TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN - XỬ LÝ ẢNH

NHẬN DIỆN CÁC LOÀI HOA

Giảng viên hướng dẫn: TS. HUỲNH HỮU HƯNG

Sinh viên thực hiện: Nguyễn Văn Hoàng Phúc
Nguyễn Văn Mạnh
Nguyễn Công Cường

Nhóm: 20Nh10

Lớp: 20T1

Table of contents

01. Giới thiệu

Đề tài nhận diện các loài hoa là một trong những lĩnh vực quan trọng của thị giác máy tính và trí tuệ nhân tạo. Mục tiêu của đề tài này là xây dựng mô hình tự động có khả năng phân loại loài hoa từ hình ảnh, giúp giảm bớt công sức và thời gian cho việc nhận diện và phân loại các loài hoa trong các ứng dụng thực tiễn.

02. Công việc liên quan

01.Phát triển của Convolutional Neural Networks (CNNs)

CNNs đã cải thiện đáng kể hiệu suất của các hệ thống nhận dạng hình ảnh trên nhiều tác vụ khác nhau như phân loại hình ảnh, nhận diện vật thể, và phát hiện vùng quan trọng.

02.Sự xuất hiện của Deep Learning và Transfer Learning

Deep learning mở ra khả năng xử lý tập dữ liệu lớn và phức tạp. Transfer learning, với việc sử dụng mô hình đã được huấn luyện trước và điều chỉnh lại cho tác vụ cụ thể => cải thiện hiệu xuất.

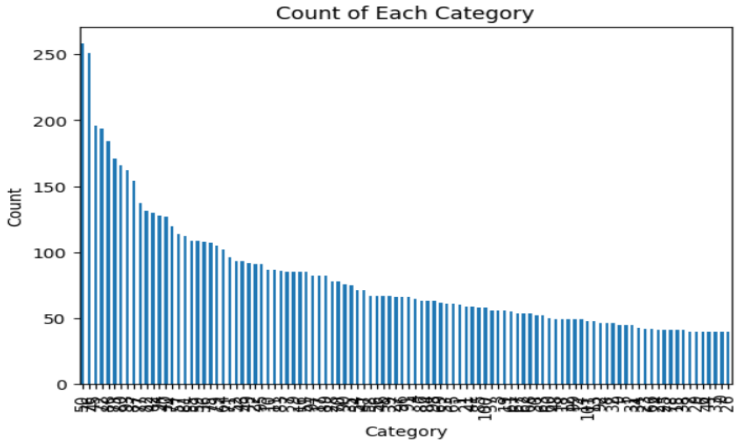
03. Sự bùng nổ của các kiến trúc dựa trên Transformer

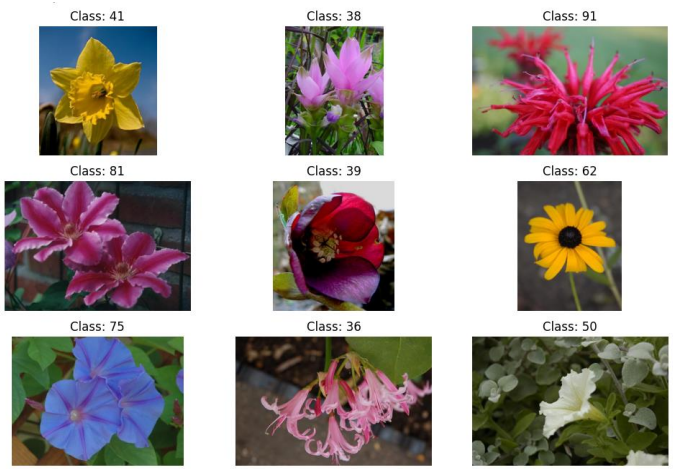
Transformer được áp dụng chủ yếu trong xử lý ngôn ngữ tự nhiên (NLP), đã được mở rộng để áp dụng cho các tác vụ nhận dạng hình ảnh và mang lại kết quả ấn tượng trong việc xử lý hình ảnh.

04. Sự phát triển của bộ dữ liệu lớn và pre-trained models

ImageNet là bộ dữ liệu siêu lớn**.** Các mô hình như BERT và GPT đã được chứng minh là hiệu quả khi được áp dụng trong image recognition sau khi được huấn luyện trước trên dữ liệu lớn.

03. Dữ liệu

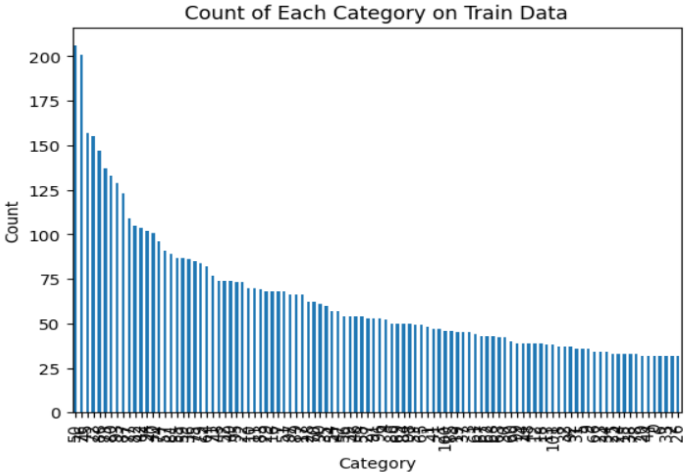


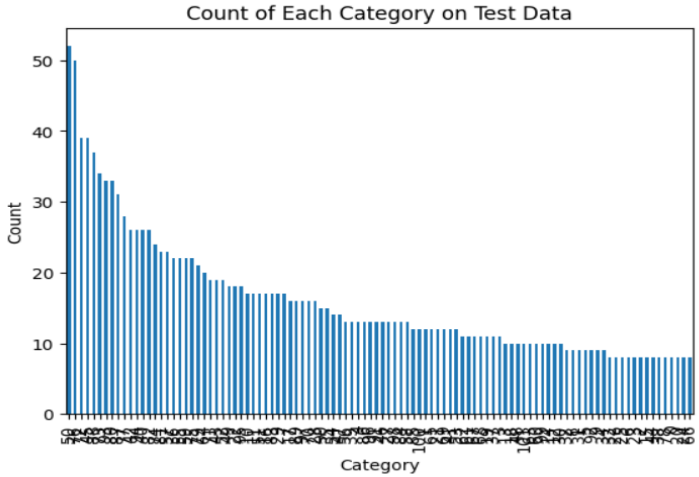


Tập dữ liệu **Oxford 102 Flower**

- Oxford 102 Flower là bộ dữ liệu phân loại hình ảnh bao gồm 102 loại hoa. Những loài hoa được chọn làm hoa thường xuất hiện ở Vương quốc Anh. Mỗi lớp bao gồm từ 40 đến 258 hình ảnh. Tổng 8190 ảnh.
- Các hình ảnh có tỷ lệ lớn, tư thế và sự thay đổi ánh sáng. Ngoài ra, có những danh mục có sự khác biệt lớn trong danh mục và một số danh mục rất giống nhau.

03. Dữ liệu





Tập dữ liệu **Oxford 102 Flower**

- Chia tập dữ liệu train:test theo tỉ lệ 8:2
- Do dữ liệu huấn luyện khá nhỏ 6551 ảnh / 102 lớp nên chúng em áp dụng các giải pháp tăng cường dữ liệu dựa trên dữ liệu gốc.
- Để tránh mất nhiều thời gian huấn luyện và lưu trữ nhưng hiệu quả vẫn tương tự, ImageDataGenerator được sử dụng, với đối tượng trên mỗi epoch tạo ra một lô dữ liệu mới từ dữ liệu gốc và áp dụng các phép biến đổi ngẫu nhiên được xác định trước

4. Giải pháp tiền xử lý

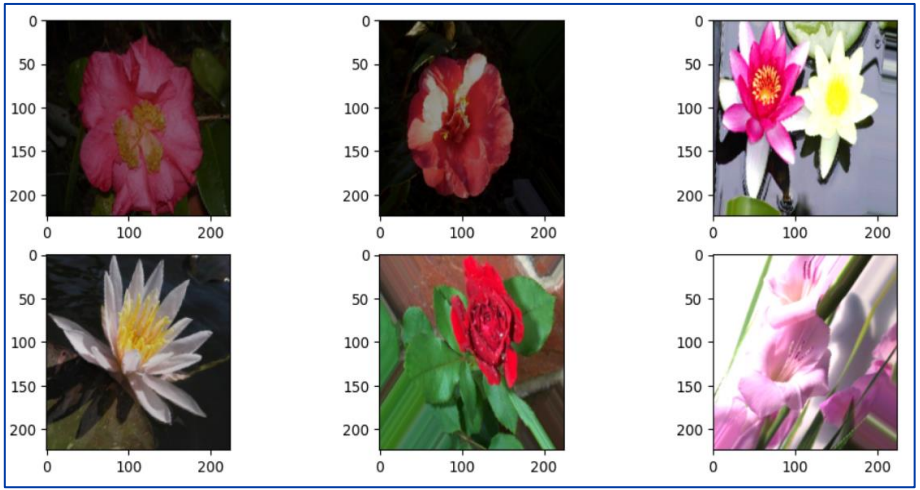
Tăng cường dữ liệu

Được thực hiện với class ImageDataGenerator có các thuộc tính sau:

- **zoom_range**: thực hiện zoom ngẫu nhiên trong một phạm vi nào đó;
- **width_shift_range**: dịch theo chiều ngang ngẫu nhiên trong một phạm vi nào đó;
- **height_shift_range**: dịch ảnh theo chiều dọc trong một phạm vi nào đó;
- **brightness_range**: tăng cường độ sáng của ảnh trong một phạm vi nào đó;
- **vertical_flip**: lật ảnh ngẫu nhiên theo chiều dọc;
- **rotation_range**: xoay ảnh góc tối đa là 45 độ;
- **shear_range**: Làm méo ảnh.

4. Giải pháp tiền xử lý

Chuẩn hoá dữ liệu



Tập dữ liệu train được thực hiện tăng cường

Tập dữ liệu test không được thực hiện tăng cường

- Thực hiện chuyển các giá trị màu sắc từ 0-255 thành số thực từ 0-1 bằng cách chia tất cả các giá trị cho 255.
- Giúp model học được những đặc trưng gần nhau hơn, ít chi phối bởi các đặc trưng quá khác biệt.

05. Giải pháp công nghệ

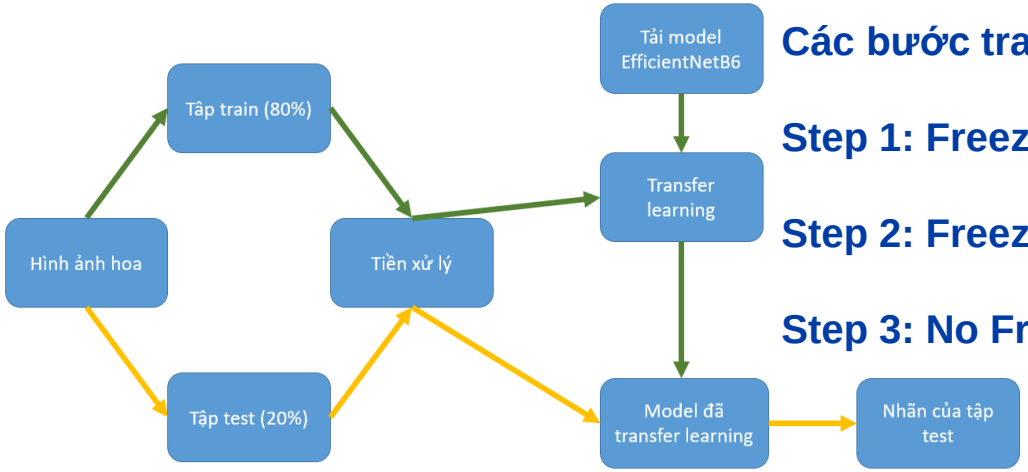
Gồm 3 giải pháp

01. Sử dụng các thuật toán CNN cơ bản và các biến thể;

02. Sử dụng kĩ thuật học chuyển giao (transfer learning);

03. Sử dụng mô hình vision transformer, transformer kĩ thuật bên NLP áp dụng vào computer vision.

Sử dụng kĩ thuật học chuyển giao (transfer learning)



Các bước transfer learning EfficientNetB6 base model:

Step 1: Freeze Base Layer + Training new Head Classifier

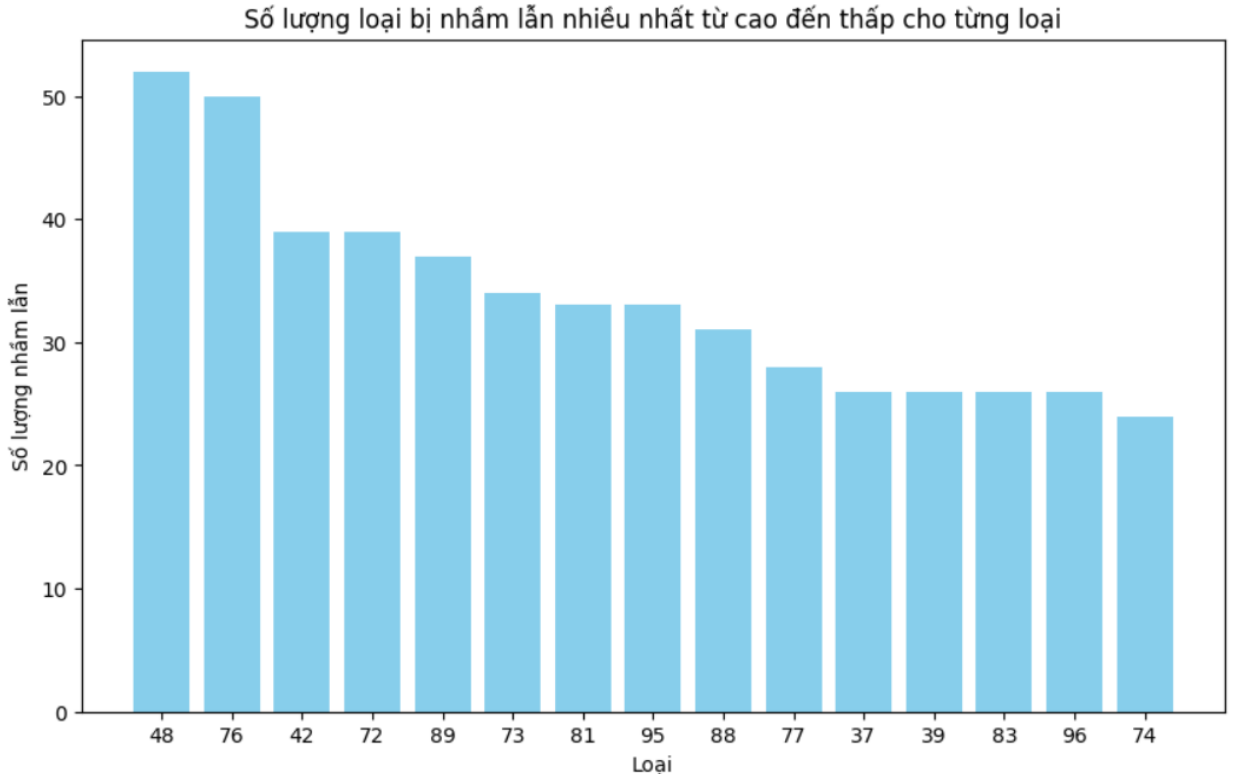
Step 2: Freeze a half layers of base_model and fine-turning

Step 3: No Freeze Base model and full fine-turning

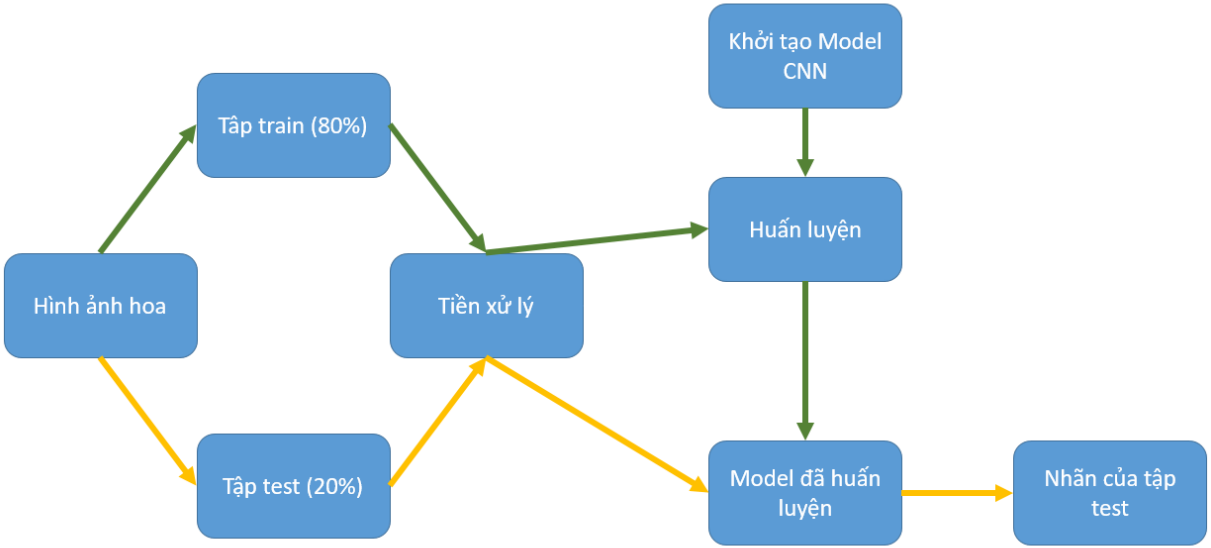
Step	Accuracy	Số epochs
1	84.50%	5
2	92.42%	5
3	92.92%	5

Chọn các lớp

```
Loại: 6 - Số lượng nhầm lẫn: 10
Loại: 7 - Số lượng nhầm lẫn: 10
Loại: 8 - Số lượng nhầm lẫn: 10
Loại: 12 - Số lượng nhầm lẫn: 10
Loại: 26 - Số lượng nhầm lẫn: 10
Loại: 45 - Số lượng nhầm lẫn: 10
Loại: 59 - Số lượng nhầm lẫn: 10
Loại: 62 - Số lượng nhầm lẫn: 10
Loại: 101 - Số lượng nhầm lẫn: 10
Loại: 27 - Số lượng nhầm lẫn: 9
Loại: 28 - Số lượng nhầm lẫn: 9
Loại: 30 - Số lượng nhầm lẫn: 9
Loại: 47 - Số lượng nhầm lẫn: 9
Loại: 80 - Số lượng nhầm lẫn: 9
Loại: 91 - Số lượng nhầm lẫn: 9
Loại: 94 - Số lượng nhầm lẫn: 9
Loại: 0 - Số lượng nhầm lẫn: 8
Loại: 9 - Số lượng nhầm lẫn: 8
Loại: 14 - Số lượng nhầm lẫn: 8
Loại: 15 - Số lượng nhầm lẫn: 8
Loại: 18 - Số lượng nhầm lẫn: 8
Loại: 19 - Số lượng nhầm lẫn: 8
Loại: 20 - Số lượng nhầm lẫn: 8
Loại: 21 - Số lượng nhầm lẫn: 8
Loại: 29 - Số lượng nhầm lẫn: 8
Loại: 34 - Số lượng nhầm lẫn: 8
Loại: 41 - Số lượng nhầm lẫn: 8
Loại: 58 - Số lượng nhầm lẫn: 8
Loại: 65 - Số lượng nhầm lẫn: 8
Loại: 78 - Số lượng nhầm lẫn: 8
```



Sử dụng các thuật toán CNN cơ bản và các biến thể



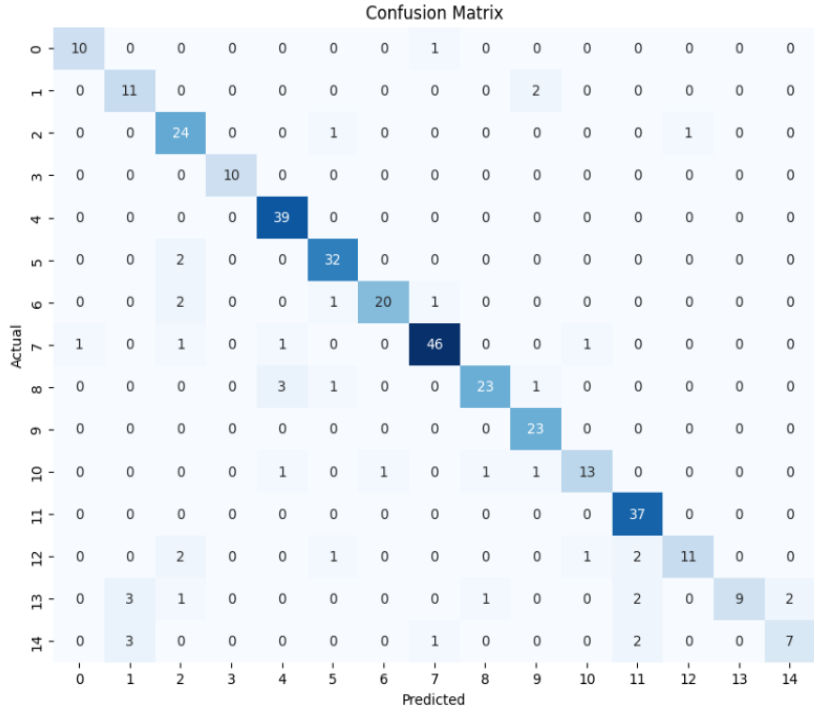
Pipeline model CNN và các biến thể

Kiến trúc model

Các kiến trúc CNN tái xây dựng và tự xây dựng:

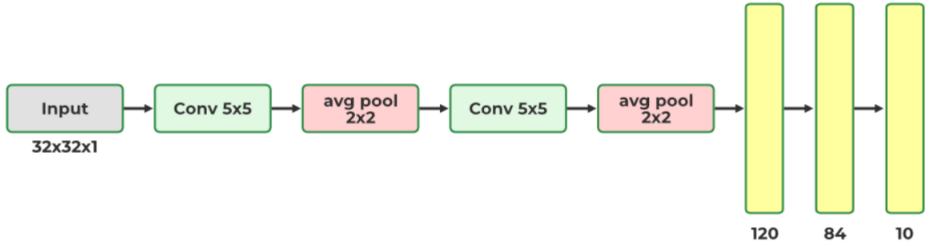
- **LeNet-5**
- **AlexNet**
- **GoogLeNet**
- **CNN tự xây dựng (Model1,2,3)**

Vì các model đơn giản độ chính xác thấp nên chúng em thực hiện giảm số lượng lớp của dữ liệu xuống 15 loài hoa.

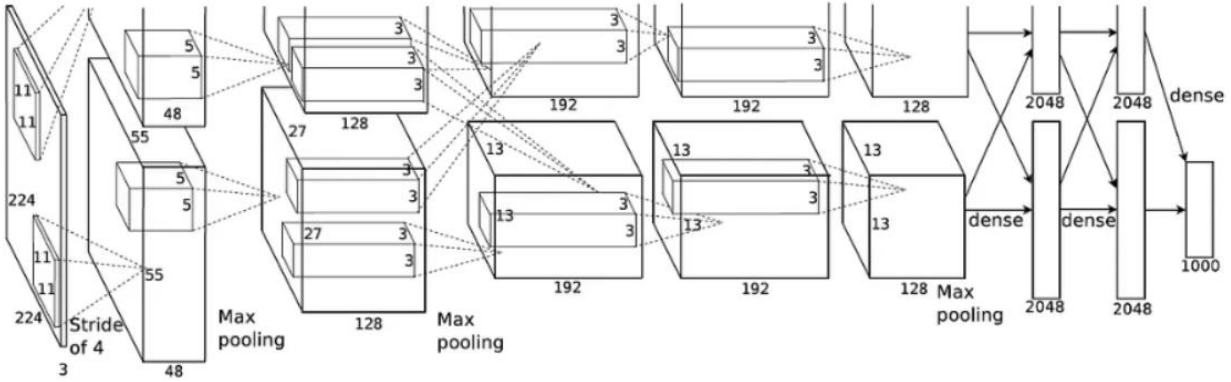


Kiến trúc model

LeNet-5



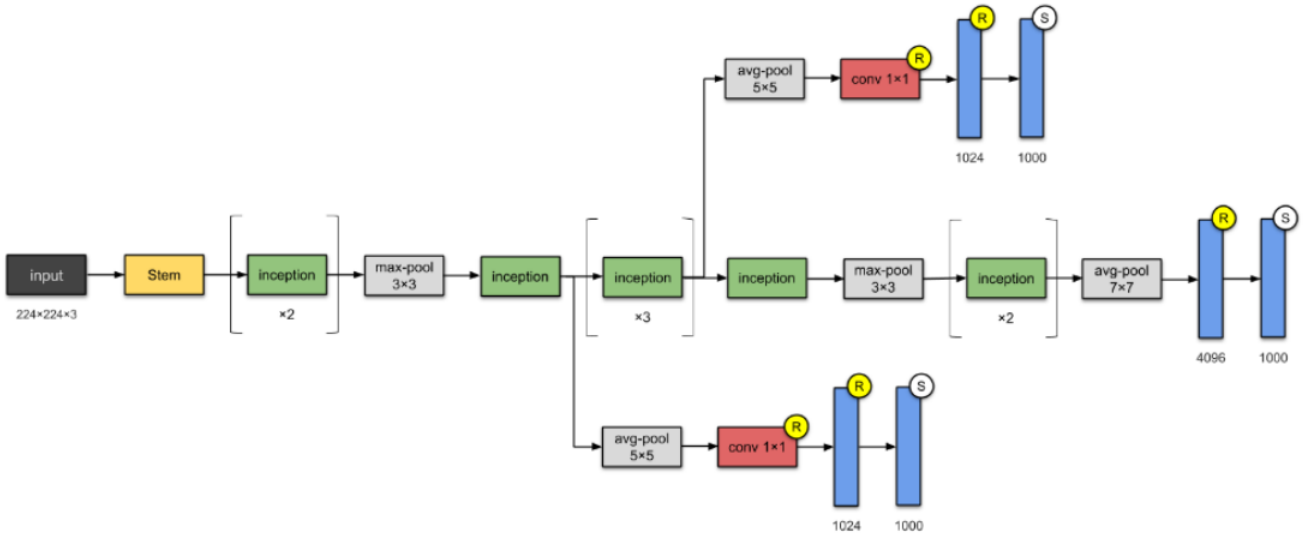
AlexNet



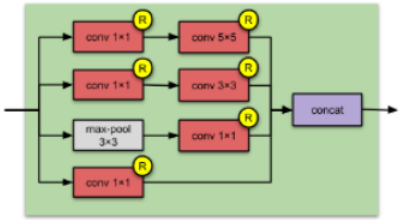
Kiến trúc AlexNet (Nó cũng bị cắt bớt từ đầu trong bài viết gốc.)

Kiến trúc model

GoogLeNet



Stem



Inception module

Kiến trúc model

```
# My model
model = M.Sequential()
model.add(L.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(L.BatchNormalization())
model.add(L.Conv2D(32, (3, 3), activation='relu'))
model.add(L.BatchNormalization())
model.add(L.MaxPooling2D((2, 2)))
model.add(L.Dropout(0.25))

model.add(L.Conv2D(64, (3, 3), activation='relu'))
model.add(L.BatchNormalization())
model.add(L.Conv2D(64, (3, 3), activation='relu'))
model.add(L.BatchNormalization())
model.add(L.MaxPooling2D((2, 2)))
model.add(L.Dropout(0.25))

model.add(L.Conv2D(128, (3, 3), activation='relu'))
model.add(L.BatchNormalization())
model.add(L.MaxPooling2D((2, 2)))
model.add(L.Dropout(0.25))

model.add(L.Flatten())
model.add(L.Dense(512, activation='relu'))
model.add(L.BatchNormalization())
model.add(L.Dropout(0.5))
model.add(L.Dense(CLASS_NUM, activation='softmax'))
```

My model 1

Kiến trúc model

```
model = M.Sequential([
    L.Conv2D(16, (5, 5), strides=(1, 1), padding='same', activation='relu', input_shape=input_shape),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),
    L.Conv2D(32, (5, 5), strides=(1, 1), padding='same', activation='relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),
    L.Conv2D(64, (5, 5), strides=(1, 1), padding='same', activation='relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),
    L.Conv2D(128, (5, 5), strides=(1, 1), padding='same', activation='relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),
    L.Flatten(),
    L.Dense(1024, activation='tanh'),
    L.Dropout(0.2),
    L.Dense(256, activation='tanh'),
    L.Dropout(0.2),
    L.Dense(CLASS_NUM, activation='softmax')
])
```

My model 2

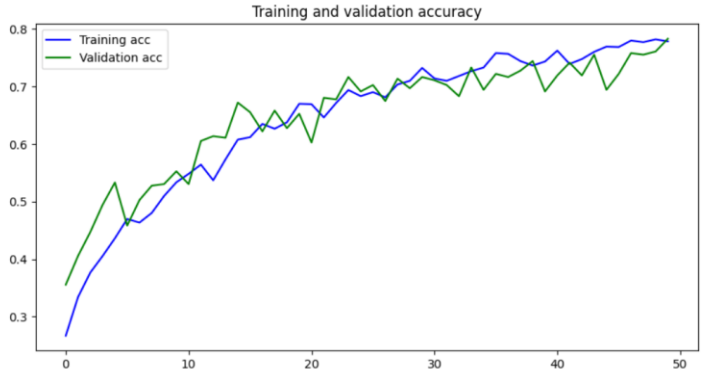
Kiến trúc model

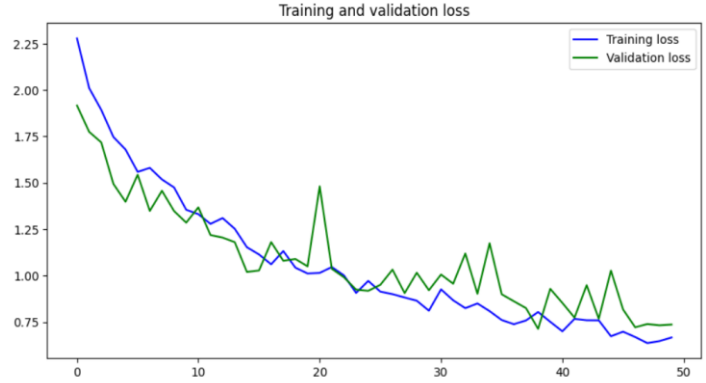
```
model = M.Sequential([
    L.Conv2D(16, (5, 5), strides=(1, 1), padding='same', input_shape=input_shape),
    L.BatchNormalization(),
    L.Activation('relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),

    L.Conv2D(32, (5, 5), strides=(1, 1), padding='same'),
    L.BatchNormalization(),
    L.Activation('relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),

    L.Conv2D(64, (5, 5), strides=(1, 1), padding='same'),
    L.BatchNormalization(),
    L.Activation('relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),

    L.Conv2D(128, (5, 5), strides=(1, 1), padding='same'),
    L.BatchNormalization(),
    L.Activation('relu'),
    L.MaxPooling2D((2, 2), strides=(2, 2), padding='valid'),

    L.Flatten(),
    L.Dense(1024, activation='relu'),
    L.Dropout(0.2),
    L.Dense(256, activation='relu'),
    L.Dropout(0.2),
    L.Dense(CLASS_NUM, activation='softmax')
])
```

My model 3

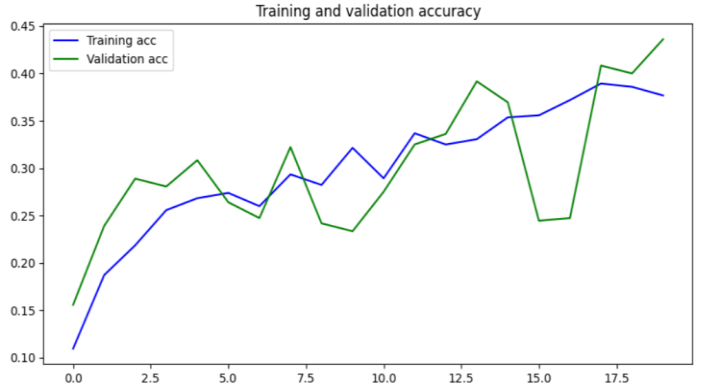
Quá trình huấn luyện

My model 2



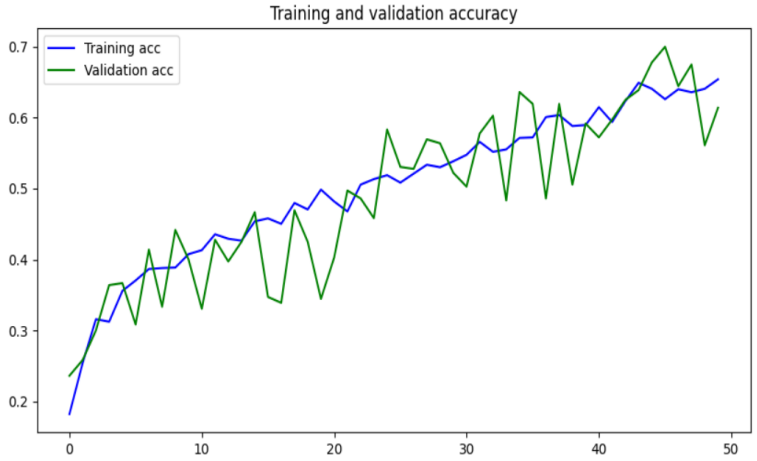


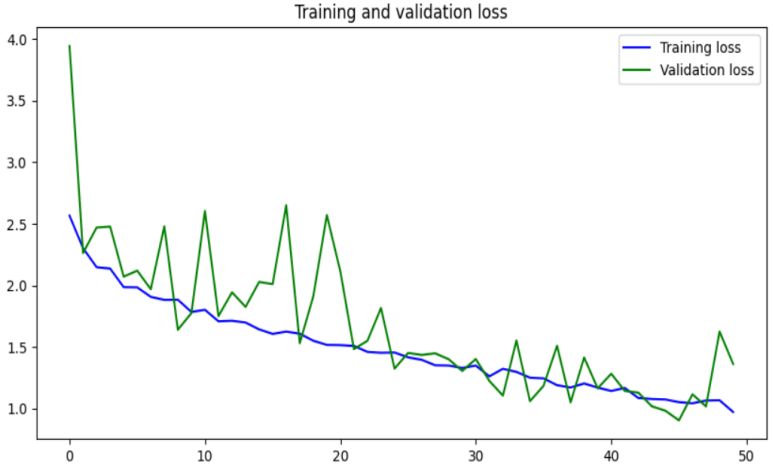
My model 3



Quá trình huấn luyện

GooLeNet





Kết quả các mô hình CNN

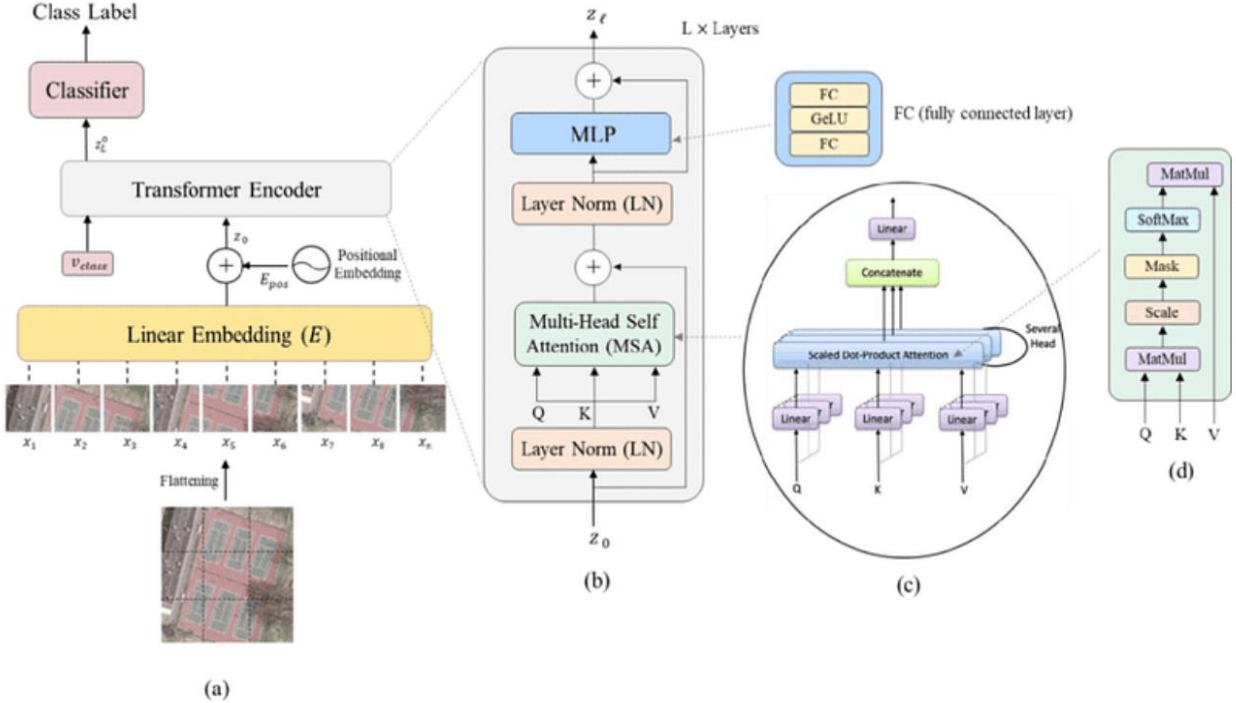
Model	Accuracy	Loss: CrossEntropy	Params
LeNet (Có tinh chỉnh)	76.11% (55e)	0.8511	30,561,199
AlexNet	49.17% (20e)	1.3552	28,878,127
GooLeNet (Có tinh chỉnh)	**89.72% (100e)**	**0.3849**	**6,260,687**
My model 1 (5 Conv + 2 Dense + kernel 3*3) Yes BatchNorm	76.42% (67e)	0.8726	51,531,183
My model 2 (4 Conv + 2 Dense + kernel 5*5) No BatchNorm	79.17% (64e)	0.6839	30,028,719
My model 3 (4 Conv + 2 Dense + kernel 5*5) Yes BatchNorm	78.33% (58e)	0.7257	30,029,679

Kinh nghiệm rút ra

Kinh nghiệm tinh chỉnh tham số và mô hình:

- **Các hàm activation trong các lớp tích chập không được khác nhau;**
- **Hàm activation trong các lớp tích chập và hàm activation trong các lớp fully connected khác nhau có thể sẽ hiệu quả hơn;**
- **Các lớp BatchNorm sẽ làm tăng tốc độ hội tụ nhưng sẽ gây biến động kết quả huấn luyện trên tập validation;**
- **Các lớp dropout giảm overfitting;**
- **Các lớp dropout nên đặt sau các bước biến đổi quan trọng (sau các khối conv, hoặc sau khi flatten);**
- **Sử dụng các khối inception module để học ở các mức độ kernel khác nhau.**

Sử dụng mô hình vision transformer



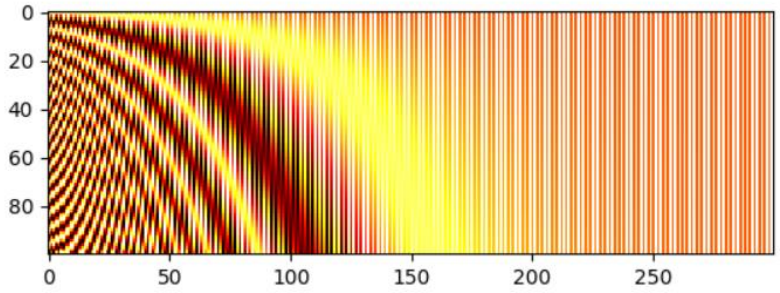
Kiến trúc Vision transformer

Những thay đổi và đề xuất

➢ Position embedding:

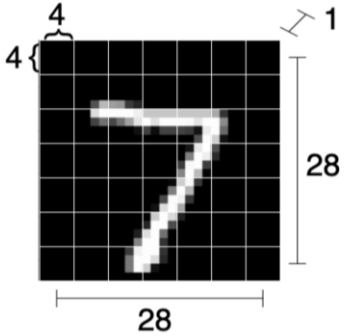
Trong bài báo chia sẻ kiến trúc nhúng embedding 1D, chúng em đề xuất nhúng embedding 2D như sau:

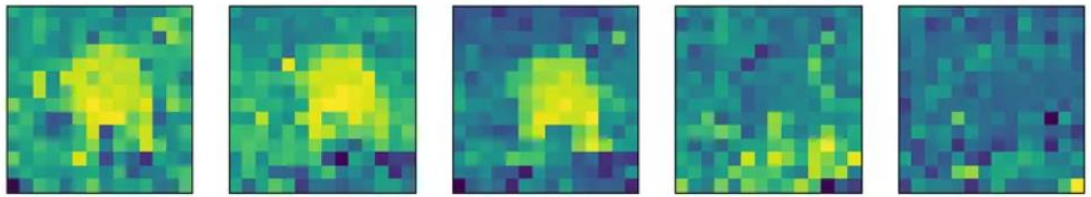
$$p_{i,j} = \begin{cases} \sin\left(\dfrac{i}{10000^{\frac{j}{d_{emb\_dim}}}}\right) & if\ j\ is\ even \\ \cos\left(\dfrac{i}{10000^{\frac{j-1}{d_{emb\_dim}}}}\right) & if\ j\ is\ odd \end{cases}$$



➢ Sử dụng CNN ở bước đầu tiên thay cho Patchifying

Patchifying





Sử dụng CNN

Kết quả huấn luyện ViT

Kiến trúc model	Accuracy
patchify bằng cách cắt hình + function position 2D	10.32%
patchify bằng convolution + position 1D (bài báo gốc)	15.45%
patchify bằng convolution + function position 2D	25.55%

- ViT đạt hiệu quả cao trên các bộ dữ liệu lớn, do dữ liệu hoa còn khá nhỏ, dễ dẫn đến việc học các đặc trưng phức tạp hơn của mô hình ViT bị hạn chế (được xem là ưu thế của phương pháp này). Do đó các cải tiến tiếp theo cần tập trung vào tập dữ liệu.

06. Kết luận và hướng phát triển

Kết quả đạt được

- Hiệu quả cao trên model GoogLeNet tái xây dựng là *89.72%*.
- Rút ra các kết luận và kinh nghiệm trong quá trình tinh chỉnh tham số.
- Độ chính xác cao nhất trên tập hoa khi thực hiện transfer learning model efficientNetB6 là *92.92%*.
- Tìm hiểu và tái xây dựng ViT nhưng kết quả vẫn còn hạn chế.

Hướng phát triển

- Đối với GoogLeNet thực hiện train trên toàn bộ dữ liệu hoa và ImageNet để học được các đặc trưng mới hơn.
- Tiếp tục phát triển và huấn luyện ViT trên dữ liệu lớn.
- Tìm hiểu các mô hình multi-task huấn luyện trên dữ liệu đa dạng và adaptor sang bộ dữ liệu hoa.

07

Demo

08

Q&A