được sử dụng cho các ứng dụng thị giác máy tính trên các thiết bị nhỏ gọn như điện thoại thông minh hay thiết bị nhúng.

### **Depthwise Separable Convolution:**

Đây là layer được giới thiệu ở MobileNetV1. Depthwise Separable Convolution là một loại tích chập mà trong đó chúng ta áp dụng một bộ lọc tích chập duy nhất cho mỗi kênh đầu vào. Nói cách khác, Depthwise Separable Convolution sẽ loại bỏ sự phụ thuộc vào độ sâu khi tính mà vẫn tạo ra được một output shape có kích thước tương đương so với tích chập thông thường. Quá trình sẽ chia thành 2 bước:

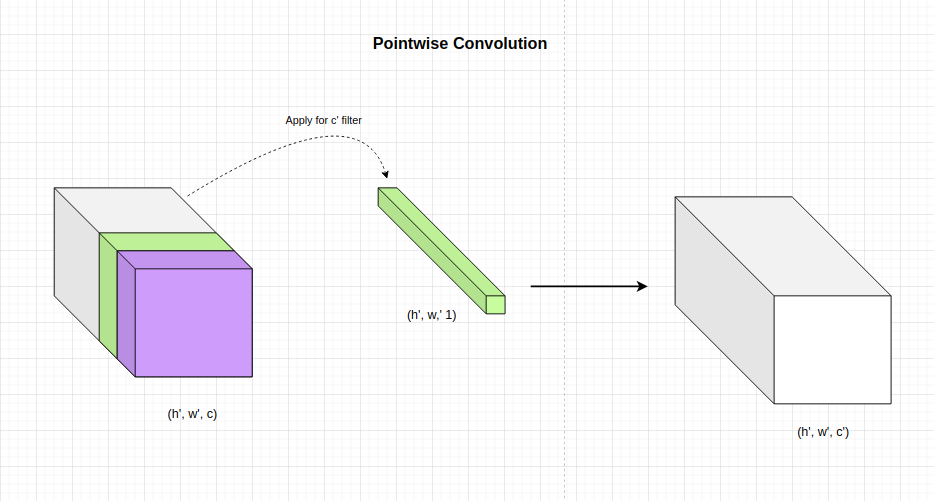
* Depthwise Convolution: chúng ta sẽ chia khối input thành những lát cắt ma trận theo chiều sâu. Thực hiện tính tích chập trên từng lát cắt như hình.

****

Hình II.A.1: Depthwise Convolution

Mỗi một channel sẽ áp dụng một bộ lọc khác nhau và hoàn toàn không chia sẻ tham số. Điều này có ba tác dụng chính cho mô hình:

* + Nhận diện đặc trưng: Quá trình học và nhận diện đặc trưng sẽ được tách biệt theo từng bộ lọc. Nếu đặc trưng trên các channel là khác xa nhau thì sử dụng các bộ lọc riêng cho channel sẽ chuyên biệt hơn trong việc phát hiện các đặc trưng. Chẳng hạn như đầu vào là ba kênh RGB thì mỗi kênh áp dụng một bộ lọc khác nhau chuyên biệt.
  + Giảm thiểu khối lượng tính toán: Để tạo ra một điểm pixel trên output thì tích chập thông thường cần sử dụng k×k×c phép tính trong khi tích chập chiều sâu tách biệt chỉ cần k×k phép tính.
  + Giảm thiếu số lượng tham số : Ở tích chập chiều sâu cần sử dụng c×k×k tham số. Số lượng này ít hơn gấp c′ lần so với tích chập chiều sâu thông thường.
* **Tích chập điểm (Pointwise Convolution)**: Có tác dụng thay đổi độ sâu của output bước trên từ c sang c′. Chúng ta sẽ áp dụng c′ bộ lọc kích thước 1×1×c. Như vậy kích thước width và height không thay đổi mà chỉ độ sâu thay đổi.

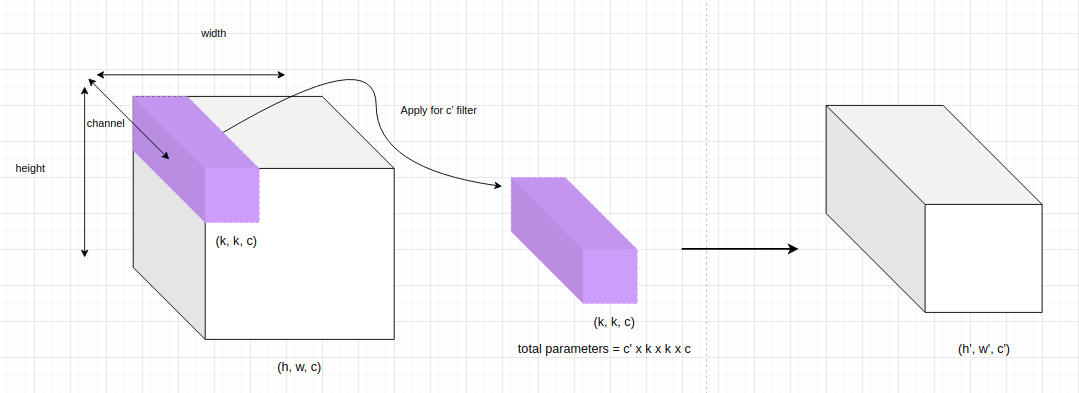


Hình II.A.2: Pointwise Convolution

Kết quả sau cùng chúng ta thu được là một output có kích thước h′×w′×c′. Số lượng tham số cần áp dụng ở trường hợp này là c′×c.

* 1. **So sánh Convolution và Depthwise Separable Convolution:**

Nhắc lại về Convolution thông thường, chúng ta có một đầu vào kích thước h×w×c, tích chập thông thường sẽ cần k×k×c tham số để thực hiện tích chập trên toàn bộ độ sâu của layers. Mỗi một bộ lọc sẽ tạo ra một ma trận output kích thước h′×w′×1. Áp dụng c′ bộ lọc khác nhau ta sẽ tạo ra đầu ra có kích thước h′×w′×c′ (các ma trận output khi áp dụng tích chập trên từng bộ lọc sẽ được concatenate theo chiều sâu) như hình minh họa bên dưới



Hình II.B.1: Convolution

2.1 So sánh về số lượng Parameter:

Số lượng Parameter cần thiết để tạo được khối output có kích thước **h’** x **w’** x **c’** từ input có kích thước **h** x **w** x **c**?

Như chúng ta đã biết thì Convolution thông thường sẽ sử dụng **c′** × **k** × **k** × **c** parameters.

Còn DSC thì sử dụng **k** × **k** × **c** parameters ở bước Depthwise convolutionvà **c’** x **c** parameters ở bước Pointwise convolution, tổng cộng DSC sử dụng **k** × **k** × **c** + **c’** x **c** parameter thông qua hai bước.

Trong đó **k**: kích thước kernel

**c’:**  số lượng của bộ lọc

Ta có tỷ lệ số lượng parameter giữa Convolution và DSC:

= =

Thông thường **c′** sẽ lớn hơn **c** vì càng ở các layer sau thì độ sâu càng lớn, nên **c’** sẽ tăng, khi **c’** tăng đến xấp xỉ **k** x **k**, biểu thức trên sẽ bằng:

Tức số lượng parameter của Convolution gấp đến lần số lượng parameter trong DSC, khi số lớp layer đủ nhiều. Điều này giúp giảm đáng kể kích thước của mô hình, đồng thời nó lý giải cho việc vì sao các model ứng dụng DSC có thể nhỏ hơn vài chục lần model ứng dụng Convolution thông thường.

Vì số lượng parameter ít hơn nhiều khi áp dụng DSC, Mobilenet model có kích thước đủ nhỏ để có thể deploy lên điện thoại và các thiết bị nhúng.

2.2 So sánh về số lượng phép toán phải thực hiện:

Số lượng phép tính cần thiết để tạo được khối output có kích thước **h’** x **w’** x **c’** từ input có kích thước **h** x **w** x **c**?

Convolution thông thường sẽ sử dụng (**h’** x **w’** x **c’**) x (**k** × **k** × **c)** phép tính, trong đó **h’** x **w’** x **c’** là số lượng pixel cần tính và **k** × **k** × **c** là số phép nhân để tính một pixel.

DSC phải thực hiện số phép tính lần lượt trên hai bước:

* Bước Depthwise convolution: (**h’** x **w’** x **c**) x (**k** × **k)** phép nhân
* Bước Pointwise convolution: (**h’** x **w’** x **c**) x (**h’** x **w’)**

Tổng lại qua hai bước, DSC cần thực hiện (**h’** x **w’** x **c**) x (**k** × **k) +** (**h’** x **w’** x **c**) x (**h’** x **w’)** phép nhân.

Ta có tỷ lệ số lượng phép nhân giữa Convolution và DSC:

= =

Có thể thấy **c’** sẽ càng tăng đối với các layer ở phía sau, vì thế **k** × **k** × **c’** sẽ tăng nhanh khi số layer càng nhiều, khi đó tỷ lệ trên sẽ càng lớn.

Điều này có nghĩa là khi số lượng layer đủ nhiều thì tỷ lệ trên sẽ lớn, tức là số phép tính mà Convolution cần sử dụng sẽ nhiều hơn rất nhiều so với DSC, đây chính là lý do vì sao chi phí tính toán của DSC thấp hơn nhiều so với Convolution thông thường.

Với chi phí tính toán thấp này, một lần nữa các model ứng dụng DSC, cụ thể hơn là Mobilenet model là những model phù hợp để deploy trên các thiết bị cấu hình yếu.

# CÁC BƯỚC THỰC HIỆN

* 1. Download Dataset và chọn model

Dataset là tập dữ liệu gồm 36 loại trái cây và rau củ được lấy từ kaggle. Để có thể deploy trên điện thoại thì ta sẽ sử dụng model MobileNet.

* 1. Tiền xử lý dữ liệu

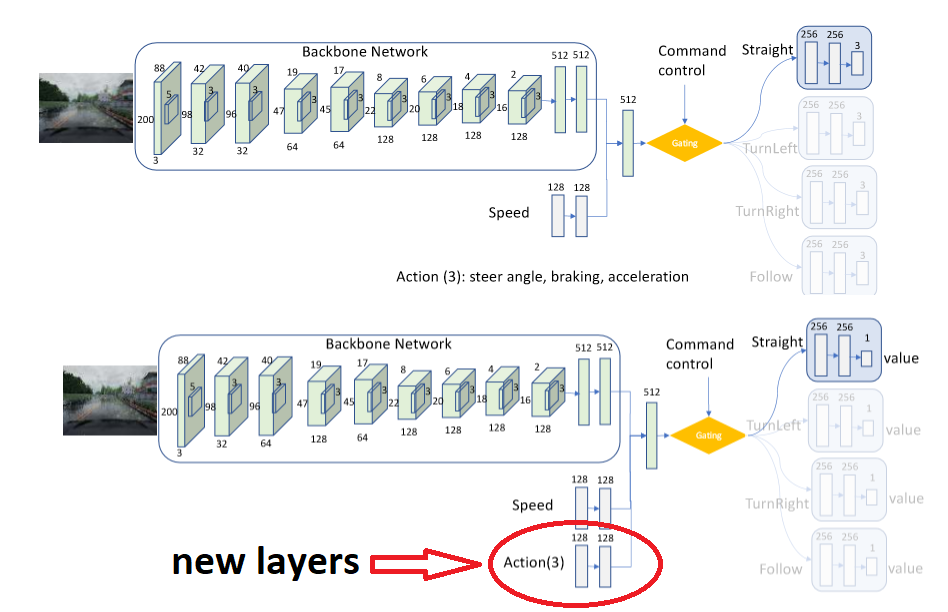
Trước khi train model, ta cần phải resize kích thước ảnh về cùng kích thước với input của model và Normalization ảnh. Sau đó, ta sẽ tăng cường data bằng các phép xoay ảnh, lật ảnh, phóng to, thu nhỏ ảnh.

* 1. Transfer learning

Transfer Learning là cách để các model truyền đạt cho nhau khả năng mà mỗi model có thể làm được. Một model có thể học trên source tasks nào đó và rồi pretrained model này được sử dụng cho model khác để model mới đó học trên target tasks nhanh hơn.

Cụ thể hơn:

* Một pretrained model đã được train trên source tasks cụ thể nào đó, khi đó một phần hay toàn bộ pretrained model có thể được tái sử dụng phụ thuộc vào nhiệm vụ của mỗi layer trong model đó
* Một model mới sử dụng một phần hay toàn bộ pretrained model để học một target tasks và tùy vào nhiệm vụ của mỗi layer mà model mới có thể thêm các layer khác dựa trên pretrained model sẵn có.



Hình III.A.1: Mô phỏng về Transfer Learning

Để sử dụng hiệu quả Pretrained Model:

* Thêm các Layer phù hợp với Target task và loại bỏ các Layer của Pretrained model không dùng đến. Để làm được điều này cần hiểu rõ về vai trò của từng layer và mục đích của chúng
* Có các Strategy(chiến lược) train tốt

Để chọn được Strategy train tốt, cần quan tâm đến khái niệm Fine-Tuning.

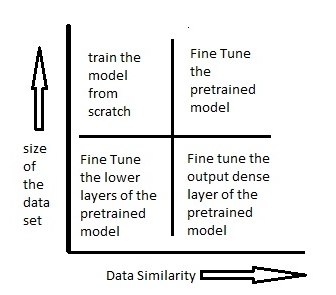
Fine-tuning:

* Fine-tuning là việc train một transfered model nhằm mục đích tối ưu hóa accuracy của model này trên target tasks. Fine-tuning giúp điều chỉnh weights của transfered model cho phù hợp với target tasks.
* Fine-tuning ra đời để giúp chọn chiến lược train hiệu quả trên transfered model của mình. Đưa ra cách(chiến lược) tối ưu để train cả phần pretrained model và phần mới trong transfered model nhằm đạt được accuracy cao trên target tasks.

Có một số cách fine-tuning:

* Đóng băng toàn bộ phần ở trước trước và thay thế các layer ở phía sau (dense output layer) sao cho phù hợp với yêu cầu của mình. Sau đó train lại model chỉ với phần được thêm vào.
* Train lại toàn bộ transfer model gồm cả phần Pretrained model và layer được thay thế
* ...

Tùy vào độ khác nhau của dataset và target task hiện tại với dataset và source task của Pretrained model và độ lớn của dataset hiện tại mà ta quyết định chọn chiến lược train phù hợp.



Hình III.C.1: Hình minh họa phân loại chiến lược Fine-tuning

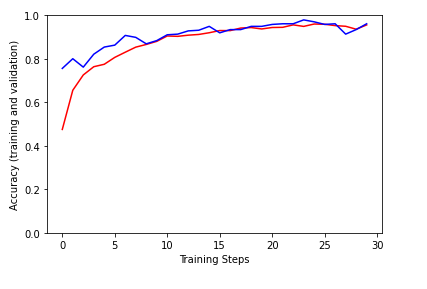
Lợi ích của Transfer Learning:

* Giảm thời gian tạo ra Model mới để thực hiện một Target task dựa trên source task.
* Giảm thời gian train vì ta tận dụng các Weight của pretrained model.

Hạn chế của Transfer Learning:

* Nếu sai sót trong quá trình thay thế, thêm/bớt các Layer sẽ dẫn đến Accuracy rất thấp
  1. Train model

Ta sẽ chọn số lượng epoch để train là 30 và dùng optimize function là Adam(thuộc họ Gradient descent, là bản nâng cấp hơn Gradient descent). Khi tăng số lượng epoch lên, model sẽ chuyển từ Underfitting->Optimal->Overfitting, vì vậy ta cần chọn epoch ở giai đoạn Optimal.



Hình III.D.1: biểu đồ accuracy của model qua từng epoch

Qua quan sát trên hình, nhóm chọn được hai model thuộc hai epoch lần lượt là 14 và 24. Chúng ta chọn hai model thuộc hai epoch trên vì dựa vào hình thì Accuracy trên tập Validate phía sau nó bắt đầu đi xuống.

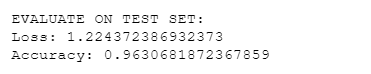
* 1. Test model

Chọn model từ epoch thích hợp sau đó test model đó trên tập test.

Vì quá trình train ta đánh giá model dựa trên tập train và tập validate nên khi test model trên tập test sẽ loại trừ được trường hợp overfitting trên tập test.

Kết quả trên tập test sẽ phản ánh đúng được khả năng dự đoán của model.

Ở đây dựa trên hình III.D.1, nhóm chọn được hai model ở hai epoch là 14 và 24, khi tiến hành train trên tập test thì kết quả của model thuộc epoch 14 nhỉnh hơn nên nhóm chọn model ở epoch 14 làm model chính thức để sử dụng.



Hình III.E.1: Accuracy trên tập test của model thuộc epoch 14

* 1. Deploy model lên điện thoại

Để deploy lên điện thoại thì ta cần sử dụng một model có tốc độ xử lý nhanh và có dung lượng thấp nên Mobilenet là model được triển khai thông dụng.

Dùng hàm trong Tensow Flow để convert model về dạng Tensow Flow Lite có thể deploy trên điện thoại.

# KẾT QUẢ

|  |  |  |  |
| --- | --- | --- | --- |
|  | InceptionNet | Inception\_ResNet | mobilenet\_v3\_small\_100\_224 |
| accuracy | 0.74 | 0.87 | 0.96 |

Hình IV.1: accuracy của các model đã train

Vì accuracy của model mobilenet\_v3\_small\_100\_224 là cao nhất và model này có kích thước nhỏ nên thích hợp để chạy trên điện thoại. Vì vậy nhóm đã sử dụng model mobilenet\_v3\_small\_100\_224 để chạy trên ứng dụng Android và cho kết quả nhận dạng khá tốt.

# PHÂN CÔNG NHIỆM VỤ

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Công việc | Người thực hiện | Mức độ hoàn thành |
| 1 | Lên kế hoạch | Phan Gia Huy  Nguyễn Quyết Thắng | 100% |
| 2 | Tìm kiếm và nghiên cứu tài liệu | Phan Gia Huy  Nguyễn Quyết Thắng | 100% |
| 3 | Viết code chương trình | Phan Gia Huy  Nguyễn Quyết Thắng | 100% |
| 4 | Báo cáo giấy và slide | Phan Gia Huy  Nguyễn Quyết Thắng | 100% |

# TÀI LIỆU THAM KHẢO

<https://arxiv.org/pdf/1704.04861.pdf>

https://arxiv.org/pdf/1801.04381.pdf

<https://arxiv.org/pdf/1905.02244.pdf>

<https://dlapplications.github.io/2018-07-15-Transfer-Learning-Basic/>

# LINK GITHUB

https://github.com/TieuLang/Do\_An\_Nhan\_Dang

1. **LINK DEMO**

https://drive.google.com/drive/folders/1keqzEmEcOi09SmY0i8OraJPsinFDjxte?usp=sharing