**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

**TRƯƠNG PHÚC ANH**

**KHÓA LUẬN TỐT NGHIỆP**

# PHÂN LỚP VẬT LIỆU SỬ DỤNG MẠNG HỌC SÂU

**CỬ NHÂN NGÀNH KHOA HỌC MÁY TÍNH**

**TP. HỒ CHÍ MINH, 2018**

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

**TRƯƠNG PHÚC ANH – 14520040**

# KHÓA LUẬN TỐT NGHIỆP

# PHÂN LỚP VẬT LIỆU SỬ DỤNG MẠNG HỌC SÂU

**CỬ NHÂN NGÀNH KHOA HỌC MÁY TÍNH**

**GIẢNG VIÊN HƯỚNG DẪN**

**TS. LÊ ĐÌNH DUY**

**Ths. MAI TIẾN DŨNG**

**TP. HỒ CHÍ MINH, 2018**

# DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số …… ngày …….. của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. …… – Chủ tịch.
2. …… – Thư ký.
3. …… – Ủy viên.
4. …… – Ủy viên.

# MỤC LỤC

# DANH MỤC HÌNH VẼ

# DANH MỤC BẢNG

# DANH MỤC TỪ VIẾT TẮT

SVM: Support Vector Machine

CNN: Convolutional Neuron Network

GTOS: Ground Terrain in Outdoor Scenes

FMD: Flicker Material Dataset

# LỜI CẢM ƠN

Sau quá trình học tập và rèn luyện tại trường Đại học Công Nghệ Thông Tin, Khoa Khoa Học Máy Tính và 4 tháng thực hiện đề tài nghiên cứu này, em xin tỏ lòng cảm ơn chân thành đến các thầy, cô giảng viên, cán bộ các phòng, ban chức năng tại trường đã giúp đỡ em hoàn thành luận văn tốt nghiệp này.

Đặc biệt, em chân thành cảm ơn thầy Lê Đình Duy và thầy Mai Tiến Dũng đã chỉ bảo, hướng dẫn và giúp đỡ em rất nhiều trong suốt quá trình thực hiện đề tài. Một lần nữa em chân thành cảm ơn và chúc các thầy dồi dào sức khoẻ.

Em cũng xin cảm ơn tất cả các bạn bè, anh chị đang học tập và làm việc tại trường, đặc biệt là MMLab đã nhiệt tình giúp đỡ em trong suốt thời gian qua.

Tuy nhiên vì kiến thức chuyên môn còn hạn chế và bản thân còn thiếu nhiều kinh nghiệm thực tiễn nên nội dung của báo cáo không tránh khỏi những thiếu xót, em rất mong nhận sự góp ý, chỉ bảo thêm của quý thầy cô để báo cáo này được hoàn thiện hơn.

# TÓM TẮT KHÓA LUẬN

Luận văn trình bày quá trình nghiên cứu, thực nghiệm và áp dụng các kiến thức trong lĩnh vực máy học, thị giác máy tính để xấy dựng một mô hình nhằm phát triển và cải thiện kết quả bài toán phân lớp bề mặt vật liệu trên những tập dữ liệu mới nhất. Sau quá trình nghiên cứu, nhóm đề xuất và hiện thực một số mô hình kết hợp deep feature và các hancrafted feature thích hợp để huấn luyện các bộ phân lớp theo nhiều cách khác nhau nhằm tăng độ chính xác cho bài toán. Mô hình đề xuất được kế thừa từ sự thành công của các phương pháp sử dụng mạng học sâu, kết hợp thêm các thông tin về hình dạng và texture của ảnh để giải quyết các trường hợp đối tượng có hình dạng hay texture giống nhau nhưng làm từ những vật liệu khác nhau. Một trong các mô hình mà nhóm hiện thực đạt kết quả 82.2% trên bộ dữ liệu Ground Terrain in Outdoor Scenes (so với 81.4% của nghiên cứu [1]) và 72.2% (so với 65.5% và 69.6% của nghiên cứu [2] và [3]) trên bộ dữ liệu Flicker Material Dataset. Sau cùng nhóm phân tích, đánh giá và giải thích kết quả. Ngoài ra, nhóm còn tiếp tục thực hiện các thực nghiệm để cải thiện hiệu suất tính toán, giảm thiểu chi phí và tăng khả năng thích ứng của các bộ phân lớp đối với dữ liệu mới bằng cách thiết kế một mạng convolutional neural network duy nhất. Cuối cùng nhóm đề ra những hướng phát triển tiếp theo của mô hình.

# Chương 1. **MỞ ĐẦU**

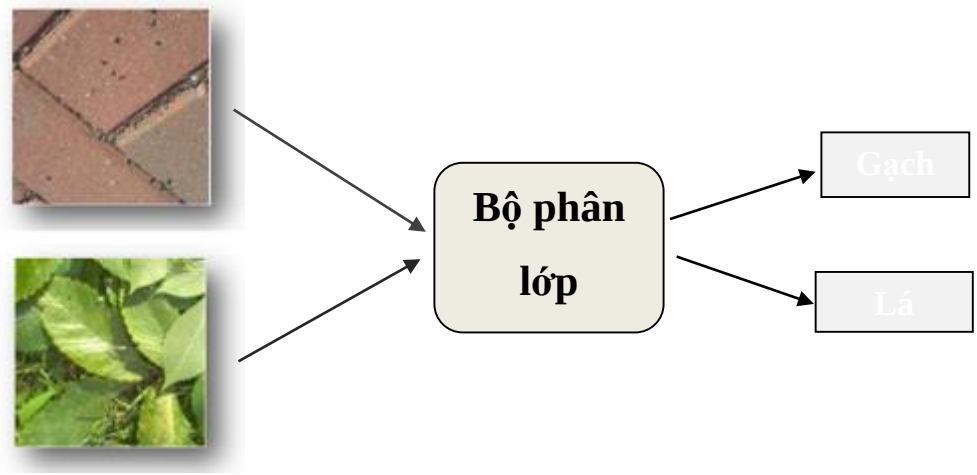
## 1.1. Giới thiệu bài toán

Bài toán mà nhóm thực hiện là “Phân lớp vật liệu” với mục tiêu chính là cung cấp thông tin vật liệu của một đối tượng hoặc một bề mặt trong ảnh. Cho trước một ảnh I, máy tính cần trả lời câu hỏi "Đối tượng (hay bề mặt này) được làm từ vật liệu gì?" chẳng hạn như gỗ, giấy, đá hay kim loại, ...

**Đầu vào**: Một ảnh I chứa hình ảnh của vật liệu cần được phân loại

**Đầu ra:** Tên của vật liệu trong ảnh I, thuộc một trong các lớp vật liệu cho trước.

Hình 1.1 là một ví dụ minh họa cho bài toán.




Hình 1.1: Phân lớp vật liệu cho biết bề mặt trong ảnh thuộc vật liệu nào

## 1.2. Tại sao cần phân lớp vật liệu?

Vật liệu của một bề mặt hay đối tượng nào đó là một thông tin rất giá trị để máy tính của thể hiểu được các thuộc tính của nó và từ đó có thể đưa ra các quyết định liên quan cũng như tương tác với chúng.

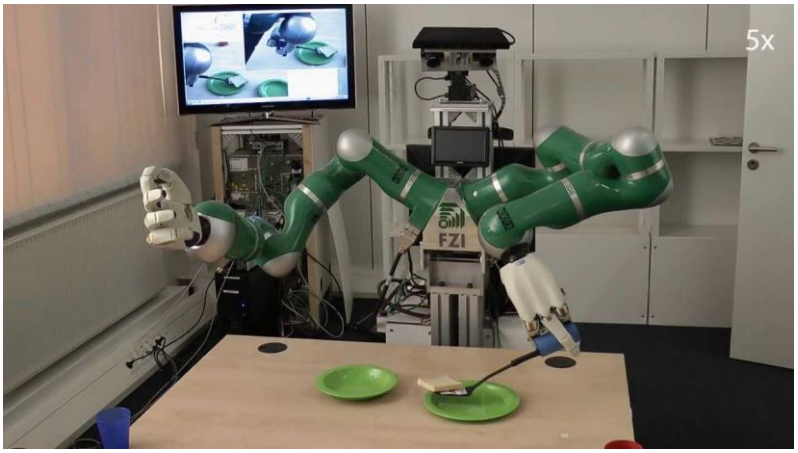
Hình 1.2 và Hình 1.3 là hai ví dụ đơn giản cho thấy máy tính có thể làm được rất nhiều việc khi có thể biết được thông tin về vật liệu. Trong Hình 1.2, với thông tin về vật liệu của các chai nước này, đơn giản nhất máy tính có thể sắp xếp chúng theo cân nặng, ngoài ra còn có thể quyết định chai nào có thể được dùng để đựng nước nóng chai nào không thể hay thậm chí chai nào có thể sử dụng để gây sát thương cho người khác (chai thủy tinh). Hình 1.4 minh họa cho một tình huống mà một chiếc xe tự lái không biết bề mặt phía trước là nước và vẫn lao thẳng tới.

Hình 1.2: Ba chai đựng nước với hình dáng tương tự nhau được làm từ những vật liệu khác nhau - quyết định những tính chất vật lý khác nhau.

Hình 1.3: Xe tự lái: "Xin lỗi, tôi không biết đó là nước :("
(Nguồn: https://bussorah.wordpress.com/2016/05/18/an-excellent-jewish-joke/)

Bài toán phân lớp vật liệu có nhiều ứng dụng trong lĩnh vực Robot Navigation với mục đích chính là giúp robot đưa ra quyết định di chuyển khi gặp những bề mặt địa hình khác nhau (Hình 1.4. Thông tin về vật liệu của bề mặt giúp robot đưa ra các quyết định khi di chuyển trên các địa hình khác nhau). Ngoài ra chúng còn được ứng dụng trong Robot Manipulation, giúp robot phân biệt các loại vật liệu khác nhau từ đó đưa ra những cách thao tác, xử lý với đối tượng khác nhau (Hình 1.5) [4].

Hình 1.4. Thông tin về vật liệu của bề mặt giúp robot đưa ra các quyết định khi di chuyển trên các địa hình khác nhau. [5]




Hình 1.5. Thông tin về vật liệu của các đối tượng giúp robot một phần trong việc đưa ra các quyết định để thao tác với chúng [4]

## 1.3. Các thách thức của bài toán

Nhiều thách thức khác nhau kết hợp làm bài toán phân lớp vật liệu rất khó giải quyết triệt để. Sự đa dạng về hình dáng, kích thước và texture là một trong số đó, có rất nhiều đối tượng có hình dáng khác nhau nhưng lại cùng loại vật liệu, ngược lại có những đối tượng trông có vẽ rất giống nhau nhưng lại làm từ những vật liệu khác nhau. Hình 1.6 là ví dụ minh họa lấy từ tập dữ Flicker Material Dataset với các đối tượng có texture giống nhau (sọc caro) nhưng được làm từ những vật liệu khác nhau.

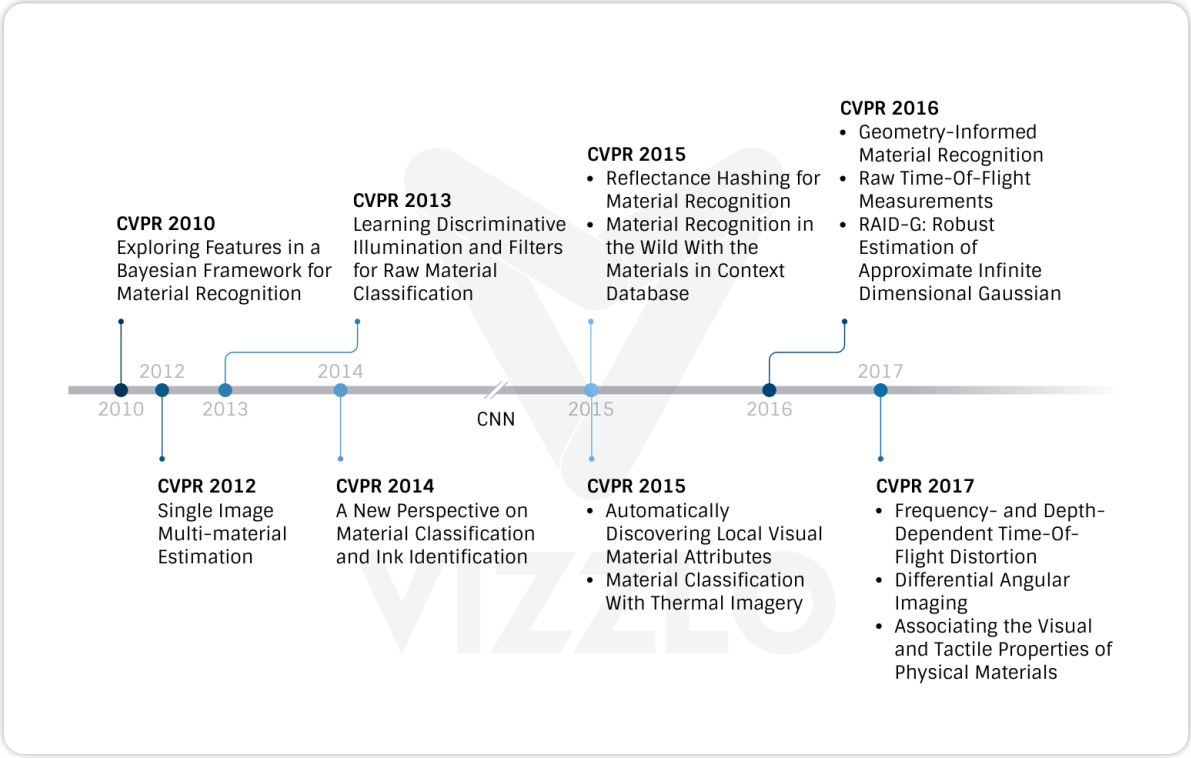

Hình 1.6: Những đối tượng với chất liệu khác nhau nhưng bề mặt lại có texture giống nhau (đều là sọc ca-rô). (Các ảnh được lấy từ Flicker Material Dataset [6])

Ngoài ra, điều kiện chiếu sáng khác nhau cũng khiến việc phân biệt giữa các vật liệu trở nên rất khó khăn (đặc biết đối với việc chỉ dùng một ảnh màu để phân biệt). Bên cạnh đó, sự chồng lắp giữa các đối tượng với nhau, giữa đối tượng với background cũng là một thách thức không nhỏ. Hình 1.7 là ví dụ lấy từ tập dữ liệu Open-Surface cho thấy sự thiếu hụt thông tin khi chỉ sử dụng ảnh màu để phân loại các vật liệu này.


Hình 1.7: Một ví dụ từ tập dữ liệu Open-Surface cho thấy sự thiếu hụt thông tin khi chỉ sử dụng ảnh màu để phân loại vật liệu [7]

# Chương 2. **CÁC CÔNG TRÌNH LIÊN QUAN**

Từ khi ra đời, bài toán phân lớp vật liệu nhận được sự chú ý của rất nhiều nhóm nghiên cứu. Cùng với sự phát triển đó, rất nhiều tập dữ liệu mới cũng được xây dựng để giải quyết nhiều vấn đề khác nhau của bài toán. Hình 2.1 là tóm tắt các công trình nghiên cứu liên quan được công bố tại hội nghị CVPR (Conference on Computer Vision and Pattern Recognition) từ năm 2010 tới nay. Bên cạnh đó Bảng 2.1 là tổng hợp một số tập dữ liệu được sử dụng.




Hình 2.1: Các nghiên cứu được công bố tại hội nghị CVPR qua các năm từ 2010 đến nay

| Name | Samples | Classes | Views | Illumination | In scene | Scene image | Camera parameters | Year |
|---|---|---|---|---|---|---|---|---|
| CUReT | 61 | 61 | 205 | 205 | No | No | No | 1999 |
| KTH-TIPS | 11 | 11 | 57 | 3 | No | No | No | 2004 |
| UBO2014 | 84 | 7 | 151 | 151 | No | No | No | 2014 |
| Reflectence disk | 190 | 19 | 3 | 3 | No | No | No | 2015 |
| 4D Light-field | 1200 | 12 | 1 | 1 | Yes | No | No | 2016 |
| NISAR | 100 | 100 | 9 | 12 | No | No | No | 2016 |
| GTOS | 606 | 40 | 19 | 4 | Yes | Yes | Yes | 2016 |

Bảng 2.1: Một số tập dữ liệu được dùng cho phân lớp dữ liệu [1]

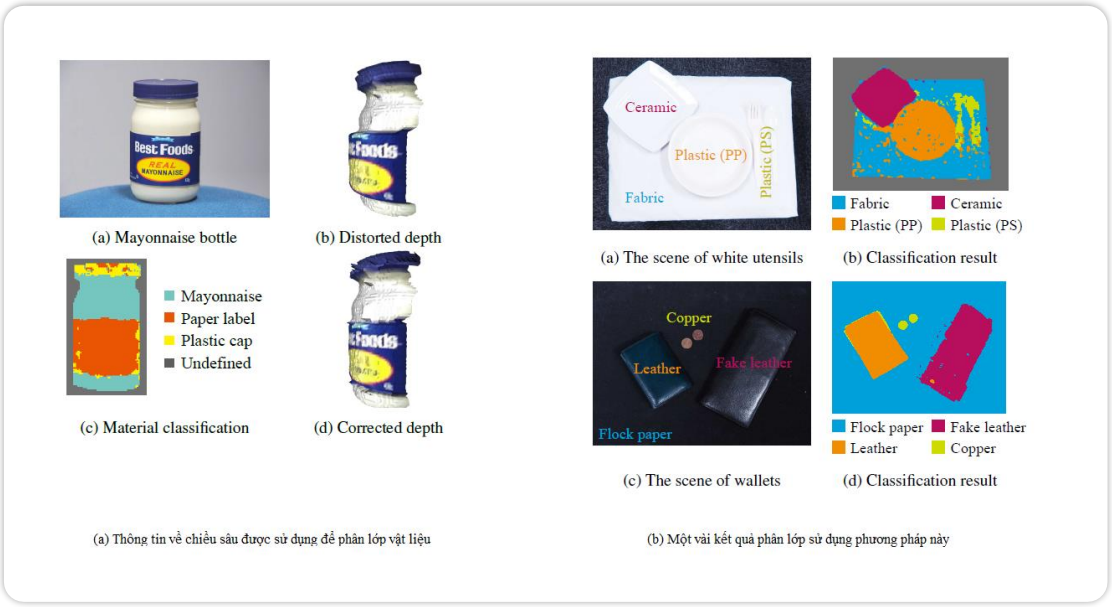
Một điểm chung sau khi CNN bắt đầu được dùng nhiều trong Thị giác máy tính và chứng minh sự hiểu quả của chúng, số lượng và chất lượng của các nghiên cứu cũng tăng lên đáng kể (kết quả phân lớp rất chính xác, gần như bằng với khả năng đoán của con người), các tập dữ liệu được dùng cũng lớn hơn, đa dạng hơn.

Có rất nhiều phương pháp khác nhau để giải quyết bài toán phân lớp vật liệu. Các phương pháp này có thể được chia thành hai nhóm chính: các phương pháp có sử dụng thông tin vật lý (lab-based methods) và các phương pháp chỉ sử dụng ảnh màu (image-based methods).

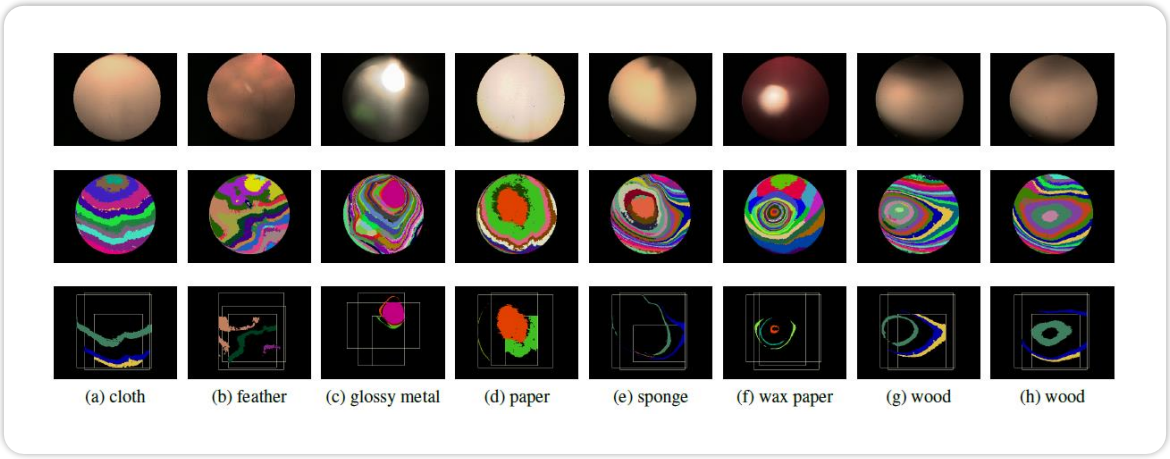
## 2.1. Nhóm các phương pháp có sử dụng thông tin vật lý (lab-based methods)

Đây là các phương pháp sử dụng chính các thông tin vật lý của đối tượng để phân lớp. Các thông tin vật lý có thể là độ đàn hồi (elasticity) [7], sự thấm nước (water permeation) [8], phản ứng với ánh sáng (optical response) hay độ phản chiếu (reflectance) [9]. Đây là các thông tin rất giá trị, vì vậy độ chính xác của các phương

pháp này thường rất cao. Tuy nhiên quá trình xây dựng các bộ dữ liệu để thực hiện các phương pháp này rất kỳ công, đòi hỏi các thiết bị đặc biệt và một môi trường lý tưởng (vì vậy nên chúng có tên lab-based). Ví dụ, GelFabric [10] - tập dữ liệu ra đời năm 2016 chứa thông tin về chiều sâu và chỉ dùng để phân biệt các loại vải khác nhau; Ground Terrain in Outdoor Scenes (GTOS) [1] - năm 2016 dùng để lấy thông tin về các góc nhìn khác nhau cho cùng một mẫu trong dữ liệu; Reflectance Disk Database [2] - năm 2015 với thông tin về sự phản chiếu của bề mặt. Thêm vào đó, các phương pháp khác nhau trong nhóm này thường đòi hỏi các thông tin vật lý khác nhau, chính vì thế tập dữ liệu của phương pháp khác nhau thường không thể dùng lại được. Ví dụ, một phương pháp dùng độ đàn hồi làm feature chính để phân lớp thì không thể dùng một tập dữ liệu chỉ có thông tin về độ thấm nước được. Vì vậy, các phương pháp đạt kết quả rất cao trên tập dữ liệu này có thể dễ dàng thất bại trên tập dữ liệu khác (hoặc không thể sử dụng trên dữ liệu khác vì thiếu thông tin). Hình 2.2 và Hình 2.3 là hai ví dụ được lấy từ hai nghiên cứu năm 2017 và 2015 sử dụng các thông tin vật lý khác nhau (thông tin về chiều sâu, độ phản chiếu ánh sáng) để phân biệt các loại vật liệu. Cả hai đều đạt kết quả khá tốt trên những tập dữ liệu có đầy đủ thông tin trên [9] [2].




Hình 2.2: Thông tin về chiều sâu được dùng để phân lớp vật liệu trong một nghiên cứu 2017 [9]



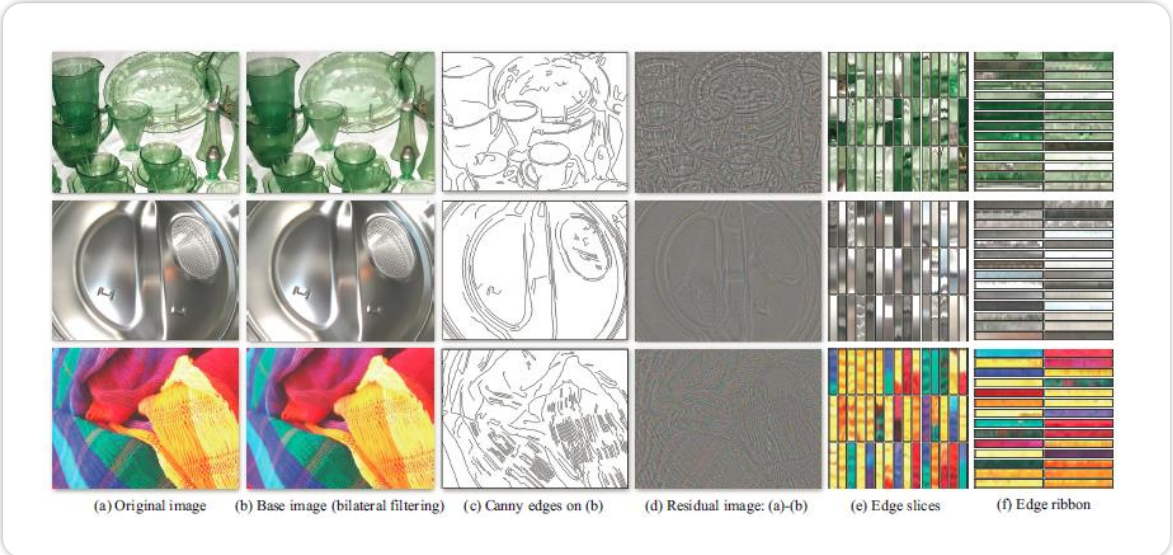

Hình 2.3: Thông tin về sự phản chiếu ánh sáng được dùng để phân lớp vật liệu trong một nghiên cứu 2015 [2]

## 2.2. Nhóm các phương pháp chỉ sử dụng ảnh màu (image-based methods)

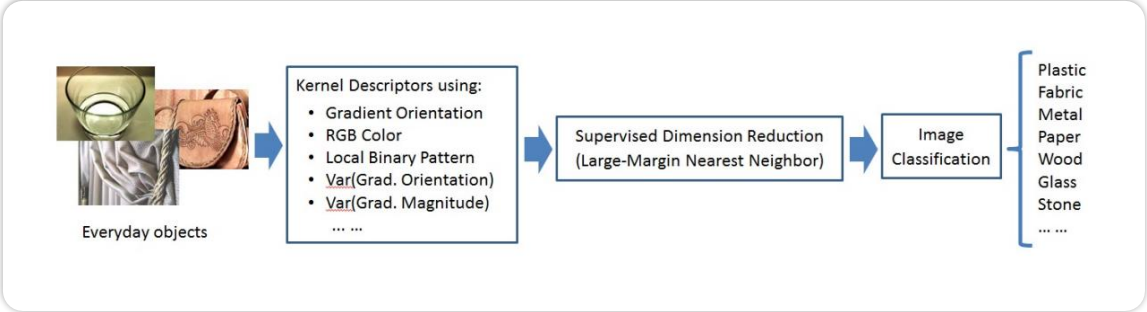
Ngược lại với các phương pháp Lab-base, Image-base là các phương pháp chỉ sử dụng ảnh màu RGB để phân lớp. Vì thế các tập dữ liệu cũng dễ dàng thu thập hơn, và các phương pháp này nếu có độ chính xác cao thì rất dễ để áp dụng vào thực tế.

Dựa trên các thông tin trực quan (chẳng hạn như màu sắc) thông qua ảnh và thường dựa trên bài toán phân loại đối tượng để tìm ra vật liệu của ảnh (vì các phương pháp hiện tại dùng nhiều các mạng CNN được huấn luyện sẵn cho bài toán phân loại đối tượng để làm gốc), vấn đề chính của các hệ thống dùng phương pháp này chính là sự thiếu thông tin và vì thế chúng dễ dàng bị "lừa" bởi các ảnh có các đối tượng tương tự nhau nhưng lại khác nhau về chất liệu. Hình 2.4 và Hình 2.5 là hai ví dụ cho các phương pháp thuộc nhóm này được lấy từ các nghiên cứu năm 2010 [11] và 2011 [12].

Trong những năm gần đây với sự phát triển của Deep Learning, các mạng CNN (Convolutional Neuron Network) được dùng thay thế cho các dạng local features này và nhiều nghiên cứu khác sử dụng các mạng CNN đã huấn luyện cho bài toán phân lớp đối tượng và đạt kết quả khá cao, tuy nhiên vẫn còn một số hạn chế như đã nêu bên trên. Nhóm tập trung giải quyết các hạn chế này và kế thừa các thành công từ CNN để cải thiện kết quả của bài toán.

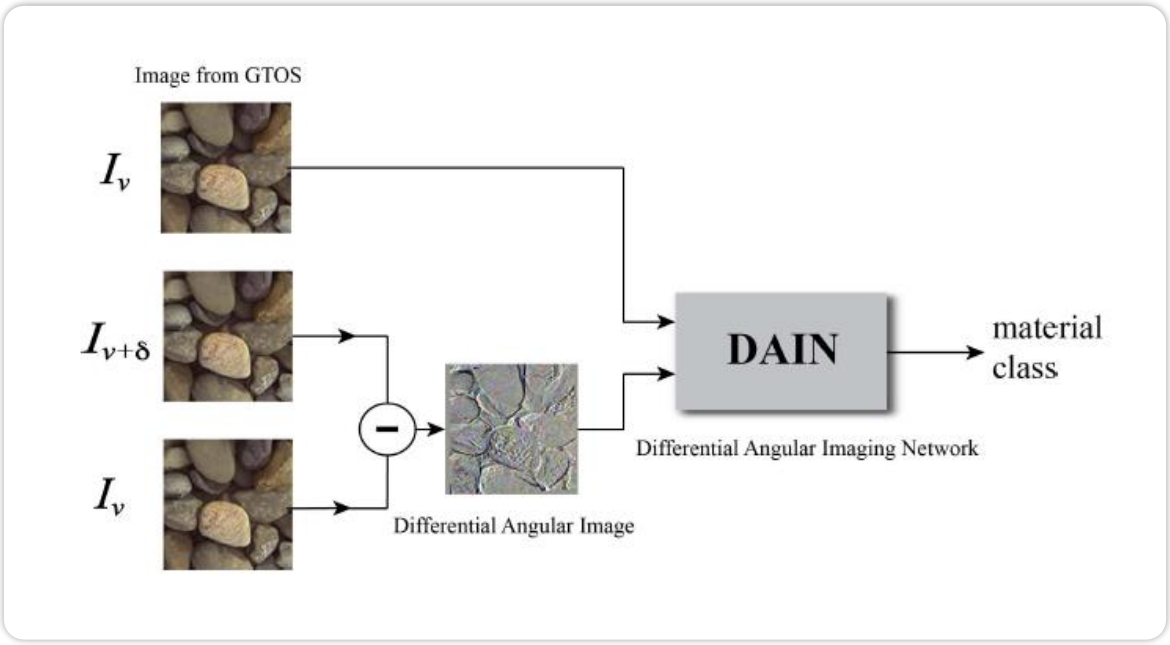



Hình 2.4: Các local features (cạnh, SIFT - Scale-Invariant Feature Transform) được được dùng để phân lớp vật liệu trong một nghiên cứu 2010 [11]

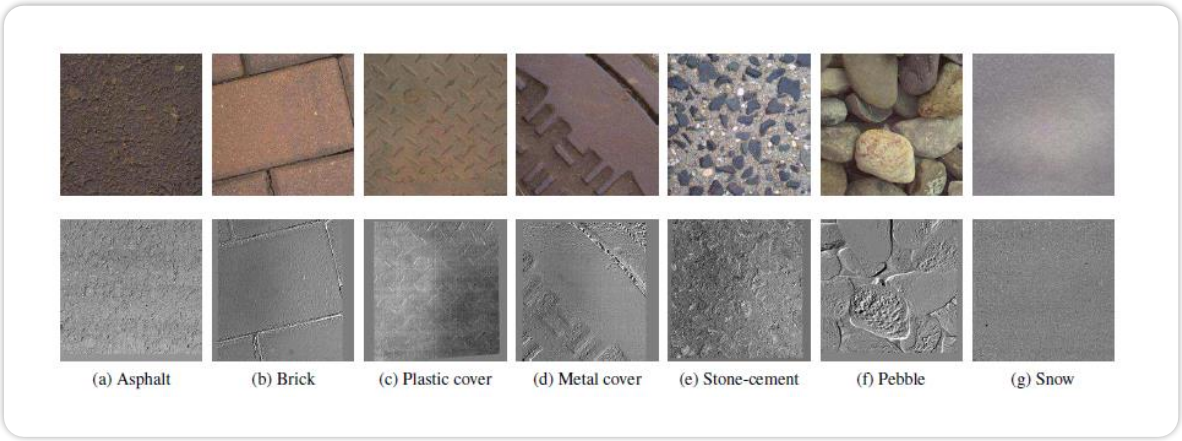



Hình 2.5: Thông tin về gradient, màu sắc, local binary patterns được được dùng để phân lớp vật liệu trong một nghiên cứu 2011 [12]

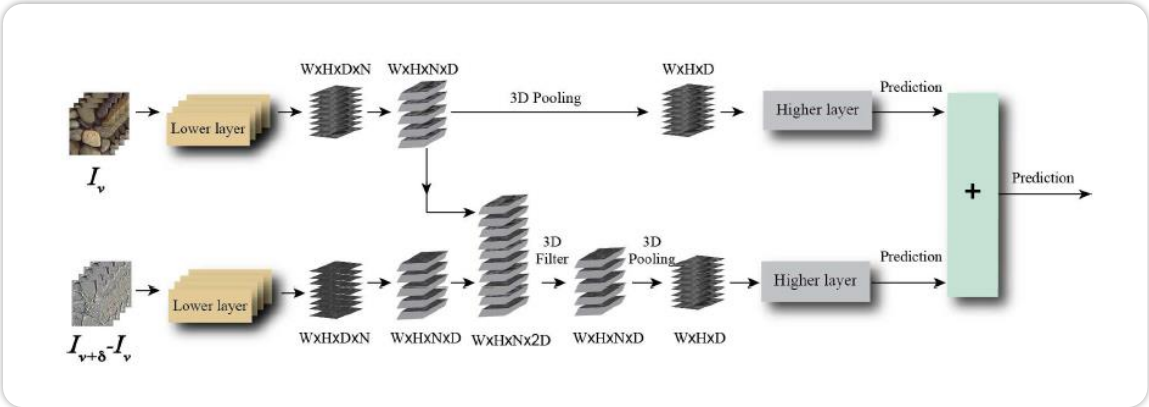
Trong đề tài này, phương pháp mà nhóm sử dụng thuộc loại Image-based vì đối với lab-based, các tập dữ liệu sẽ được xây dựng dựa trên phương pháp (đã có phương pháp X và xây dựng dữ liệu để phục vụ cho phương pháp X), cho nên các thông tin vật lý này dường như đã được sử dụng một cách triệt để nhất và nhóm sẽ có ít cơ hội hơn để cải tiến chúng. Ngoài ra, nhóm còn được thúc đẩy bởi một two-stream network (mạng với 2 nhánh chính) được dùng trên tập GTOS năm 2017 [1]. Hình Hình 2.6 thể hiện ý tưởng chính của nghiên cứu này, sử dụng một hệ thống mạng CNN với hai nhánh đầu vào (một là ảnh gốc, hai là ảnh thể hiện thông tin khác nhau về góc nhìn (view-point) - hay được tác giả gọi là Differential Angular Image). Differential Angular Image được tạo thành bằng cách lấy hai ảnh của cùng một mẫu dữ liệu được chụp từ hai góc khác nhau (hai góc này chênh lệch không quá nhiều) trừ nhau (Hình 2.7).




Hình 2.6: Một mạng CNN với hai nhánh đầu vào (two-stream network) được dùng trên tập GTOS năm 2017 [1]




Hình 2.7: Differential Angular Image được tạo thành bằng cách lấy hai ảnh của cùng một mẫu dữ liệu được chụp từ hai góc khác nhau (hai góc này chênh lệch không quá nhiều) trừ nhau [1]



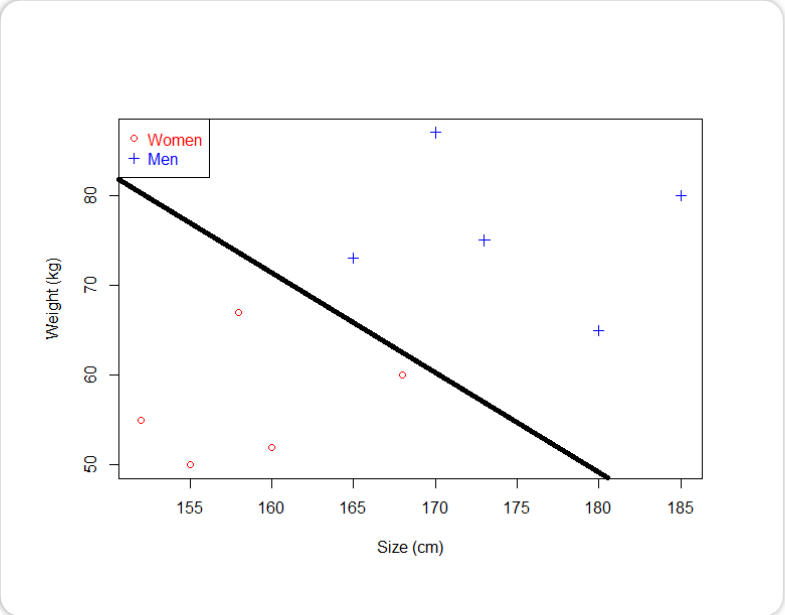

Hình 2.8: Cấu trúc bên trong của DAIN (Differential Angular Image Network) [1]

# Chương 3. CƠ SỞ LÝ THUYẾT

Để xây dựng các bộ phân lớp cho bài toán, nhóm sử dụng thuật toán Support Vector Machine, kế thừa thành công từ các mạng Convolutional Neutron Network cho bài toán phân lớp đối tượng để sử dụng cho bài toán phân lớp vật liệu bằng cách kết hợp với các Handcrafted feature thể hiện thông tin về hình dạng và texture của ảnh. Chương 3 trình bày cơ sở lý thuyết của các kỹ thuật trên.

## 3.1. Support Vector Machines (SVMs)

Support Vector Machine là một trong những thuật toán học máy có giám sát (supervised learning) hiệu quả nhất. Được xây dựng dựa trên một nền tảng toán học vững chắc, SVM thường cho kết quả tốt trong nhiều bài toán mà không cần điều chỉnh quá nhiều. Tuy nhiên, cũng chính vì điều này mà nó thường được xem như một hộp đen (black box). Có nhiều thuật toán SVM khác nhau nên chúng thường được gọi chung là SVMs. Ở dạng chuẩn, SVM là thuật toán phân lớp nhị phân (binary classification). Từ dữ liệu huấn luyện, SVM xây dựng (learn) một siêu phẳng (hyperplane) để phân lớp (classify) tập dữ liệu thành hai lớp riêng biệt (Hình 3.1).




Hình 3.1: SVMs được dùng để phân lớp tập dữ liệu thành hai lớp phân biệt

SVMs là kết quả của một quá trình nghiên cứu xuyên suốt nhiều năm của rất nhiều người. Thuật toán SVM đầu tiên thuộc về Vladimir Vapnik năm 1963. Sau này, ông làm việc với Alexey Chervonenkis về cái được gọi là lý thuyết VC (VC theory - Vapnik–Chervonenkis theory), giải thích quá trình học từ góc nhìn của thống kê, và cả hai đều đóng góp rất nhiều cho SVM. Lịch sử chi tiết về SVM có thể được tìm thấy ở đây (http://www.svms.org/history.html).

Trên thực tế, SVMs đã chứng minh được sự hiệu quả của mình trong rất nhiều lĩnh vực khác nhau: phân loại văn bản (text categorization), nhận diện ảnh (image recognition) và bioinformatics (Cristianini & ShaweTaylor, 2000). Trong phần này, nhóm sẽ trình bày ý tưởng chính đằng sau SVM. Các khác niệm cơ bản được sử dụng trong SVM: vector, linear separability và hyperplanes sẽ được trình bày tại Phụ lục 1.

**Ý tưởng chính của SVM**

Với bài toán phân lớp, mục tiêu của SVM là tìm một hyperplane "tối ưu" nhất để chia các điểm dữ liệu thành hai phần (binary classification). Để so sánh giữa hai hyperplane để chọn cái tốt hơn, chúng ta cần một thước đo. Ở đây chúng ta có hai tiêu chí để chọn một hyperplane, một là hyperplane này phải nằm giữa phân tách hai lớp dữ liệu và hai là hyperplane phải cách xa điểm dữ liệu nằm gần nó nhất (hay chính là bài toán maximize margin) .

Cho tập dữ liệu $D = \{(x_i, y_{\_i}) | x_i \in \mathbb{R}, y_i \in \{1; -1\}\}$. Chúng ta cần tìm một hyperplane $wx + b = 0$ (hay chính là tìm w và b) sao cho M đạt giá trị lớn nhất với:

$$M = min_{i=1}^{m} \left\{ y_i \left( \frac{wx_i + b}{||w||} \right) \right\}$$

Đây chính là bài toán maximize khoảng cách từ điểm dữ liệu gần hyperplane nhất đến hyperplane (maximize margin). Lưu ý, $y_i \left( \frac{wx_i+b}{||w||} \right)$ sẽ cho kết quả dương nếu điểm dữ liệu được phân lớp đúng và ngược lại âm nếu nó bị phân lớp sai.

Với bài toán tối ưu (Optimization Problem) này, để có thể sử dụng các phương pháp có sẵn từ toán học để giải, trước tiên chúng ta cần biến đổi nó một chút để về dạng đơn giản hơn.

Bài toán chúng ta cần giải là:

$$Maximize\ M\ theo\ w\ và\ b$$

$$Sao\ cho: y_i\left(\frac{wx_i+b}{||w||}\right) \geq M, i = 1,\ldots,m$$

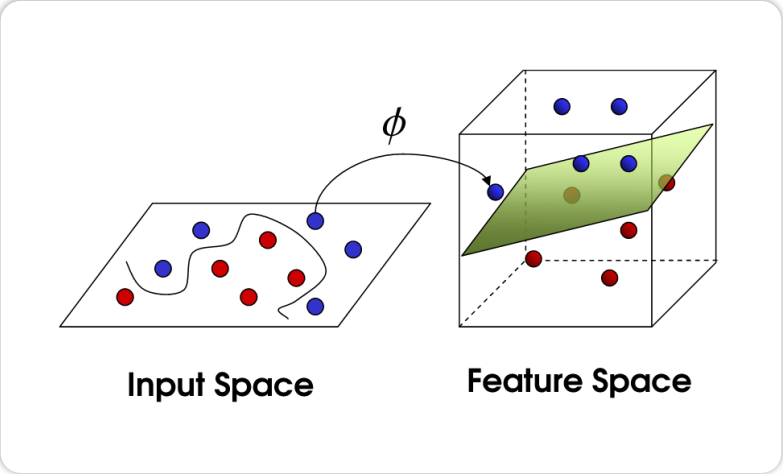
Đầu tiên vì ta có thể scale vector w và b một cách thích hợp mà không làm thay đổi phương trình hyperplane (ví dụ: hyperplane $5x_1 + x_2 + 1 = 0$ cũng có thể scale thành $10x_1 + 2x_2 + 2 = 0$), nên ta có thể đặt:

$F = min_{i=1}^{m}\{y_i(wx_i + b)\}$ mà không ảnh hưởng đến kết quả bài toán. Khi đó bài toán được viết lại:

$$Maximize\ \frac{1}{||w||}\ theo\ w\ và\ b$$

$$Sao\ cho: y_i(wx_i + b) \geq 1, i = 1,\ldots,m$$

Sau khi dùng các phương pháp toán học đã có sẵn để giải quyết bài toán tối ưu này, chúng ta sẽ tìm được hyperplane phân tách các điểm dữ liệu thành hai lớp, với dữ liệu mới ta chỉ cần thay vào phương trình của hyperplane và quyết định xem nó thuộc lớp nào.




Hình 3.2: Dữ liệu được ánh xạ lên không gian cao hơn trước khi các bộ phân lớp được học

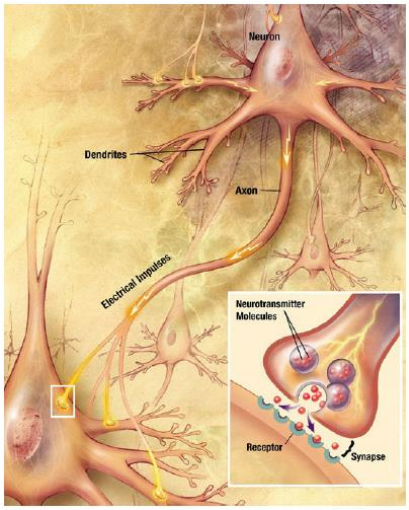
Tuy nhiên dữ liệu thực tế rất phức tạp và đan xen lẫn nhau khiến chúng ta không thể tìm được hyperplane nào có thể phân tách chúng trên không gian hiện tại, khi đó một phép ánh xạ sẽ được thực hiện để đưa các điểm dữ liệu này vào không gian với số chiều lớn hơn và hy vọng tìm được một hyperplane ở không gian này. Hình 3.2 là ví dụ cho phép ánh xạ này. Nếu chúng ta xem các chấm màu xanh và đỏ là những quả bong bóng, trong hình bên trái, các quả bóng này được đặt trên bàn, nếu chúng phân bố không quá đan xen vào nhau, ta có thể dùng một cây que dài để chia các quả bóng thành hai tập xanh và đỏ mà không động đến các quả bóng. Lúc này, khi đưa một quả bóng mới đặt lên mặt bàn, bằng cách xác định nó nằm bên phía nào của cây que, ta có thể dự đoán màu sắc của quả bóng đó. Các quả bóng tượng trưng cho các điểm dữ liệu, màu xanh và đỏ tượng trưng cho 2 lớp. Cái bàn tượng trưng cho một mặt phẳng. Cây que tượng trưng cho một siêu phẳng đơn giản đó là một đường thẳng. Tuy nhiên khi các quả bóng nằm đan xen nhau trên bàn (dữ liệu phức tạp hơn), lúc này không thể dùng cái que để phân tách chúng, để làm được điều này chúng ta cần hất các quả bóng bay lên (chính là phép ánh xạ), từ đó có thể sử dụng một tờ giấy để phân tách chúng (tờ giấy chính là siêu phẳng). Trên thực tế, SVMs thực hiện ánh xạ bằng việc

sử dụng các kernel thích hợp. Việc lựa chọn kernel với các tham số thích hợp sẽ giúp SVM học được các bộ phân lớp tốt hơn trên những tập dữ liệu khác nhau

Trong luận văn này, nhóm chọn SVM để huấn luyến các bộ phân lớp dựa trên các đặc trưng rút trích từ ảnh để phân lớp các loại vật liệu vì sự hiệu quả và cơ sở toán học chắc chắn của nó.

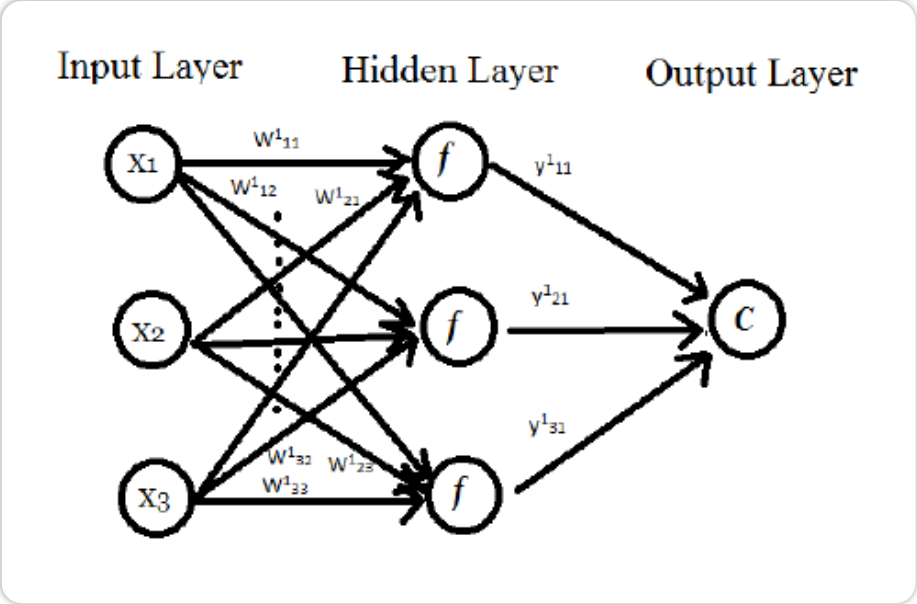
### 3.2. Neuron Network

Neural Network là một loại hệ thống máy tính đặc biệt được lấy cảm hứng từ cấu trúc não của con người. Một tế bào trong não người được gọi là nơron có hai thành phần chính: Axons và Dendrites. Những tế bào thần kinh này hoạt động bằng cách xử lý các tín hiệu điện. Dựa trên tín hiệu nhận được thông qua Dendrites, Axons thực hiện mã hóa thần kinh (Neural coding), sau đó được phân phối tín hiệu này qua các Neurons và có thể được coi là đầu ra của hệ thần kinh. Axon của một Neuron được kết nối với Dendrites của Neurons khác, cứ như vậy các Neurons liên kết với nhau và hình thành Neurons Network trong não người (Hình 3.3).




Hình 3.3: Tín hiệu được truyền từ Axons của một neuron đến Dendrites của neuron tiếp theo (Nguồn: https://www.tes.com/lessons/zMaG-i_CdPxTNg/neurons-and-synapses).

Với các tiến bộ trong khoa học máy tính, hiện nay chúng ta đã có thể phát triển hệ thống mạng hoạt dựa trên cách lan truyền tín hiệu của các nerouns trong hệ thần kinh. Thành phân cơ bản của hệ thống này là các nút (node) tương tác với nhau thông qua các trọng số và hàm lan truyền. Các node tương tự như các nerouns trong não. Một mạng neroun là sự kết hợp của nhiều layer của các node. Các layers được chia thành input layer, output layer và intermediate layer (còn được gọi là hidden layer). Đầu ra của mỗi node được tính toán thông qua hàm kích hoạt (activation function). Mỗi node gắn với mỗi giá trị trọng số (weight) và độ lệch (basis) riêng, hai giá trị. Trong nhiều trường hợp, mạng neroun là một hệ thống thích ứng (adaptive system) tự thay đổi cấu trúc của mình (các trọng số) dựa trên các thông tin bên ngoài hay bên trong đi qua mạng trong quá trình học.




Hình 3.4: Cấu trúc của một Feedforward Perceptron Neural Network đơn giản.

Hình Hình 3.4 là cấu trúc của một mạng neural đơn giản. Thuật ngữ "Feedforward" được dùng để chỉ cách dữ liệu đi qua mạng, trong trường hợp này dữ liệu chỉ đi qua mạng một chiều. Layer đầu tiên lấy các thông tin từ đầu vào, được gọi là input layer. Khi các giá trị đầu vào được truyền qua input layer, mỗi node sẽ tính toán giá trị output thông qua weights và biases, sau đó truyền output này tới hidden layer. Tại đây, các nodes dùng activation function để tính toán đầu ra rồi tiếp tục truyền tới layer cuối. Layer này được gọi là output layer, tính toán tất cả các "score" từ giá trị lan truyền từ layer trước. Với bài toán phân lớp, kết quả cuối cùng chính là lớp với giá trị "score" cao nhất sau khi qua output layer.

Giá trị đầu ra của mỗi node trong mạng được tính toán theo phương trình (3.7). Trong đó, $y_j$ là giá trị đầu ra của node thứ j của layer hiện tại, n là số lượng node của layer trước truyền thông tin vào node hiện tại, $f$ là hàm kích hoạt của node, $w_{mn}$ là trọng số được truyền từ node thứ m của layer trước đến node thứ n của layer hiện tại, $b_j$ là bias của node hiện tại (đang cần tính đầu ra)

$$y_j = f\left(\sum_{i=1}^{n}\left(w_{ij}.x_i\right) + b_j\right) \quad (3.7)$$

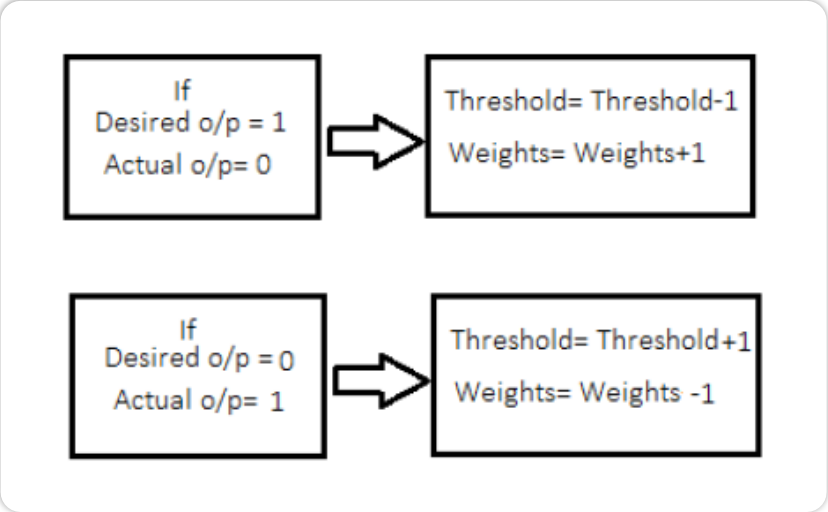
Node đầu ra C sẽ nhận giá trị của các $y_j$ sau đó chọn lớp có score lớn nhất là output cuối cùng. Phương trình (3.8), (3.9) và (3.10) là cách tính giá trị đầu ra của từng node trong hidden layer:

$$y_1 = f(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1) \quad (3.8)$$

$$y_2 = f(w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2) \quad (3.9)$$

$$y_3 = f(w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + b_3) \quad (3.10)$$

Một hệ thống mạng neural với khả năng học tập hạn chế đã được phát triển trong công trình của Lugar và Stubble [13] và được gọi là Perceptron. Mỗi node sẽ có đầu ra là 0 hoặc 1 được tính toán dựa trên một giá trị ngưỡng đặc biệt. Mạng neural này sẽ được học trên một tập dữ liệu huấn luyện. Mỗi mẫu trong tập huấn luyện gồm các giá trị đầu vào và giá trị đầu ra mong muốn tương ứng với nó. Các giá trị đầu vào được truyền vào input layer, đi qua mạng neural và cho một giá trị đầu ra (ở đây là 0 hoặc 1). Nếu giá trị đầu ra của mạng không giống với giá trị đầu ra mong muốn ban đầu thì giá trị các weights và thresholds sẽ được điều chỉnh để có kết quả tốt hơn. Hình 3.5 là một cách thay đổi weights và thresholds để có kết quả tốt hơn.




Hình 3.5: Một trong những cách đơn giản thay đổi weights và thresholds của một node để có kết quả tốt hơn

Perceptron sau đó được phát triển thành mạng với nhiều hidden layer (multilayer). Cấu trúc này được học dựa trên thuật toán lan truyền ngược (backpropagation) Thuật toán này tuân theo quy tắc delta để tính toán độ lỗi tại nút đầu ra. Độ lỗi của nút cuối cùng trong mạng (output node) được lan truyền ngược qua mạng sau mỗi lần huấn luyện, giá trị các weights sẽ được cập nhật dựa trên hàm lỗi. Cách học này còn được gọi là gradient descent.

Giá trị đầu ra của một node phụ thuộc vào hàm kích hoạt và trong mạng lan truyền ngược là một hàm sigmoid (Phương trình (3.11)) với:

$f_j = \sum_{i=1}^{n}(w_{ij}x_i)$.

$$\sigma(f_x) = \frac{1}{1+e^{-fx}} \tag{3.11}$$

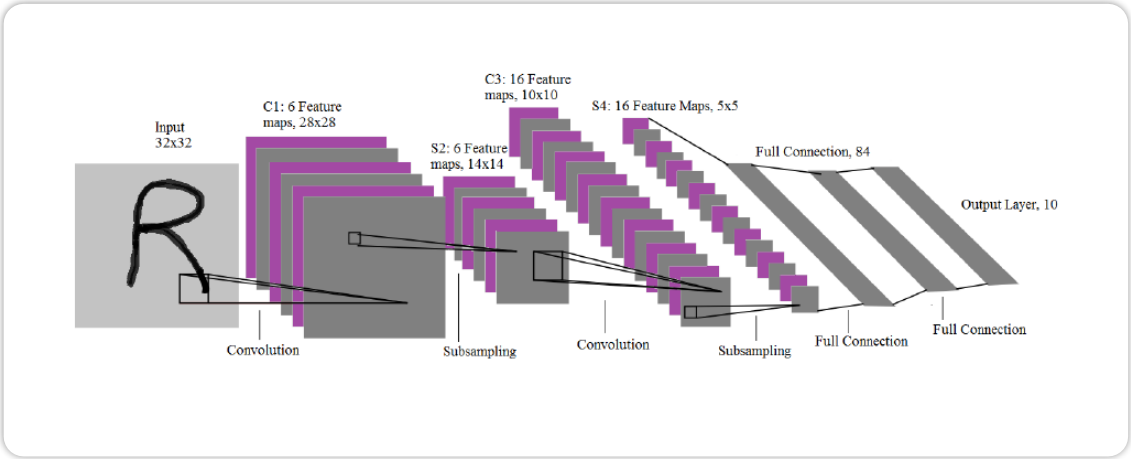
Hàm sigmoid áp dụng cho tất cả các node trừ các input node và giá trị đầu ra được giới hạn trong khoảng [0,1].

### 3.3. Convolutional Neuron Network

Convolutional Neural Networks (CNNs) là một loại Neural Network đặc biệt. Nguyên tắt hoạt động của CNNs tương tự như một Neural Network bình thường: lấy

input tại input layer, truyền giá trị này qua các hidden layer, dùng activation function tính toán giá trị đầu ra tại mỗi node. Score tại các node ở layer cuối (output layer) được lan truyền ngược và các trọng số của mạng được cập nhật sau mỗi lần huấn luyện. Trong thực tế, CNNs khó huấn luyến hơn các Neural Network bình thường do sự phức tạp của nó.Nhưng điều gì khiến CNNs đặc biệt hiệu quả trong các vấn đề của Thị Giác Máy Tính? Hãy cùng tìm hiểu bên dưới.

Các Neural Network bình thường không thích ứng tốt với các ảnh đầu vào khác nhau. Với ảnh đầu vào 32x32 RGB, input layer sẽ có 32*32*3 = 3072 weights. Nhưng trên thực tế, kích thước của ảnh đầu vào lớn hơn nhiều, cho một ảnh với kích thước 600x300 RGB, sẽ có tới 600*300*3 = 540000 weights chỉ tính input layer - một con số không hề nhỏ! Để khắc phục điều này, CNN nhận cả bức ảnh làm đầu vào. Năm 2012, CNNs bắt đầu nhận được sự chú ý của nhiều nhà nghiên cứu trong lĩnh vực phân lớp ảnh khi Alex Krizhevsky thắng giải trong cuộc thi của Imagenet bằng việc thiết kế một mạng CNN và giảm độ lỗi trong việc phân lớp đối tượng từ 26.2% xuống còn 15.3% [14]. CNN tận dụng lợi thế của bất kỳ cấu trúc đầu vào nào thể hiện mối tương quan không gian (chẳng hạn như hình ảnh) và sắp xếp các neural theo không gian này. Sự sắp xếp này cho phép thông tin đi qua mạng hiệu quả và giảm đáng kể số lượng tham số của mạng. Cấu trúc của một mạng CNN gồm nhiều phần khác nhau: Convolutional Layer, Pooling Layer, Rectied Linear Unit hoặc ReLU Layer và Fully Connected Layer. Một trong những kiến trúc CNN cơ bản (LeNet-5) dùng cho nhận diện chữ viết được thể hiện trong Hình 3.6

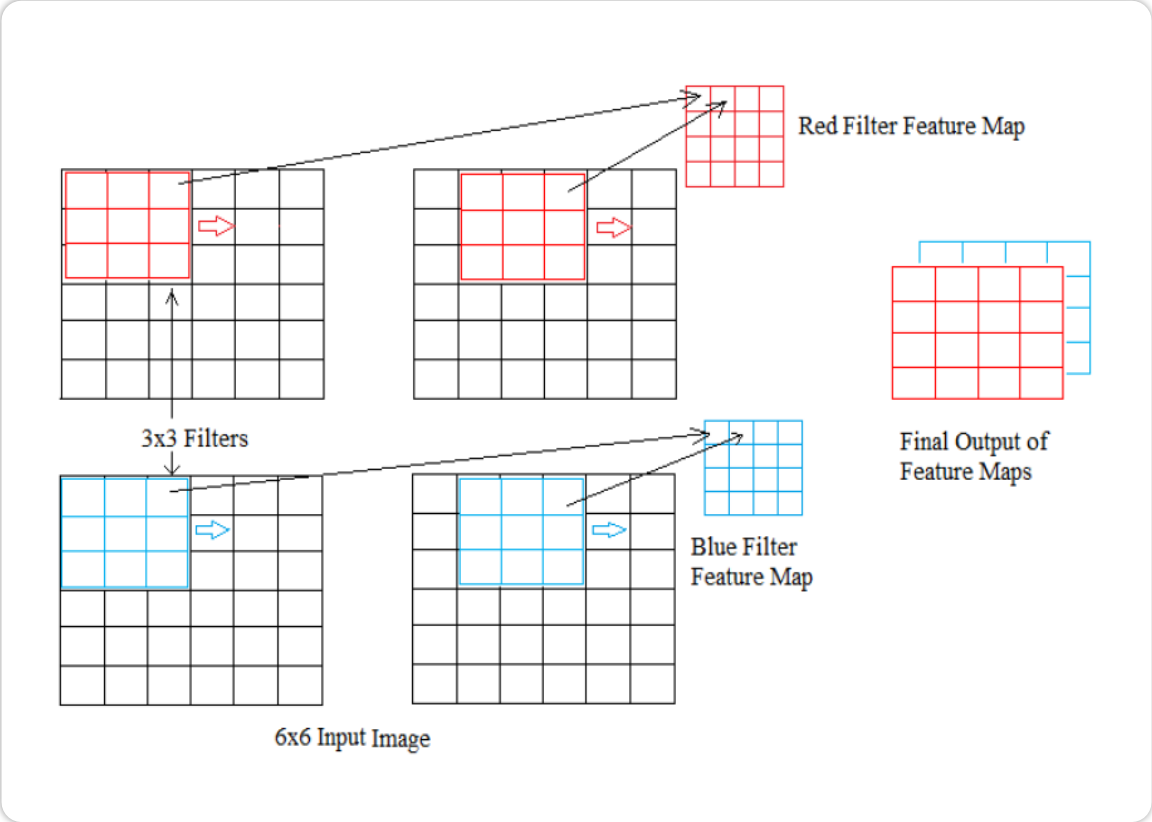



Hình 3.6: Cấu trúc mạng LeNet-5, một mạng CNN dùng cho nhận diện chữ viết [15]

**Convolutional layer**

Convolutional layer là thành phần chính thực hiện hầu hết các tính toán trong một mạng CNN, Mỗi convolutional layer chứa một tập các filters (hay kernels) với kích thước ứng với các weights và biases. Trong quá trình lan truyền của một mạng CNN, mỗi kernel trượt qua tất cả các vùng của ảnh đầu vào và thực hiện phép toán dot product trên mỗi vùng đó. Mỗi kernel rút trích các đặc trưng khác nhau từ ảnh. Các phép toán này làm giảm dần số chiều (kích thước) của đầu vào nhưng cũng làm tăng độ sâu của mạng.

Ba tham số quyết định đầu ra của một convolutional layer bao gồm: depth, stride và zero-padding. **Depth** thay đổi đầu ra của convolutional layer dựa vào số lượng kernel của layer đó. Số lượng kernel càng lớn thì đầu ra của layer sẽ càng "sâu". **Stride** là tham số thể hiện bước nhảy của kernel khi trượt trên ảnh để thực hiện dot product. Nếu stride có giá trị là 1 thì kernel se trượt một pixel sau mỗi phép dot product. Bằng cách này, stride sẽ quyết định kích thước của đầu ra. **Zero-padding** cũng ảnh hưởng đến kích thước của đầu ra thông qua việc thêm 0 vào các cạnh của ảnh hay ma trận đầu vào.

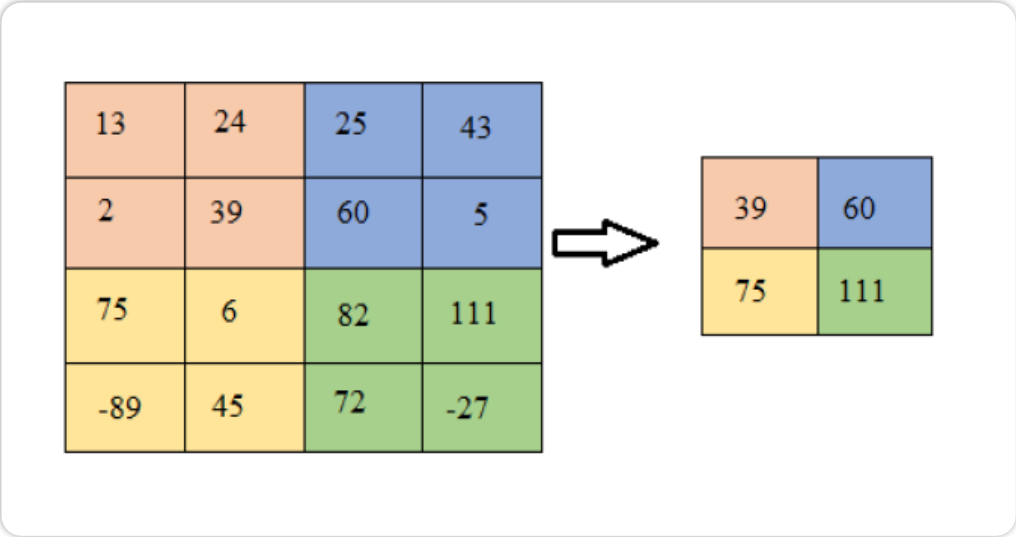



Hình 3.7: Hai filter 3x3 được dùng để tính toán hai 4x4 2D feature map từ một ma trận đầu vào 6x6

Hình 3.7 thể hiển quá trình tính toán của một convolutional layer sử dụng 2 filters 3x3 cho ma trận đầu vào 6x6 (không dùng zero-padding). Sau khi các phép toán dot product hoàn thành, kết quả đầu ra của layer này là hai 2D feature map 4x4 (hay có thể nói đầu ra có kích thước 4x4x2). Đây chính là ví dụ cho việc giảm kích thước đầu vào nhưng tăng chiều sâu của layer trong mạng CNN. Một cách tổng quát, với ma trận đầu vào kích thước NxN đi qua một convolutional layer với Y filters MxM (stride = 1 và không dùng zero-padding) thì đầu ra có kích thước (N - M + 1) X (N - M + 1) X Y

**Pooling Layer**

Là một loại layer quan trọng khác trong CNN, pooling layer thường được dùng ở giữa hai convolutional layers. Pooling layer được dùng để giảm kích thước của ma trận đầu vào (vì vấy nó còn có tên khác là downsampling layer). Kiểu pooling layer phổ biến nhất là max pooling layer, thường sử dụng với filter có kích thước 2x2 và stride = 2 pixels. Hình 3.8 cho thấy tác dụng của layer này, giảm kích thước ma trân đầu vào từ 4x4 xuống còn 2x2.



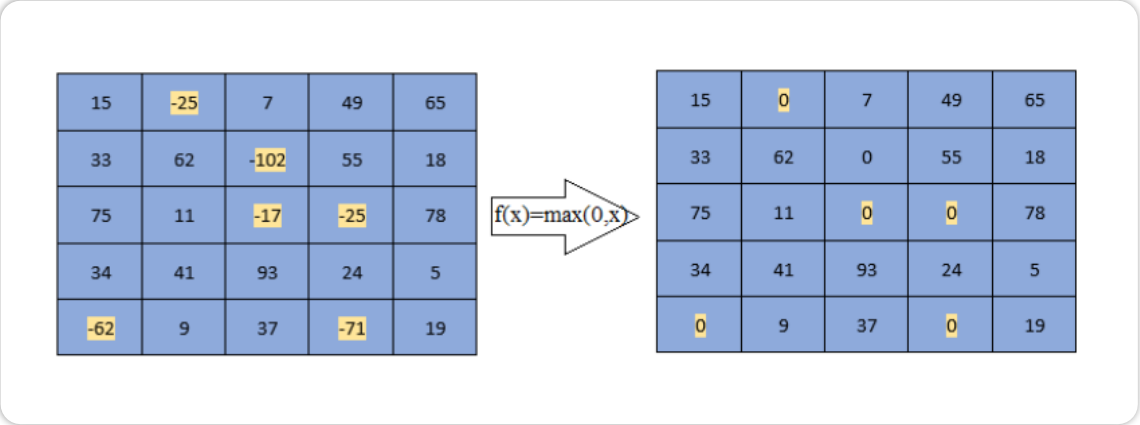

Hình 3.8: Maxpooling được dùng để giảm kích thước ma trận đầu vào

Những phương pháp gần đây thường không dùng pooling layer ở giữa hai convolutional layer. Thay vào đó, để đạt được tác dụng tương tự pooling layer (giảm kích thước), stride của convolutional layer thường được dùng với giá trị cao hơn .

**Rectied Linear Unit (ReLU) Layer**

Rectied Linear Unit hay ReLU layer thường được dùng sau convolutional layer và trước pooling layer. Thực nghiệm cho thấy layer này được dùng để tăng hiệu quả tính toán của mạng CNN [16].

ReLU layer không thay đổi kích thước của ma trận đầu vào. Trong ví dụ Hình 3.9, layer này sử dụng hàm $f(x) = max(0, x)$ để thay đổi tất cả các giá trị âm thành 0.




Hình 3.9: ReLU layer sử dụng hàm $f(x) = max(0, x)$ để thay đổi tất cả các giá trị âm thành 0.

**Fully Connected Layer**

Fully connected layer được dùng như layer cuối cùng của một mạng CNN (output layer). Tại đây, mỗi node trong layer trước liên kết với tất cả các node của layer tiếp theo, Layer này được dùng để chuyển đổi feature map từ layer trước thành classification score.

## 3.4. Transfer learning

Với bài toán phân lớp ảnh, để đạt kết quả tốt đòi hỏi một cấu trúc mạng thích hợp, điều này phụ thuộc nhiều vào tập dữ liệu huấn luyện. Trong thực tế, dữ liệu huấn luyện rất lớn và đa dạng. Một cấu trúc mạng tốt có thể học được khối lượng dữ liệu lớn thường rất phức tạp và đòi hỏi khá nhiều về phần cứng để thực hiện huấn luyện. Alexnet [14] sử dụng hơn 1.2 triệu ảnh thuộc 1000 lớp khác nhau để huấn luyện trên hai GTX 580 3GB GPUs trong gần một tuần.

Với các bài toán trên tập dữ liệu khác, kỹ thuật transfer learning được dùng để sử dụng lại các mạng phức tạp đã được huấn luyện sẵn này, và chỉ huấn luyện lại một phần nhỏ để thích hợp với bài toán mới. Cách tiếp cận này đã được được rất nhiều kết quả tốt trong nhiều bài toán phân lớp trên những tập dữ liệu khác nhau.

Một cách đơn giản nhưng hiểu quả để áp dụng kỹ thuật này là sử dụng lại toàn bộ một mạng CNN được train trên tập dữ liệu lớn, thay layer cuối cùng bằng một

Softmax Regression nhưng với số lượng units bằng với số lượng class ở bộ cơ sở dữ liệu mới và chỉ huấn luyện lại tham số cho layer này để giảm thiểu chi phí tính toán và thời gian

Ngoài ra, chúng ta cũng có thể sử dụng feature map ở những layer gần cuối (fully connected layers) rút ra từ các mạng được huấn luyện sẵn như một feature (deep feature) và sử dụng các thuật toán máy học như SVMs (Support Vector Machines) để huấn luyện các bộ phân lớp cho bài toán mới.

## 3.5. Handcrafted features

Hand-crafted features là thuật ngữ dùng để chỉ các đặc trưng được rút trích từ thông tin từ ảnh (giá trị màu của các pixels) được nghĩ ra bởi con người và hiện thực thông qua các thuật toán cụ thể (Ví dụ như SIFT, Bag of worlds, Haris coner detector, Canny edges detector, …). Các đặc trưng thuộc loại này rất đa dạng và phụ thuộc nhiều vào bài toán cần giải quyết. Trước khi có CNN chúng là lựa chọn phổ biến cho các bài toán phân lớp khi kết hợp với SVM.

Tuy sự phổ biến và độ hiệu quả của các handcrafted features không còn bằng các đặc trưng rút từ các mạng CNN đã được huấn luyện sẵn, nhưng các handcrafted features này cũng đem lại các thông tin hữu ít và thông dụng mà từ đó có thể kết hợp chúng với các các đặc trưng từ CNN để cải thiện kết quả phân lớp. Có rất nhiều nghiên cứu theo hướng tiếp cận này và chứng minh được sự hiểu quả khi kết hợp chúng: [16], [17], [1].

Trong luận văn này, nhóm sử dụng hai loại handcrafted features: canny edge detector thể hiện thông tin về hình dạng và Entropy fitering để thể hiện thông tin về texture của ảnh.

### Canny edges detector

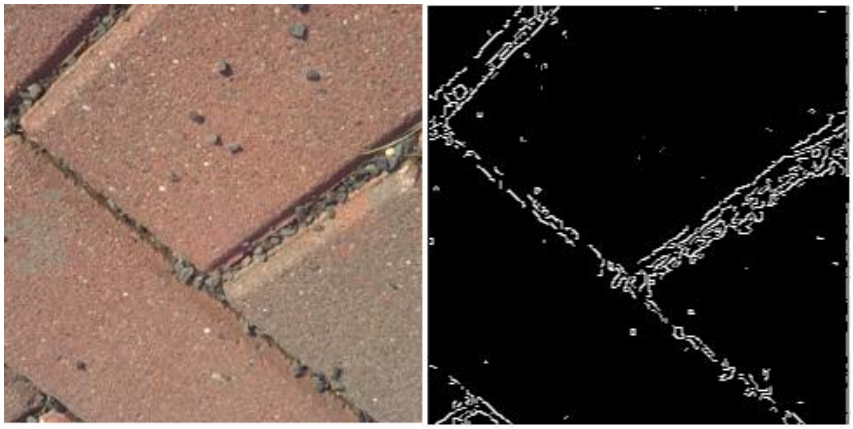
Canny edges detector là một kỹ thuật để rút trích các thông tin hữu ích về hình dạng của các đối tượng trong ảnh và làm giảm thiểu đáng kể lượng dữ liệu cần sử lý (). Nó

được áp dụng rộng rãi trong nhiều hệ thống thị giác máy tính khác nhau. Kỹ thuật này hoạt động dựa trên các tiêu chí để rút trích thông tin về cạnh bao gồm:

1. Detect các cạnh với độ lỗi nhỏ để phát hiện nhiều cạnh nhất có thể từ ảnh.
2. Các điểm trên cạnh được phát hiện cần được chuẩn hóa vào trung tâm của cạnh đó.
3. Một cạnh trên ảnh chỉ được đánh dấu 1 lần để tránh bị nhiễu khi lặp lại nhiều lần 1 cạnh.

Quy trình của thuật toán có thể được chia nhỏ thành 5 bước khác nhau:

1. Sử dụng Gaussian filter để khử nhiễu cho ảnh.
2. Tính gradients của ảnh đã được khử nhiễu.
3. Sử dụng "non-maximum suppression" để loại bớt các cạnh nhỏ
4. Sử dụng một ngưỡng kép (double threshold) (trên và dưới) để lấy các cạnh tiềm năng nhất
5. Loại bỏ các cạnh có ít liên kết với các cạnh xung quanh.



Hình 3.10. Canny edges detector được dùng để rút các thông tin về cạnh (bên trái) từ ảnh (bên phải) (Ảnh lấy từ tập dữ liệu Ground Terrain in Outdoor Scenes)

**Entropy filter**

Entropy filter là một kỹ thuật trong Image Processing và được dùng nhiều trong Texture Analysis để rút trích các thông tin trực quan về texture của ảnh được mô tả

qua các thuật ngữ như: thô, mịn, mượt, gập ghềnh, … đặc trưng cho sự thay đổi không gian trong ảnh. Kỹ thuật này đặc biệt hữu ích trên các ảnh mà các phương pháp thresholding truyền thống (như các thuật toán edges detector) không thể sử dụng một cách hiệu quả. Nhóm sử dụng nó như một sự bổ sung cho edges detector vì trên thực tế, thông tin về cạnh có thể không hiệu quả trên nhiều bề mặt ngoài trời (như đá, thảm cỏ, nước, …).

# Chương 4. **PHƯƠNG PHÁP ĐỀ XUẤT**

## 4.1. Ý tưởng

Sau khi thực hiện các phương pháp baseline với deep feature rút từ mạng CNN VGG16, nhóm nhận thấy việc sử dụng các đặc trưng cho bài toán phân lớp đối tượng để phân loại vật liệu khiến các bộ phân lớp dễ phân loại sai cho những ảnh có đối tượng giống nhau về hình dạng hoặc texture nhưng làm từ vật liệu khác nhau (Hình 4.1 và Hình 4.2). Để vừa có thể hạn chế sự nhầm lẫn này mà vẫn kế thừa được sự thành công từ các mạng CNN đã được huấn luyện trên những tập dữ liệu rất lớn cho bài toán phân lớp đối tượng từ đó có thể cải thiện kết quả phân lớp, nhóm đề xuất phương pháp kết hợp (dựa trên sự thành công của các nghiên cứu trước đó cũng sử dụng phương pháp kết hợp đặc trưng: [16], [17], [1].):

- ✓ Deep feature rút trích từ một CNN được huấn luyện sẵn trên các tập dữ liệu lớn. (kế thừa).
- ✓ Handcrafted features thể hiện thông tin về hình dạng và textures của đối tượng (giải quyết các trường hợp nhầm lẫn).

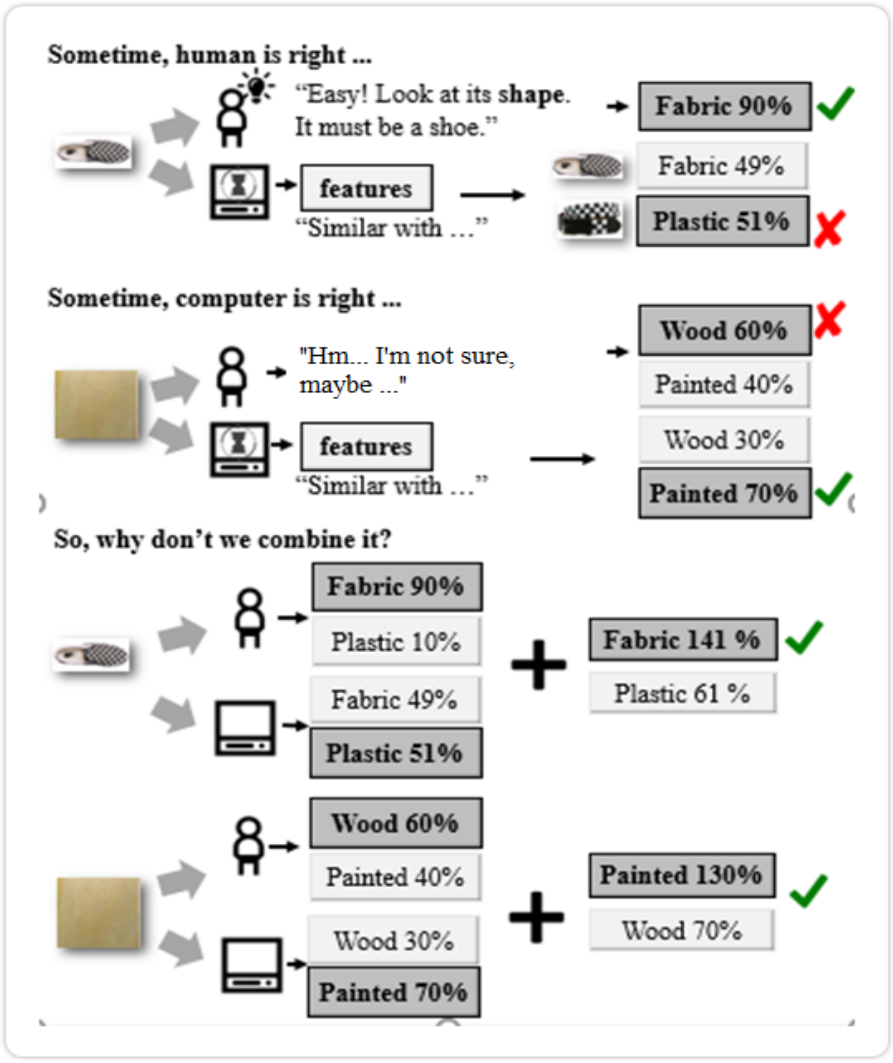

Hình 4.1: Các đối tượng có hình dạng khác nhau được làm từ vật liệu khác nhau nhưng lại giống nhau về texture (Ảnh từ Flicker Material Dataset)

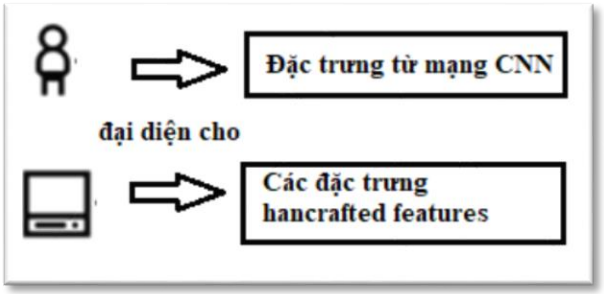
Về bản chất, các phép kết hợp này đều là kết hợp các vector với nhau. Khi bắt đầu các thực nghiệm, vì chưa có cơ sở phương pháp kết hợp nào tốt hơn nên nhóm quyết

định sử dụng hai phương pháp kết hợp cơ bản để thực hiện chúng, đó là: phép kết hợp trung bình cộng và phép nối vector (cụ thể sẽ được trình bày bên dưới).

Hình 4.2: Các đối tượng chính của hai ảnh đều có hình dạng giống nhau và có thể bị nhầm lẫn cả hai đều là đá.

Hình 4.3 minh họa cho ý tưởng này, việc kết hợp các đặc trưng giúp chúng có thể bổ trợ cho nhau trong các trường hợp nhập nhằng giữa các lớp (có kết quả gần như nhau – không có kết quả nào trội hơn hẳn). Như vậy, khi kết hợp nhiều đặc trưng, cơ hội phân lớp chính xác các trường hợp nhập nhằn này sẽ cao hơn khi chỉ dùng một đặc trưng duy nhất.




Hình 4.3: Deep features rút trích từ một mạng CNN đã được huấn luyện sẵn thê hiện cho cách con người học, kết hợp chúng với các handcrafted features thích hợp có thể giúp cải thiện kết quả phân lớp

## 4.2. Hướng tiếp cận

Dựa trên thành công của các phương pháp kết hợp trong các nghiên cứu trước ( [16], [17], [1]), nhóm đề xuất 3 mô hình kết hợp khác nhau như bên dưới. Ngoài ra, sau khi hoàn thành thực nghiệm của 3 mô hình này và xem xét các kết quả đạt được, nhóm đề xuất cách tích hợp các mô hình vào một mạng CNN duy nhất để tăng hiệu suất tính toán, không gian lưu trữ và khả năng thích ứng với dữ liệu mới.

### 4.2.1. Mô hình post-fusion

Cấu trúc của mô hình này gồm 3 nhánh song song nhau. Nhánh thứ nhất nhận ảnh đầu vào sau đó dùng một mạng CNN được huấn luyện sẵn để rút deep feature và huấn luyện bộ phân lớp thứ 1. Nhánh thứ 2 và thứ 3 lần lượt nhận cùng ảnh đầu vào đó và dùng cái bộ lọc tương ứng để rút các thông tin về hình dạng và texture trong ảnh (handcrafted features) sau đó được dùng để huấn luyện bộ phân lớp thứ 2 và 3.

Sau khi 3 bộ phân lớp này đã được huấn luyện, chúng được dùng để phân lớp các ảnh trong tập test và cho ra 3 kết quả khác nhau (3 vectors of scores), 3 kết quả này sau đó được kết hợp với nhau để cho ra kết quả cuối cùng (Hình 4.4). Công thức (4.1) là cách kết hợp các kết quả phân lớp của ba bộ phân lớp:

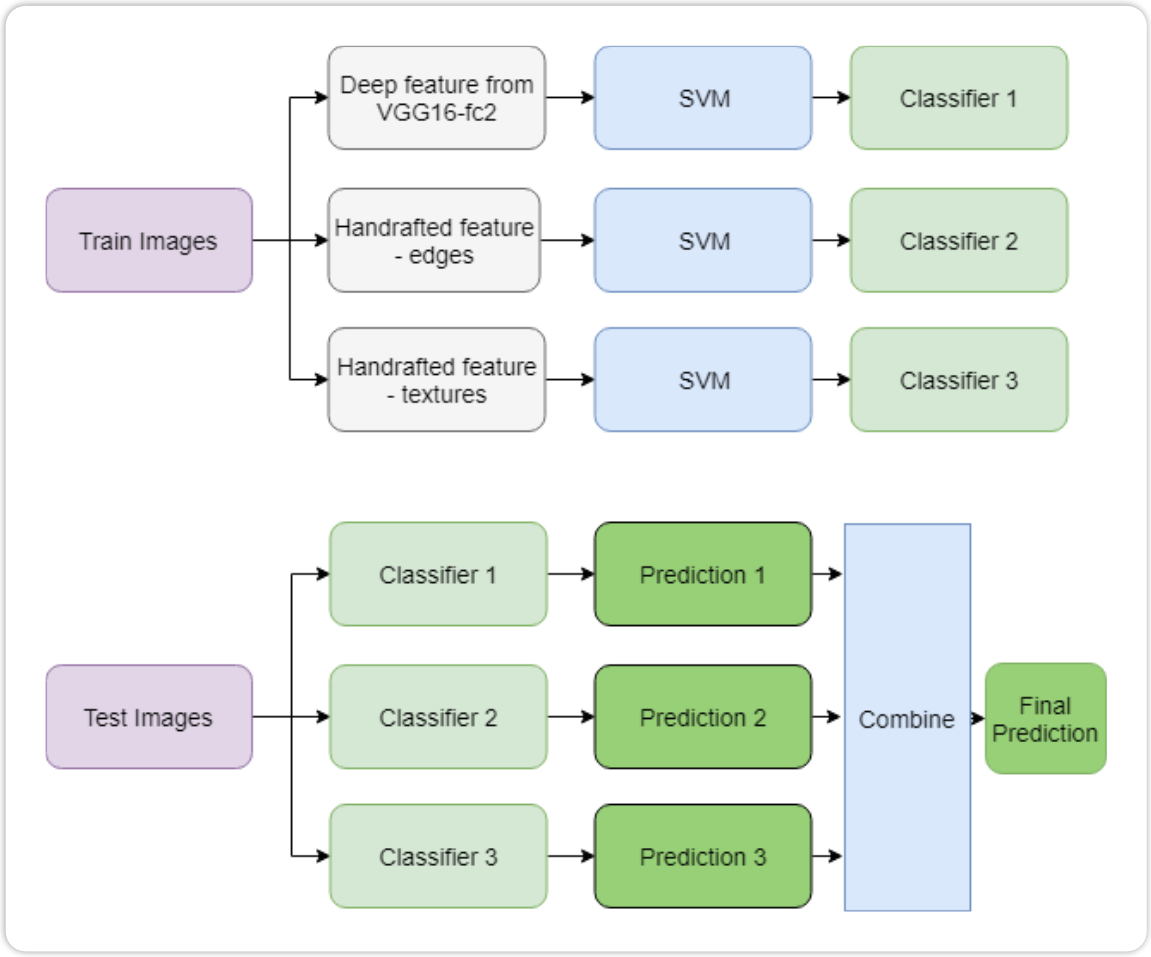
$P_1 = (p_1^1, p_2^1, \dots, p_n^1)$

$P_2 = (p_1^2, p_2^2, \dots, p_n^2)$

$P_3 = (p_1^3, p_2^3, \dots, p_n^3)$

Trong đó: $p_i^j$ là kết quả phân lớp cho lớp $i$ của bộ phân lớp $j$.

$$P_{combined} = \left(\frac{p_1^1+p_1^2+p_1^3}{3}, \dots, \frac{p_n^1+p_n^2+p_n^3}{3}\right) \quad (4.1)$$

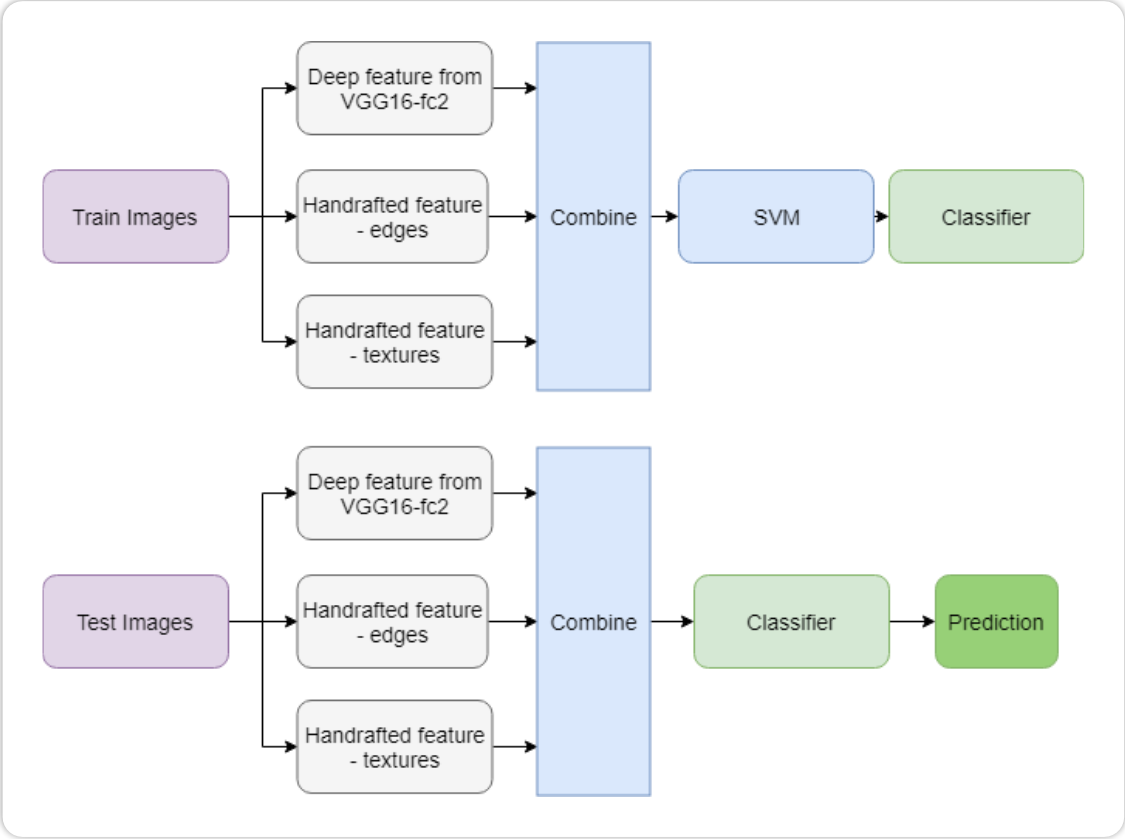



Hình 4.4: Mô hình 1: Kết hợp kết quả phân lớp dưới dạng xác suất (probability predictions) từ nhiều bộ phân lớp khác nhau

### 4.2.2. Mô hình pre-fusion

Với mô hình này, đầu vào cũng có ba nhánh tương tự mô hình thứ nhất, tuy nhiến chúng sẽ không đi song song, thay vào đó các feature rút trích từ ba nhánh sẽ được kết hợp thành một và chỉ huấn luyện một bộ phân lớp duy nhất trên feature đã được kết hợp này. Trong quá trình test, các features từ ba nhánh cũng được kết hợp tương tự và sau đó đi qua bộ phân lớp đã huấn luyện để có kết quả cuối cùng (Hình 4.5). Công thức (4.2) là cách các features từ các nhánh được kết hợp với nhau. Trong đó, $f_i^j$ là giá trị thứ $i$ trong feature vector thứ $j$ và $m, n, p$ lần lượt là độ dài của ba vector feature trong mô hình này.

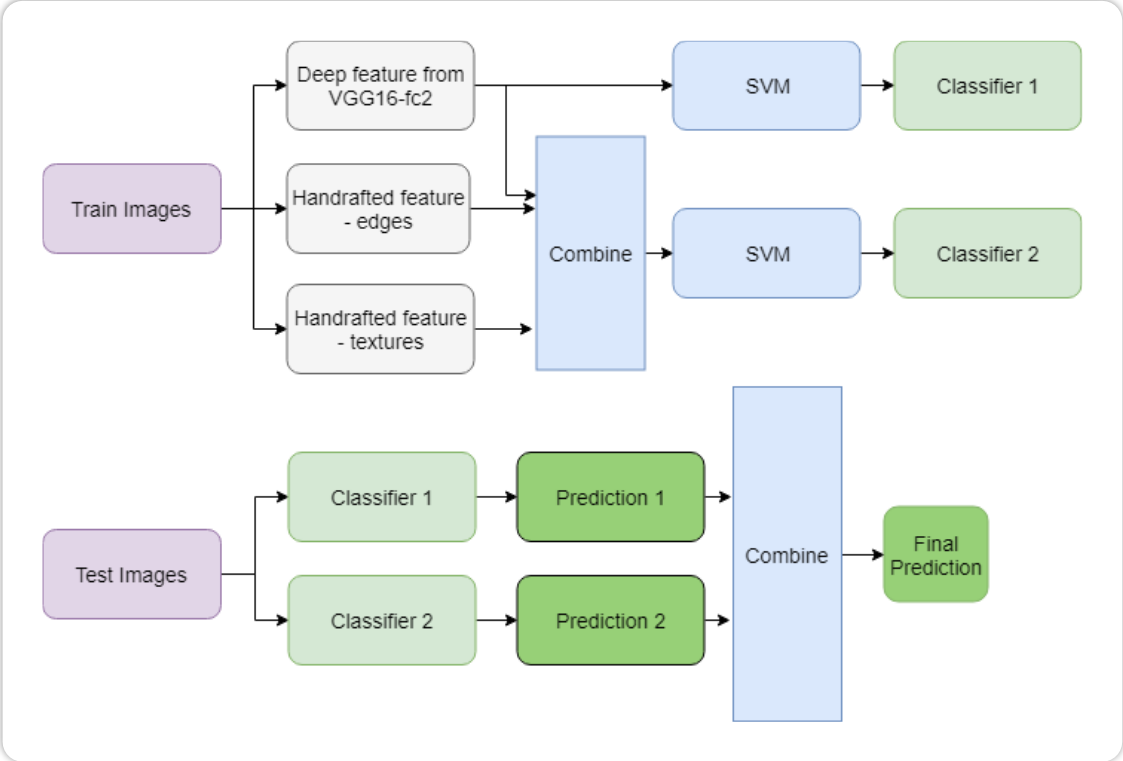
$$F_{combined} = \left(f_1^1, f_2^1, \dots, f_n^1, f_1^2, \dots f_m^2, f_1^3, \dots, f_p^3\right) \quad (4.2)$$




Hình 4.5: Mô hình 2: kết hợp nhiêu features khác nhau và huấn luyện một bộ phân lớp duy nhất (pre-fusion)

### 4.2.3. Mô hình full-fusion

ô hình này là sự kết hợp của pre-fusion và post fusion bên trên để có thể cho kết quả tốt nhất. Đầu tiên, ảnh huấn luyện sẽ được rút deep feature và các handcrafted features tương tự như trên, sau đó deep feature sẽ dùng để huấn luyện một bộ phân lớp riêng, cùng với đó, deep feature cũng sẽ được kết hợp với các handcrafted features còn lại để huấn luyện một bộ phân lớp riêng (Hình 4.6).




Hình 4.6: Mô hình 3: Kết hợp mô hình 1 cho kết quả phân lớp và mô hình 2 cho các features dùng để huấn luyện các bộ phân lớp.

Với mô hình này, nhánh trên (chỉ dùng deep feature) có thể coi là transfer learning từ một mạng CNN đã được huấn luyện sẵn trên ImageNet và tương ứng với cách dùng bài toán phân lớp đối tượng để giải bài toán phân lớp vật liệu, cùng với đó nhánh 2 kết hợp với các handcrafted features thích hợp sẽ giải quyết được các trường hợp nhập nhằng khi hai đối tượng có cùng hình dạng hay texture, chính vì thế kết quả phân lớp sẽ tốt hơn.

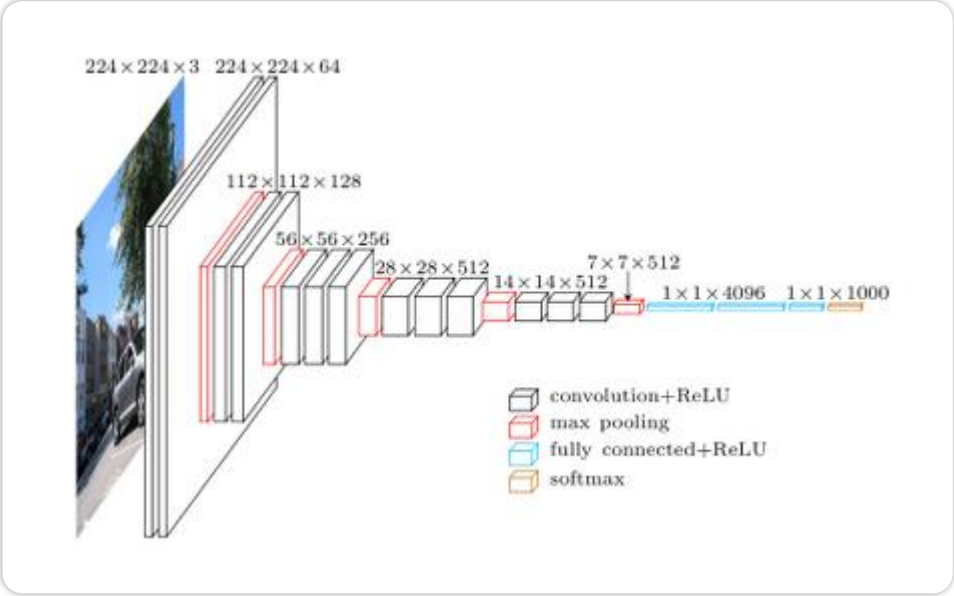
## 4.3. Thiết kế mạng CNN

Cách rút trích features và huấn luyện bằng SVMs như các mô hình trên cần lưu trữ một lượng lớn dữ liệu (các feature đã được rút trích). Bên cạnh đó quá trình huấn luyện (bằng SVMs) mất rất nhiều thời gian, tuy nhiên để có một kết quả tốt nhóm cần phải thử nghiệm với rất nhiều bộ tham số khác nhau (của SVMs) nên thời gian và khối lượng tính toán không hề nhỏ, thêm vào đó các bộ phân lớp sau khi đã huấn

luyện không thể dùng lại trong lần huấn luyện sau. Thế nên, nhóm đã đề xuất phương pháp kết hợp các mô hình trên vào một mạng CNN duy nhất. Mục tiêu được đề ra là với mạng CNN nhóm sẽ có thể:

1. Gộp cả 3 quá trình rút trích feature, huấn luyện và test vào một quá trình duy nhất là huấn luyện mạng CNN này.
2. Giảm chi phí (không gian lưu trữ, khối lượng và thời gian tính toán).
3. Khả năng mở rộng và sử dụng lại: mạng CNN có khả năng tự thay đổi tham số để thích ứng tốt hơn với dữ liệu mới (các bộ phân lớp của SVMs thì không thể).

Nhóm chọn VGG16 để tìm hiểu cấu trúc và dựa trên nó để xây dựng một mạng CNN riêng vì mạng này đã chứng minh được sự hiệu quả trên ImageNet và được kế thừa trong nhiều bài toán phân lớp khác nhau trước đó. Với số lượng layer (16) so với các mạng CNN khác như Resnet (152), GoolgeNet (22), đây là lựa chọn mà nhóm cho là hợp lý với lượng dữ liệu không quá lớn trong hai tập dữ liệu mà nhóm thực hiện các thực nghiệm.




Hình 4.7: Cấu trục mạng VGG16 được nhóm dùng để rút trích đặc trưng ở layer 'fc2' và thực hiện các thực nghiệm ở phần thực nghiệm với VGG16 [18]

Quá trình thực nghiệm với VGG16 qua các bước sau đây:

1. Fine-tune VGG16: Thay fully-connected layer cuối cùng để đầu ra là 39 lớp (trên GTOS) thay vì 1000 lớp (trên ImageNet) và huấn luyện lại giá trị tham số của layer này, sau đó huấn luyện lại tất cả các layer để so sánh kết quả.
2. Tìm hiểu sự ảnh hưởng của các layer trong mạng bằng cách bỏ bớt layer trong mạng, quan sát ảnh hưởng của các layer này trên kết quả thu được.
3. Qua quan sát từ bước 2 và dựa trên nền của VGG16 chọn các layer thích hợp để xây dựng một mạng CNN ít layer hơn nhưng đạt kết quả xấp xĩ.

# Chương 5. **THỰC NGHIỆM VÀ KẾT QUẢ**

Nhóm thực hiện thực nghiệm trên hai tập dữ liệu Ground Terrain in Outdoor Scenes (GTOS) [1] và Flicker Material Dataset (FMD) [19], sau đó so sánh kết quả với các phương pháp state-of-the-art để xem xét sự ảnh hưởng của các handcrafted features trên những dataset khác nhau.

Trên mỗi tập dữ liệu, nhóm đầu tiên thực nghiệm với từng cấu trúc mạng nhóm đề xuất trong phần trên và đánh giá sự hiểu quả của chúng, sau đó so sánh kết quả khi sử dụng các handcrafted feature khác nhau để chỉ ra rằng việc kết hợp handcrafted features càng thích hợp với tập dữ liệu, kết quả sẽ càng tốt. Sau cùng, để có thể thiết kế một CNN có thể tích hợp các phần trên lại với nhau, nhóm thực hiện một số thực nghiệm nhằm hiểu rõ hơn cấu trúc, cách hoạt động của từng layer của một mạng CNN phổ biến và hiểu quả - VGG16, từ đó có thể dựa trên cấu trúc của mạng này cũng như chọn được những layer thích hợp nhất để xây dựng một CNN riêng.

## 5.1. Môi trường thực nghiệm

Các thực nghiệm trong luận văn này được thực hiện trên PC có cấu hình:

1. Processor: Intel(R) Core(TM) i7 - 62000U CPU @ 2.40GHz 3.2GHz
2. RAM: 16GB
3. Opera system: Windows 10 (x64)

Các công cụ được sử dụng bao gồm:

1. Python 3.6+: Ngôn ngữ lập trình chính để hiện thực các bước thực nghiệm
2. Matlab: Hổ trợ rút trích thông tin về texture và các thực nghiệm nhỏ liên quan
3. scikit-learn: Hổ trợ quá trình huấn luyện các bộ phân lớp với SVM, visualize kết quả
4. Keras: Hổ trợ việc rút trích deep feature từ CNN và thực hiện các thực nghiệm trên VGG16
5. Tensorflow: back-end cho Keras.
6. OpenCV: Dùng cho việc rút trích các handcrafted features và visualize kết quả

## 5.2. Các tập dữ liệu

Như đã đề cập bên trên, nhóm sử dụng hai tập dữ liệu để thực nghiệm: GTOS và FMD.

GTOS là tập dữ liệu mới nhất cho bài toán phân lớp vật liệu, được hoàn thành vào năm 2017 bởi Rutgers ECE Vision Lab. Bao gồm hơn 34000 ảnh thuộc 40 lớp khác nhau được thu thập dưới nhiều điều kiện thời tiết, chiếu sáng khác nhau, đây là một trong những tập dữ liệu đa dạng nhất cho bài toán phân lớp vật liệu.

Hình 5.1: Một cảnh ngoài trời được dùng dể lấy các mẫu cho tập dữ liệu GTOS với nhiều góc nhìn nhìn, điều kiện chiếu sáng khác nhau [1]

Ở chiều hướng ngược lại, FMD là một trong những tập dữ liệu phổ biến nhất cho phân loại vật liệu ngay từ những ngày đầu ra đời của bài toán này khoảng đầu năm 2009. Tất cả các ảnh dùng trong hai tập dữ liệu này đều có kích thước 244 x 244.

Hình 5.2: Một tập ảnh từ FMD, đảm bảo sự đa dạng về điều kiện chiếu sáng, bố cục, màu sắc, kết cấu [6]

## 5.3. Độ đo đánh giá

Với bài toán phân lớp, có nhiều độ đo có thể được dùng để đánh giá kết quả thực nghiệm, nhưng các nghiên cứu hiện nay chủ yếu tập trung vào đồ chính xác trung bình (mean accuracy) [1] [2] [3] [8]. Độ chính xác trung bình được tính theo công thức sau:

$$Acc = \frac{n_{true\_samples}}{n_{total\_samples}} \; x100 \; (\%)$$

Trong đó:

$Acc$: Độ chính xác trung bình của bộ phân lớp cho lớp i (%).

$n_{true\_samples}$: số lượng ảnh được phân lớp đúng trong lớp i.

$n_{total\_samples}$: tổng số lượng ảnh đã được gán nhãn thuộc lớp i.

## 5.4. Quá trình thực nghiệm và kết quả

Nhóm cài đặt các thực nghiệm bằng ngôn ngữ Python với sự hỗ trợ của framework scikit-learn và keras (trên nền tensorflow). Mạng VGG16 được huấn luyện sẵn trên ImageNet cũng được dùng cho việc transfer learning và các thực nghiệm khác nhằm hiểu rõ hơn về cách hoạt động của các layer trong CNN.

Nhóm sử dụng cùng tập dữ liệu với các nghiên cứu trước [1] [2] [3]., cụ thể như sau

### Tập dữ liệu GTOS

Với tập GTOS, dữ liệu được chia thành 5 tập khác nhau theo tỷ lệ 70% cho huấn luyện và 30% cho đánh giá (23970 ảnh cho huấn luyện và 10272 ảnh cho đánh giá). Để đảm bảo không có sự chồng chéo giữa các tập huấn luyện và đánh giá, nếu một

ảnh của một mẫu x nào đó nằm trong tập huấn luyện, thì toàn bộ ảnh còn lại của mẫu đó cũng nằm trong tập huấn luyện.

**Tập dữ liệu FMD**

Với tập FMD, dữ liệu cũng được chia thành 5 tập nhưng với tỷ lệ 80% cho huấn luyện và 20% (800 ảnh cho huấn luyện và 200 ảnh để đánh giá) cho đánh giá giống như được sử dụng trong các nghiên cứu trước [2] [3].

### 5.4.1. Thực nghiệm với ba mô hình đề xuất

**Tiền xử lý dữ liệu**

Ảnh từ các tập dữ liệu đầu vào được cho qua một bước tiền xử lý trước khi sử dụng để xây dựng các bộ phân lớp. Đầu tiên, các ảnh được "crop" thành kích thước 224 x 224 để có sự đồng bộ và có thể dùng làm đầu vào của mạng CNN đã được huấn luyện sẵn trên Image-Net. Sau đó, mỗi ảnh được chuẩn hóa (standardize) theo công thức sau:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Trong đó,

$x'$: giá trị (của một trong 3 kênh màu RGB) tại điểm ảnh sau khi chuẩn hóa.

$x$: giá trị (của một trong 3 kênh màu RGB) tại điểm ảnh trước khi chuẩn hóa.

$\bar{x}$: giá trị trung bình của kênh màu tương ứng.

$\sigma$: độ lệch chuẩn của các giá trị trong kênh màu tương ứng.

Lý do thực hiện các bước chuẩn hóa trên là vì dữ liệu thực tế (đặc biệt là ảnh) có vùng giá trị rất lớn và điều này làm cho việc tính toán khoảng cách giữa các điểm dữ liệu (như là khoảng cách Euclidean) trong quá trình xây dựng các bộ phân lớp bị ảnh hưởng rất nhiều dẫn tới kết quả phân lớp không được tốt. Vì vậy vùng giá trị của dữ liệu cần được chuẩn hóa trước khi sử dụng chúng cho các bộ phân lớp.

**Rút trích đặc trưng**

Nhóm thực hiện rút trích các đặc trưng cần thiết cho các bộ phân lớp cùng lúc sử dụng nhiều máy khác nhau để giảm thiểu thời gian cho công đoạn này.

- ✓ Deep feature: nhóm sử dụng mạng VGG16 với tham số đã được huấn luyện trên Image-Net để rút trích deep feature tại layer 'fc2'.

- ✓ Đặc trưng về hình dáng: nhóm sử dụng thuật toán "Canny edge" với giá trị thresholds trong khoảng [100, 200] để rút trích các thông tin về hình dạng (sử dụng thư viện OpenCV).
- ✓ Đặc trưng về texture: nhóm sử dụng hai texture filter: *local range* và *local entropy* được tích hợp trong Mathlab cho bài toán texture segmentation để rút trích các thông tin về texture với vùng local neighborhood có kích thước 3x3.

Sau khi các đặc trưng được rút trích, nhóm thực hiện bước kết hợp các feature này để chuẩn bị huấn luyện các bộ phân lớp theo các mô hình đã trình bày ở trên.

**Quá trình huấn luyện**

Quá trình huấn luyện các bộ phân lớp được thực hiện bằng SVM với thư viện scikit-learn (Python) trên nhiều bộ tham số khác nhau nhằm điều chỉnh và tìm ra các kernel thích hợp cho các tập dữ liệu. Cụ thể, nhóm thực nghiệm với linear kernel, poly kernel và rbf kernel với các bộ tham số được configure trong các khoảng: $C \in [1x10^{-5}, 1000]$, $D(degree) \in [1, 5]$ và $G(gamma) \in [10^{-5}, 1]$.

**Quá trình thử nghiệm các bộ phân lớp**

Sau khi đã huấn luyện các bộ phân lớp, chúng được dùng để thử nghiệm trên các tập test khác nhau tương ứng và kết quả được lưu dưới dạng vector giá trị xác suất mà một lớp nào đó là đầu ra cuối cùng. Sau đó, các kết quả xác suất này được kết hợp với nhau theo phép toán trung bình để cho ra một vector xác xuất duy nhất. Cuối cùng lớp nào có giá trị xác suất lớn nhất sẽ là đầu ra của bộ phân lớp. Ngoài ra, để thuật tiện cho việc theo dõi kết quả và điều chỉnh tham số để huấn luyện lại các bộ phân lớp theo chiều hướng tốt hơn, nhóm có xem xét kết quả từ các confusion matrix được lưu lại và kết quả của các lớp có giá trị xác suất lớn thứ 2, 3, 4, …

Bên dưới là một số kết quả thu được.

| **Mô hình** | **Deep feature** | **Deep + edges** | **Deep + edges + texture** |
|---|---|---|---|
| Post-fusion | 75 ± 1.8 | 75.5 ± 3.0 | 76.3 ± 1.9 |
| Pre-fusion | 75 ± 1.8 | 77.8 ± 2.5 | 79.5 ± 2.5 |
| Full-fusion | 75 ± 1.8 | 78.9 ± 2.2 | **82.2 ± 2.3** |

Bảng 5.1: Kết quả thực nghiệm ba mô hình trên tập GTOS (%)

| Mô hình | Deep feature | Deep + edges | Deep + edges + texture |
|---|---|---|---|
| Post-fusion | 74.2 ± 1.4 | 75.1 ± 2.2 | 76.3 ± 1.7 |
| Pre-fusion | 74.2 ± 1.4 | 75.3 ± 1.5 | 76.3 ± 1.7 |
| Full-fusion | 74.2 ± 1.4 | 75.5 ± 1.2 | **77.2 ± 0.9** |

Bảng 5.2: Kết quả thực nghiệm ba mô hình trên tập FMD (%)

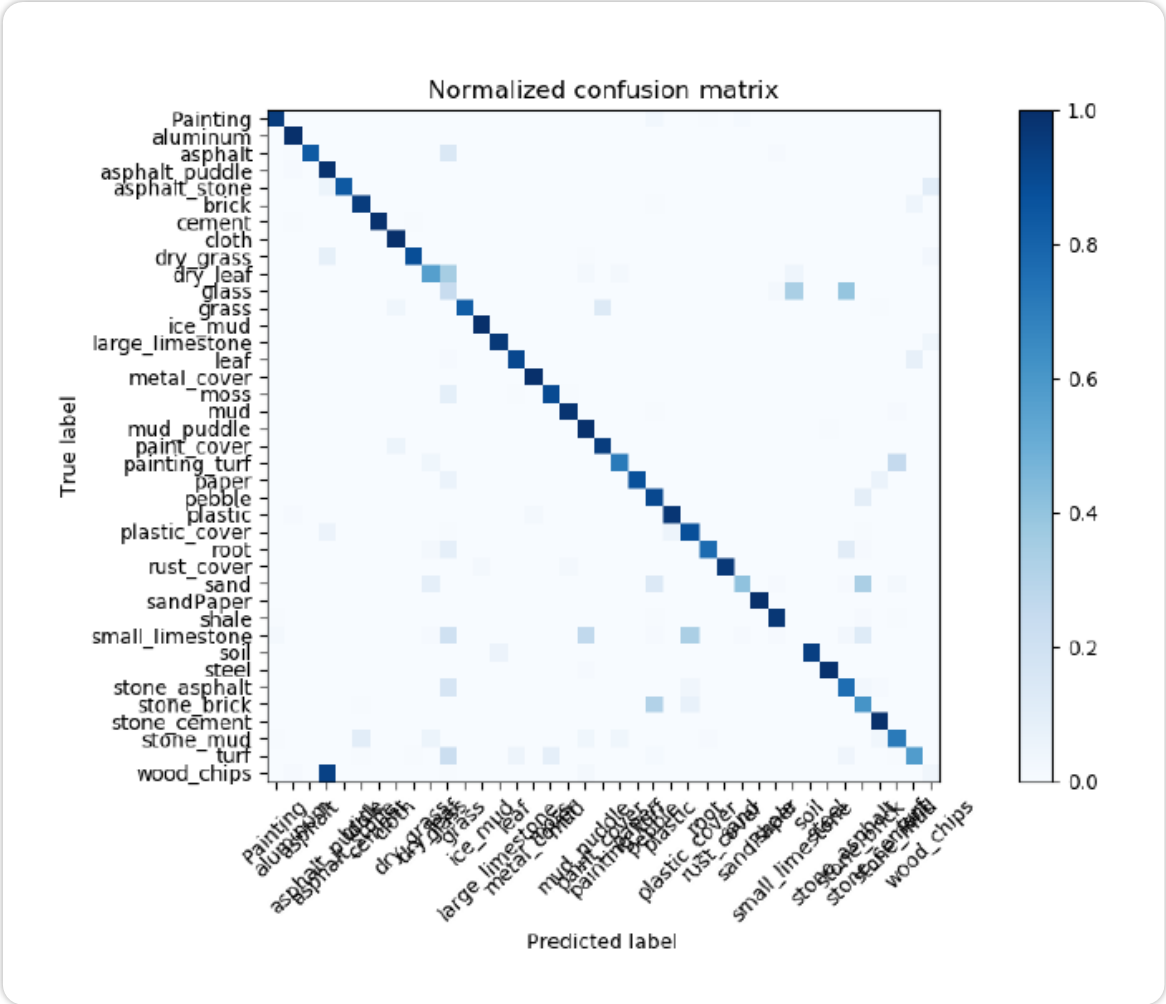
Bảng 5.1 và Bảng 5.2 thể hiện kết quả trên tập GTOS và FMD với ba mô hình đề xuất. Kết quả cho thấy mô hình thứ 3 hoạt động tốt nhất. Hai handcrafted features được dùng thể hiện thông tin về hình dạng và texture hoạt động hiệu quả trên hai tập dữ liệu này. Tuy nhiên, có thể thấy ảnh hưởng của hai features này trên hai tập dữ liệu là khác nhau, trên GTOS hiệu quả của chúng lớn hơn so với trên FMD (đối với cả ba mô hình). Lý do cho điều này được cho là do nhiều ảnh trong FMD là một đối tượng chính nằm trên một background còn GTOS ngược lại, toàn bộ hình đều đồng nhất là một bề mặt duy nhất (Hình 5.3). Chính background này khiến thông tin về hình dạng và texture bị ảnh hưởng khác nhiều.

Hình 5.3: Ảnh từ FMD (bên trái) trong lớp vật liệu "giấy" lại có background là một mặt đường khiến thông tin về texture không còn sự hiệu quả, trong khi ảnh từ GTOS là một bề mặt đồng nhất duy nhất (lớp "Gạch")

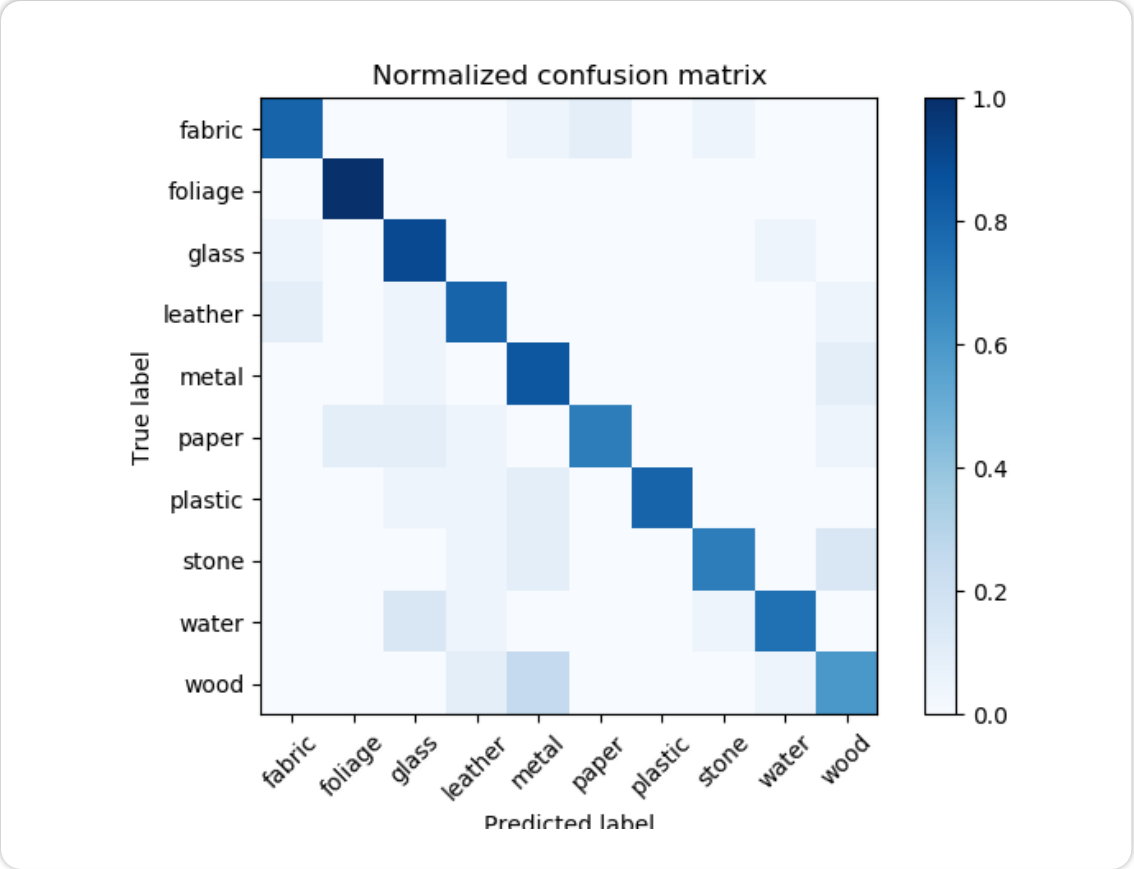
| Phương pháp | GTOS | FMD |
|---|---|---|
| DAIN [1] | 81.2 ± 1.7 | |
| Reflectance [2] | | 65.5 |

| | | |
|---|---|---|
| SIFT IFV+fc7 [3] | | 69.6 ± 0.3 |
| Ours | **82.2 ± 2.3** | **77.2 ± 0.9** |

Bảng 5.3: So sánh kết quả với các phương pháp khác trên tập GTOS và FMD (%)




Hình 5.4: Normalized confusion matrix trên GTOS




Hình 5.5: Normalized confusion matrix trên FMD

| **Class** | **Acc (%)** | **Class** | **Acc (%)** | **Class** | **Acc (%)** | **Class** | **Acc (%)** |
|---|---|---|---|---|---|---|---|
| Painting | 95 | Glass | 83 | Painting turf | 96 | Small limestone | 87 |
| Aluminum | 82 | Grass | 98 | Paper | 90 | Soil | 77 |
| Asphalt | 90 | Ice mud | 84 | Pebble | 91 | Steel | 96 |
| Asphalt puddle | 76 | Large limestone | 95 | Plastic | 89 | Stone asphalt | 40 |
| Asphalt stone | 60 | Leaf | 99 | Plastic cover | 98 | Stone brick | 92 |
| Brick | 99 | Metal cover | 92 | Root | 99 | Stone cement | 97 |
| Cement | 71 | Moss | 88 | Rust cover | 94 | Stone mud | 97 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Cloth | 57 | Mud | 56 | Sand | 71 | Turf | 94 |
| Dry grass | 93 | Mud puddle | 23 | Sand paper | 87 | Wood chips | 98 |
| Dry leaf | 95 | Paint cover | 85 | Shale | 97 | | |

Bảng 5.4: Kết quả phân lớp cho từng lớp trong tập GTOS

| **Class** | **Acc (%)** | **Class** | **Acc (%)** |
|---|---|---|---|
| fabric | 75 | paper | 55 |
| foliage | 70 | plastic | 70 |
| glass | 95 | stone | 65 |
| leather | 90 | water | 80 |
| metal | 65 | wood | 85 |

Bảng 5.5. Kết quả phân lớp cho từng lớp trong tập FMD

### 5.4.2. Thực nghiệm với VGG16.

Bên dưới trình bày các bước mà nhóm đã thực hiện bước Fine-tune mạng VGG16 trên tập dữ liệu GTOS.

**Tiền xử lý dữ liệu**

Ngoài các bước tiền xử lý như đã thực hiện ở phần thực nghiệm với các mô hình đề xuất, nhóm thực hiện thêm một bước thực hiện các phép transformations ngẫu nhiên (xoay, phóng to, thu nhỏ, lật, …) để tăng kích thước cũng như sự đa dạng của tập dữ liệu (vì mạng CNN cần lượng dữ liệu khá lớn để có thể đạt được sự hiệu quả)

**Quá trình fine-tune**

Quá trình fine-tune trải qua các bước sau:

1. Khởi tạo một mạng với cấu trúc của VGG16
2. Lấy tham số đã được huấn luyện trên Image-Net truyền vào mạng này
3. Bỏ layer cuối cùng đi (1000 classes layer)

4. Thêm một layer softmax với số lượng lớp tương ứng với tập dữ liệu vào cuối mạng (40 lớp với tập GTOS).
5. Đánh dấu những layer cần train, và những layer muốn giữ lại tham số
6. Tùy chỉnh các tham số và tiến hành huấn luyện.

Các tham số huấn luyện cho tập GTOS được cấu hình như sau (tham khảo từ https://www.learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/):

1. $batch\_size = 5$: Tham số này được chỉnh theo phần cứng để huấn luyện nhanh hơn
2. $epochs = 20$
3. $steps\ per\ epoch = \frac{số\ lượng\ ảnh\ huấn\ luyện}{batch\ size}$
4. $validationSteps = \frac{số\ lưởng\ ảnh\ dùng\ để\ validate}{batch\ size}$

**Kết quả hiện tại**

Bảng 5.6 thể hiện một số kết quả trong thực nghiệm với VGG16

| | **Chỉ huấn luyện lại layer cuối** | **Huấn luyện lại toàn bộ layer** | **Chỉ sử dụng deep feature từ VGG16** | **Sử dụng kết hợp đặc trưng theo mô hình full-fusion** |
|---|---|---|---|---|
| Số lượng tham số | 39,039 | 134,299,583 | | |
| Độ chính xác | 77.9% | 78.1% | 75% | 82.2% |
| Không gian lưu trữ | ≈556 MB | ≈556 MB | >4GB (bao gồm các đặc trưng và bộ phân lớp) | > 10GB (bao gồm các đặc trưng và bộ phân lớp) |

Bảng 5.6. Kết quả thực nghiệm với VGG16 trên tập dữ liệu GTOS

Kết quả sau khi fine-tune trên GTOS cho thấy không gian lưu trữ giảm mạnh vì không cần tốn không gian lưu trữ đặc trưng (chỉ cần lưu lại các tham số của mạng) và độ chính xác giảm khoảng 4 – 5 % so với mô hình 3.

Bảng 5.7

# Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 6.1. Kết luận

Sau giai đoạn luận văn, nhóm đã áp dụng được các kiến thức chuyên ngành đã được học vào bài toán phân lớp vật liệu. Sau quá trình tìm hiểu, tổng hợp các nghiên cứu trước, nhóm đã nắm được các phương pháp cơ bản hiện có cho bài toán, dựa vào đó đưa ra một số mô hình nhằm cải thiện kết quả. Báo cáo đã trình bày những mô hình nhóm đề xuất cũng như quá trình cài đặt, thử nghiệm đánh giá các mô hình này trên những tập dữ liệu khác nhau.

## 6.2. Hướng phát triển

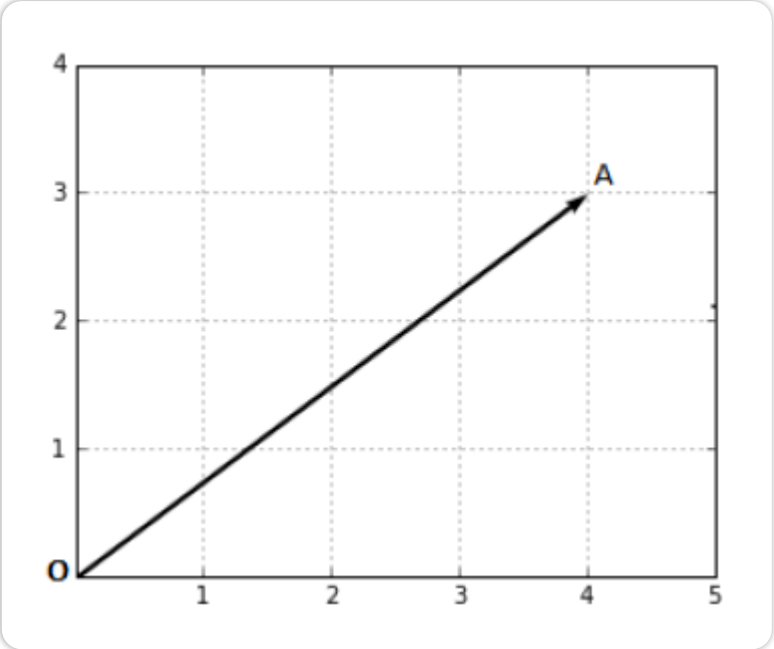
Với kết quả hiện tại của đề tài, nhóm nhận thấy các mô hình đề xuất còn có nhiều hướng phát triển, chỉnh sửa nhằm cải thiện cả về độ chính xác, hiệu suất, tài nguyên sử dụng. Bên dưới là một số hướng phát triển tiếp theo cho đề tài mà nhóm dự định thực hiện.

1. Tiếp tục các thực nghiệm với mạng VGG16 để có một cái nhìn sâu hơn về cách các layer của nó hoạt động, từ đó có thể thiết kế một mạng mới thích hợp cho bài toán hiện tại. Điều này giúp tích kiệm chi phí huấn luyện các mô hình trên đồng thời cũng tăng độ thích ứng của mô hình với dữ liệu mới (so với việc dùng SVMs để huấn luyện các bộ phân lớp).
2. Thay đổi cách kết hợp features và probability prediction như đã trình bày trong ba mô hình trên. Có thể kết hợp theo trọng số thay vì dùng phép nối vector và trung bình cộng như hiện tại.
3. Kết hợp thêm các thông tin khác có thể lấy từ ảnh (ví dụ như thông tin về góc nhìn - một mẫu với nhiều góc nhìn khác nhau, nghiên cứu [1] đã cho thấy thông tin này rất giá trị trong bài toán phân lớp vật liệu).

# PHỤ LỤC 1
# CÁC KHÁI NIỆM LIÊN QUAN TỚI SVM

**Vector**

Vector là một đối tượng toán học có thể được biểu diễn bằng một mũi tên (Hình 6.1). Vector được đặc trưng bởi hai thuộc tính chính là: độ dài (vector magnitude) và hướng (vector direction). Trong SVM, vector được dùng để biểu diễn cho các điểm dữ liệu (feature vector).

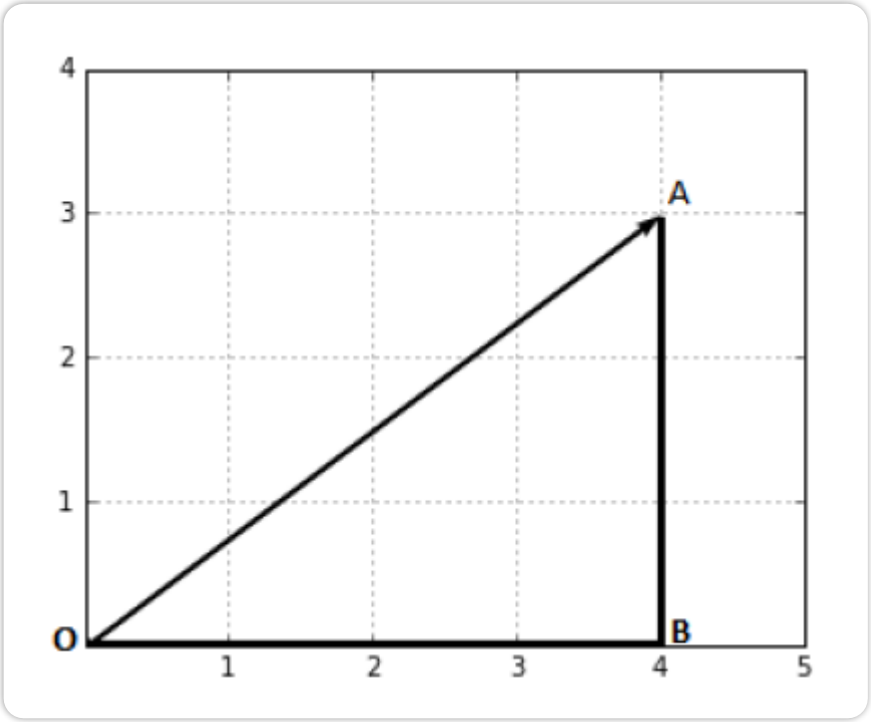



Hình 6.1: Vector là một đối tượng hình học được biểu diễn bằng một mũi tên

**Vector magnitude**

Magnitude của một vector $x = (x_1, \ldots, x_n)$ (kí hiệu là ||x||) là độ dài của vector x được tính dựa trên định lý Pythagorean (Hình 6.2). Công thức (3.1) là công thức chung để tính magnitude của vector x được.

$$||x|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad (3.1)$$



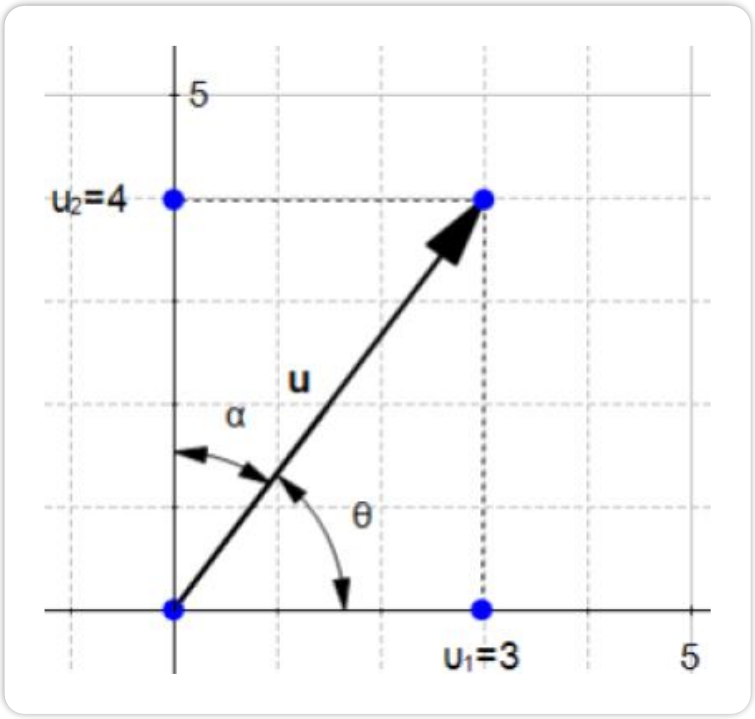

Hình 6.2: Magnitude của vector $\overrightarrow{OA}$ là độ dài đoạn OA

**Vector direction**

Hướng của vector $x = (x_1, \ldots, x_n)$ là một vector:

$$w = \left(\frac{x_1}{||x||}, \frac{x_2}{||x||}, \ldots, \frac{x_n}{||x||}\right)$$

Về mặt hình học, các giá trị của vector $w = (w_1, \ldots, w_n)$ chính là giá trị cosin của các góc mà vector x lần lượt tạo với các trục tọa độ trong không gian của nó. Hình 6.3 thể hiện góc của vector x với hai trục tọa độ trong không gian hai chiều.

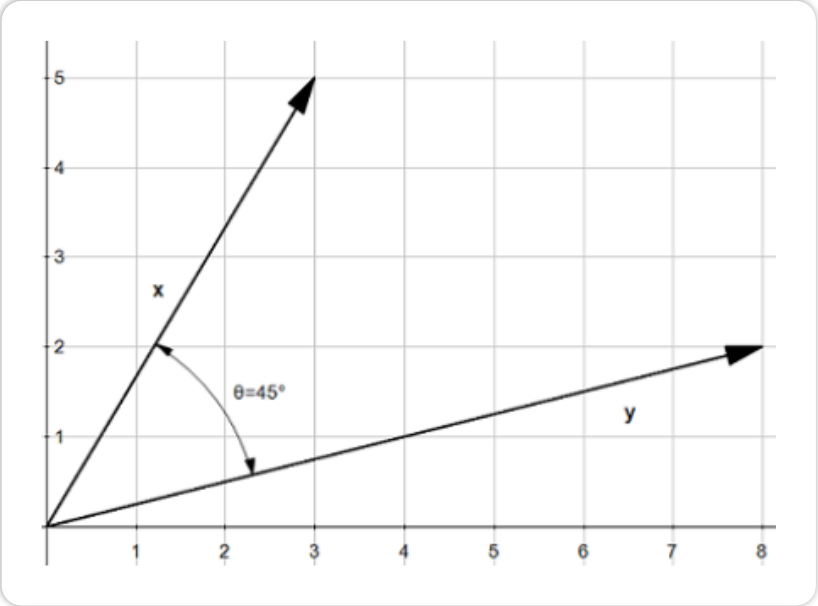



Hình 6.3: Hướng của vector u được thể hiện bởi góc mà u tạo với các trục tọa độ trong không gian của nó

$$w = \left(\frac{u_1}{||u||}, \frac{u_2}{||u||}\right) = (\cos\theta\,, cos\alpha)$$

**Tích vô hướng (dot product)**

Dot product là phép toán phổ biến được thực hiện trên hai vector và cho kết quả là một số thực (còn được gọi là scalar - vì vậy dot product còn có tên khác là scalar product). Dot product thể hiện mối quan hệ của hai vector. Về mặt hình học, dot product là tích của magnitude hai vector và cosin của góc tạo bởi hai vector này (Hình 6.4 và công thức (3.4)). Về mặt đại số, dot product có thể định nghĩa bằng công thức (3.5). Dot product là phép toán cơ bản được dùng nhiều trong việc xây dựng SVM, đặc biệt là tìm hyperplane.



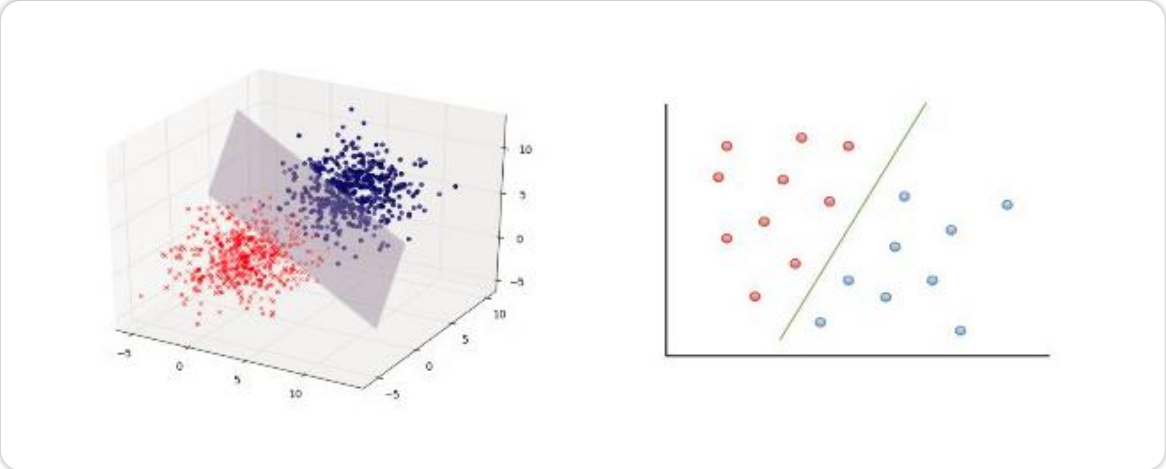

Hình 6.4: Hai vector x và y

$$xy = ||x||.||y||.cos\theta \quad (3.4)$$

$$xy = \sum_{i=1}^{n}(x_i.y_i) \quad (3.5)$$

**Hyperplane (Siêu phẳng)**

Về mặt hình học, có thể hiểu hyperplane là một không gian con có số chiều nhỏ hơn không gian chứa nó. Trong không gian hai chiều, hyperplane là đường thẳng. Trong không gian ba chiều, hyperplane là một mặt phẳng (Hình 6.5).



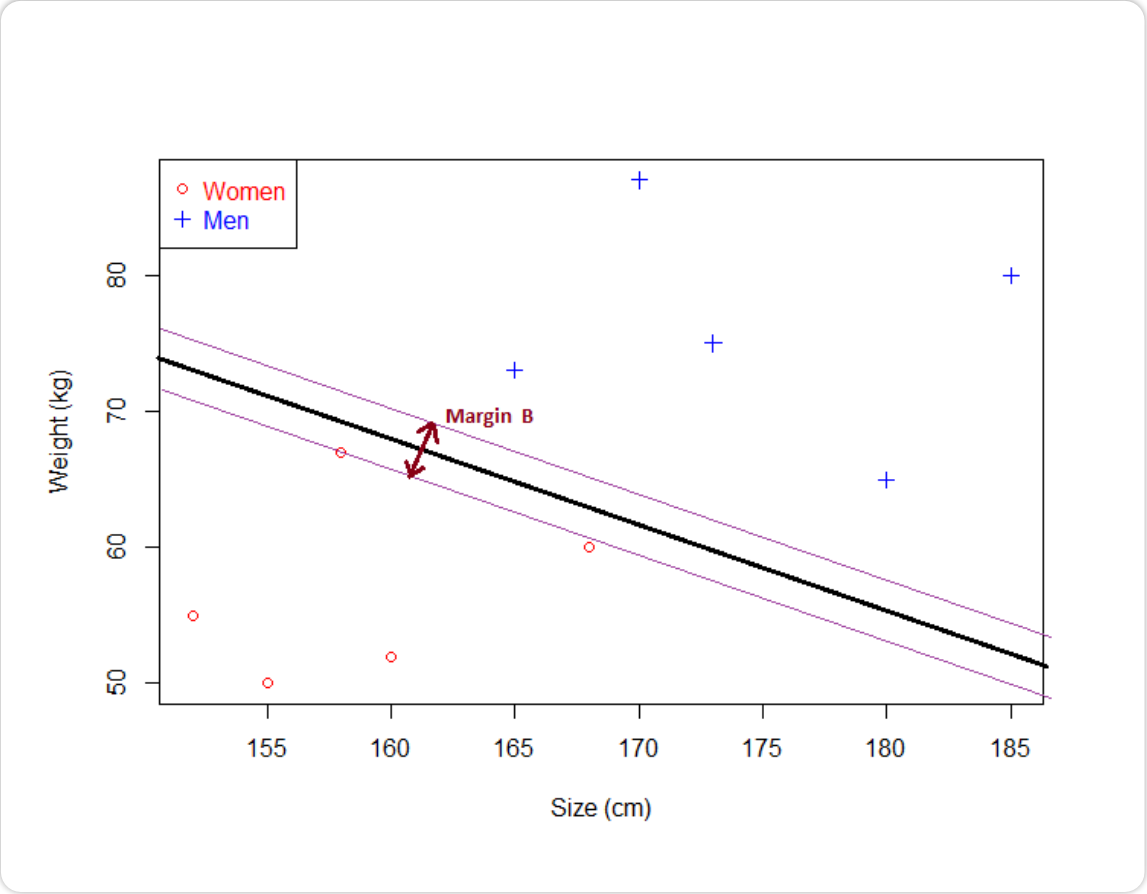
Hình 6.5: Hyperplane trong không gian hai chiều (đường thẳng) và ba chiều (mặt phẳng)

Với phương trình đường thẳng $y = ax + b$, ta có thể viết lại thành $x_2 = ax_1 + b$ hay $ax_1 - x_2 + b = 0$. Nếu đặt hai vector $x = (x_1, x_2)$ và $w = (a, -1)$ thì phương tình đường thẳng trở thành phương trình (3.6) . Đây cũng chính là phương trình chung của hyperplane trên không gian bất kỳ.

$$wx + b = 0 \tag{3.6}$$

**Margin**

Margin là một thuật ngữ quan trọng trong SVM chỉ khoảng cách giữa siêu phẳng đến 2 điểm dữ liệu gần nhất tương ứng với các phân lớp (Hình 6.6). Để có được bộ phân lớp tốt nhất, SVM cố gắng maximize margin này, từ đó thu được một siêu phẳng tạo khoảng cách xa nhất với hai lớp, nhờ vậy có thể giảm thiểu việc phân lớp sai (misclassification) đối với điểm dữ liệu mới đưa vào.




Hình 6.6: Margin được maximize để giảm thiểu phân lớp sai cho dữ liệu mới

# TÀI LIỆU THAM KHẢO


[1] J. Xue, H. Zhang, K. Dana and K. Nishino, "Differential angular imaging for material recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] H. Zhang, K. Dana and K. Nishino, "Reflectance hashing for material recognition," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, 2015.

[3] S. Bell, P. Upchurch, N. Snavely and K. Bala, "Material recognition in the wild with the materials in context database (supplemental material)," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] M. a. N. P. Saadat, "Industrial applications of automatic manipulation of flexible materials," *Industrial Robot: An International Journal*, vol. 29, pp. 434--442, 2002.

[5] Y.-K. a. O. S.-Y. Na, "Hybrid control for autonomous mobile robot navigation using neural network based behavior modules and environment classification," *Autonomous Robots*, vol. 15, pp. 193--206, 2003.

[6] L. Sharan, R. Rosenholtz and E. Adelson, "Material perception: What can you see in a brief glance?," *Journal of Vision*, vol. 9, pp. 784-784, 2009.

[7] A. Davis, K. L. Bouman, J. G. Chen, M. Rubinstein, F. Durand and W. T. Freeman, "Visual vibrometry: Estimating material properties from small motion in video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[8] P. Saponaro, S. Sorensen, A. Kolagunda and C. Kambhamettu, "Material classification with thermal imagery.," in *CVPR*, 2015.

[9] K. Tanaka, Y. Mukaigawa, T. Funatomi, H. Kubo, Y. Matsushita and Y. Yagi, "Material Classification using Frequency-and Depth-Dependent Time-of-Flight Distortion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[10] W. Yuan, S. Wang, S. Dong and E. Adelson, "Connecting look and feel: Associating the visual and tactile properties of physical materials," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR17), Honolulu, HI, USA*, 2017.

[11] C. Liu, L. Sharan, E. H. Adelson and R. Rosenholtz, "Exploring features in a bayesian framework for material recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010.

[12] D. Hu, L. Bo and X. Ren, "Toward Robust Material Recognition for Everyday Objects.," in *BMVC*, 2011.

[13] G. F. Luger, "Artificial intelligence: structures and strategies for complex problem solving," *Scalable Computing: Practice and Experience,* vol. 9, 2005.

[14] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[15] S. a. M. H. a. D. W. J. Han, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[16] L. a. G. S. a. B. S. Nanni, "Handcrafted vs. non-handcrafted features for computer vision classification," *Pattern Recognition,* vol. 71, pp. 158--172, 2017.

[17] H. a. C.-R. A. a. B. A. a. G. H. a. S. N. a. F. M. a. T. J. a. G. F. a. M. A. Wang, "Cascaded ensemble of convolutional neural networks and handcrafted features for mitosis detection," in *International Society for Optics and Photonics*, 2014.

[18] X. a. Z. J. a. H. K. a. S. J. Zhang, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence,* vol. 38, pp. 1943--1955, 2016.

[19] J. DeGol, M. Golparvar-Fard and D. Hoiem, "Geometry-informed material recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.