ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

BÀI TẬP MÔN NHẬN DẠNG

NEURAL NETWORK

Người thực hiện 20120547 – Võ Thành Phong

Thành phố Hồ Chí Minh – 5/2023

MỤC LỤC

I. Thông tin cá nhân:

Sinh viên: Võ Thành Phong

Mã số sinh viên: **20120547**

Email: 20120547@student.hcmus.edu.vn

II. Yêu cầu kĩ thuật:

- File Source code ở dạng .ipynb được khuyến khích chạy trên môi trường **google colab.**

- Do có sử dụng mô hình deep learning xử lý dữ liệu dạng hình ảnh, nên số lượng tham số rất lớn và thời gian training sẽ có thể tiêu tốn nhiều thời gian, do đó khi chạy file .ipynb trên google colab **cần liên kết thời gian chạy với GPU** của google colab cho việc training mô hình deep learning.

- Do tránh tình trạng đường dẫn cứng (đường dẫn tĩnh) trong file notebook nên việc tải file và xuất file, tải thư viện sẽ được chạy cục bộ trên phiên làm việc của notebook trên google colab, do đó sau khi chạy xong foile notebook trên google colab thì **trước khi tắt file google colab hãy vào mục folder bên phải để tải các file submission cũng như models về máy, do khi tắt file notebook khỏi google colab thì lần mở lại sau sẽ không còn file đã xuất ra nữa mà phải chạy lại từ đầu file notebook đó một lần nữa trên google colab.**

III. Giới thiệu bài toán:

- Nhận dạng khuôn mặt (Face detection) là một bài toán quen thuộc và có nhiều ứng dụng trong lĩnh vực nhận dạng cũng như thị giác máy tính. Nhận dạng khuôn mặt là quá trình nhận vào một hình ảnh khuôn mặt hoặc trích xuất khuôn mặt từ hệ thống thời gian thực nào đó và nhận biết xem khuôn mặt đó là của cá nhân nào.

- Một số ứng dụng của nhận dạng khuôn mặt: Dùng trong các hệ thống nhận dạng sinh trắc học, các hệ thống giám sát và an ninh, giao diện người-máy, điểm danh tự động, …. Và bối cảnh của bài tập lần này là xây dựng một hệ thống tự động điểm danh giúp tạo ra một trải nghiệm thuận tiện cho cả sinh viên và giảng viên. Sinh viên không cần phải đứng xếp hàng để điểm danh và có thể tiết kiệm thời gian chờ đợi. Đối với giảng viên, quá trình điểm danh tự động giúp họ tập trung vào việc dạy và quản lý lớp học một cách hiệu quả hơn.

- Tuy nhiên tồn tại nhiều khó khăn khi làm việc với khuôn mặt mà các mô hình học máy thông thường khó có thể giải quyết như:

  + Biến đổi và đa dạng của khuôn mặt: Khuôn mặt có nhiều biến đổi và đa dạng trong góc nhìn, tỉ lệ, ánh sáng, biểu cảm, cấu trúc hình dạng và các yếu tố khác. Điều này làm cho việc nhận dạng khuôn mặt trở nên khó khăn, đặc biệt là với các mô hình học máy thông thường có khả năng hạn chế trong việc tổng quát hóa và xử lý các biến đổi này.

  + Lượng dữ liệu cần là rất lớn.

  + Dễ bị overfitting.

+ Tốc độ xử lý: Một khó khăn khác là tốc độ xử lý khi áp dụng các mô hình học máy cho nhận dạng khuôn mặt trong thời gian thực. Việc xử lý ảnh và tính toán các đặc trưng của khuôn mặt có thể đòi hỏi tài nguyên tính.

- Do đó **Neural Network sẽ là lựa chọn tối ưu để giải quyết bài toàn lần này**. Và bài tập này cũng sẽ tập trung xây dựng kiến trúc mạng mà cụ thể là **kiến trúc CNN và một vài kiến trúc liên quan CNN** để giải quyết bài toán nhận dạng khuôn mặt.

IV. Xây dựng bộ dữ liệu:

1. Giới thiệu bộ dữ liệu ban đầu

- Bộ dữ liệu có tên là Face Dataset được cung cấp trên trang web Kaggle thỏa giấy phép sử dung CC0: Public Domain.

- Link truy cập bộ dữ liệu: https://www.kaggle.com/datasets/vasukipatel/face-recognition-dataset

- Bộ dữ liệu chứa các ảnh là các khuôn mặt của 31 người nổi tiếng trên thế giới, khi tải về từ kaggle và giải nén sẽ có các thư mục và file như sau:

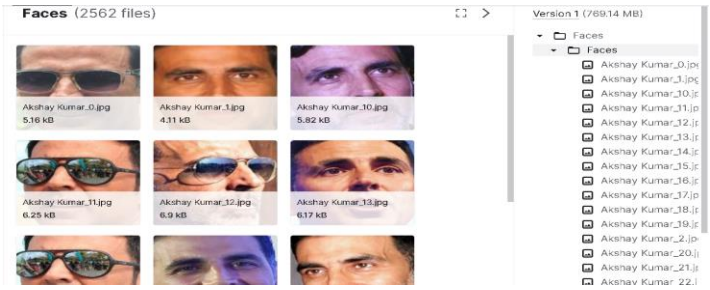
+ File ‘Dataset.csv’: file này chứa thông tin tên ảnh cũng như nhân vật mà ảnh đó thuộc về.

C1

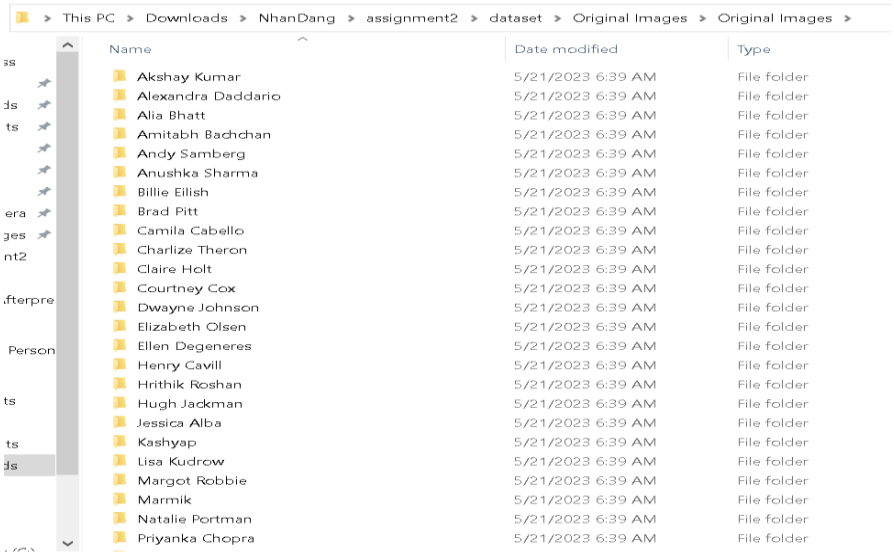
	A	B
1	id	label
2	Robert Downey Jr_87.jpg	Robert Downey Jr
3	Lisa Kudrow_64.jpg	Lisa Kudrow
4	Ellen Degeneres_34.jpg	Ellen Degeneres
5	Billie Eilish_3.jpg	Billie Eilish
6	Hrithik Roshan_35.jpg	Hrithik Roshan
7	Vijay Deverakonda_39.jpg	Vijay Deverakonda
8	Tom Cruise_21.jpg	Tom Cruise
9	Alia Bhatt_41.jpg	Alia Bhatt
10	Elizabeth Olsen_36.jpg	Elizabeth Olsen
11	Charlize Theron_29.jpg	Charlize Theron
12	Natalie Portman_74.jpg	Natalie Portman
13	Priyanka Chopra_6.jpg	Priyanka Chopra
14	Priyanka Chopra_73.jpg	Priyanka Chopra
15	Natalie Portman_99.jpg	Natalie Portman
16	Zac Efron_6.jpg	Zac Efron
17	Marmik_5.jpg	Marmik
18	Camila Cabello_9.jpg	Camila Cabello
19	Elizabeth Olsen_23.jpg	Elizabeth Olsen
20	Claire Holt_28.jpg	Claire Holt

Dataset

+ Folder Faces: Trong folder này lại chứa một folder khác cũng có tên là ‘Faces’ chứa 2562 ảnh cho 31 người nổi tiếng trên thế giới. **Và ta sẽ sử dung folder này kết hợp với file ‘Dataset.csv’ để tiến hành tạo dữ liệu phù hợp cho bài toán.**



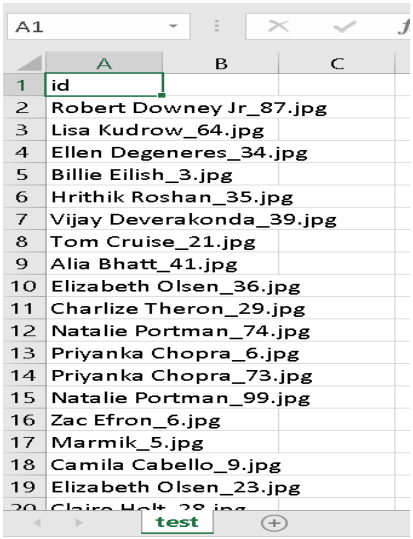
+ Cuối cùng là folder có tên ‘Original Images’, ta sẽ không quan tâm đến folder này vì nó thật ra là chứa những folder cho từng người nổi tiếng đã được sắp xếp lại ảnh đúng với người đó.



2. Thiết kế lại bộ dữ liệu cho quá trình huấn luyện và đánh giá

- Tạo file train.csv và file test.csv: Dùng file ‘Dataset.csv’ để tạo hai file vừa đề cập như sau:

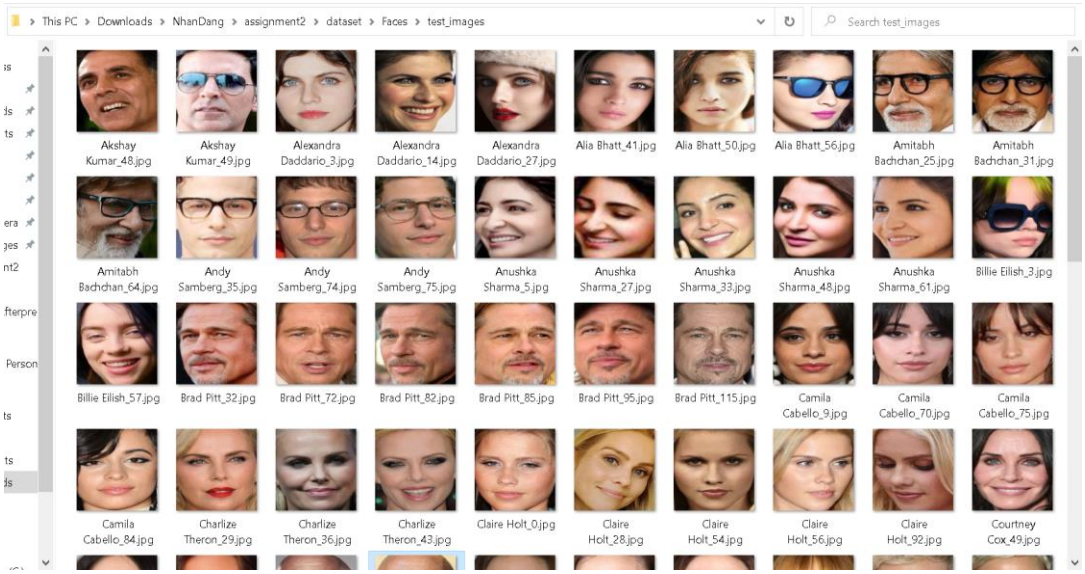
+ Đối với file ‘test.csv’: Lấy 100 dòng đầu của file ‘Datset.csv’ để làm file ‘test.csv’, tất nhiên sẽ xóa đi cột label và chỉ giữ lại cột id (tức là cột tên ảnh), em cũng đã tiến hành kiểm tra khi trong 100 dòng đầu này có đủ tên ảnh của tất cả 31 người trong bộ dữ liệu.



+ Đối với file ‘train.csv’: Chỉ việc dùng file ‘Dataset.csv’ và xóa đi 100 dòng được dùng làm file ‘test.csv’, tất nhiên giữ nguyên cả 2 cột id và label cho quá trình huấn luyện.

- Tạo 2 folders ảnh riêng biệt cho việc train và test: Dùng folder Faces ban đầu để tạo 2 folders ‘train_images’ và ‘test_images’:

+ Đối với folder ‘test_images’: Từ folder ‘Faces’ ban đầu, em sẽ tìm đúng các ảnh theo tên trong file ‘test.csv’ và di chuyển chúng sang folder ‘test_images’.



+ Đối với folder ‘train_images’: Từ folder ‘Faces’ ban đầu sau khi đã loại hết các ảnh cho việc test thì những ảnh còn lại trong folder ‘Faces’ sẽ là ảnh cho folder ‘train_images’.

- Link đến bộ dữ liệu đã được tái cấu trúc: https://drive.google.com/drive/folders/1nzu3roXcv3nMaXEqGu0-9ERIzWFY9_Lp?usp=share_link

V. Tổng quan quá trình thiết kế mô hình

- Các mô hình để nhận dạng trên bộ dữ liệu faces dataset của em sẽ bao gồm 2 mô hình như sau:

+ Mô hình mạng CNN mà cụ thể là kiến trúc VGG16.

+ Mô hình cải tiến của CNN mà cụ thể là kiến trúc ResNet-50.

- Tổng quan quá trình xây dựng 2 mô hình này như sau:

+ Tăng cường dữ liệu với **Data Augmentation**.

+ Thiết kế một lớp Dataset phục vụ cho việc tải dữ liệu theo từng batch.

+ Xây dựng, huấn luyện, đánh giá và dự đoán trên mô hình kiến trúc VGG16.

+ Xây dựng, huấn luyện, đánh giá và dự đoán trên mô hình kiến trúc ResNet-50.

VI. Tăng cường dữ liệu với Data Augmentation

- Dữ liệu là yếu tố rất quan trọng đối với việc xây dựng mô hình học sâu, do đó khi số lượng dữ liệu quá ít và có thể làm giảm độ chính xác của mô hình thì việc tìm cách tăng số lượng dữ liệu là đều bắt buộc.

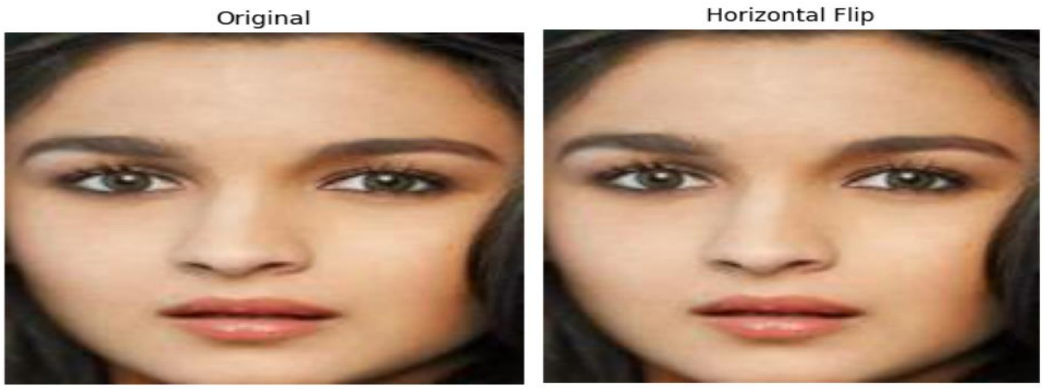
- Dữ liệu là yếu tố rất quan trọng đối với việc xây dựng mô hình học sâu, do đó khi số lượng dữ liệu quá ít và có thể làm giảm độ chính xác của mô hình thì việc tìm cách tăng số lượng dữ liệu là đều bắt buộc.

- Tuy nhiên việc tìm kiếm dữ liệu là không dễ dàng và có thể tốn chi phí, do đó một trong nhưng phương pháp rất dễ thực hiện và được áp dụng rất thường xuyên là data augmentation.

- Data augmentation cho hình ảnh là cách tận dụng những hình ảnh sẵn có sau đó sử dụng những kỹ thuật biến đổi hình học, thay đổi không gian màu, thêm nhiễu, xóa ngẫu nhiên…

- Các kĩ thuật tăng cường phổ biến như là: lật ảnh, xoay ảnh, cắt ảnh, thêm nhiễu, đổi hệ màu, …

```
# data_transforms cho việc tăng cường dữ liệu
training_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.Resize((224,224),antialias=True),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=30)]
)
valid_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.Resize((224,224),antialias=True)]
)
```



VII. Thiết kế lớp Dataset phục vụ việc tải dữ liệu theo từng batch

- Framework Pytorch cho phép tự thiết kế lớp Dataset để sau này tải dữ liệu theo từng batch trong quá trình huấn luyện.

```
df=pd.read_csv('./train.csv')
classes=list(set(list(df['label'].values)))
classes=sorted(classes)
```

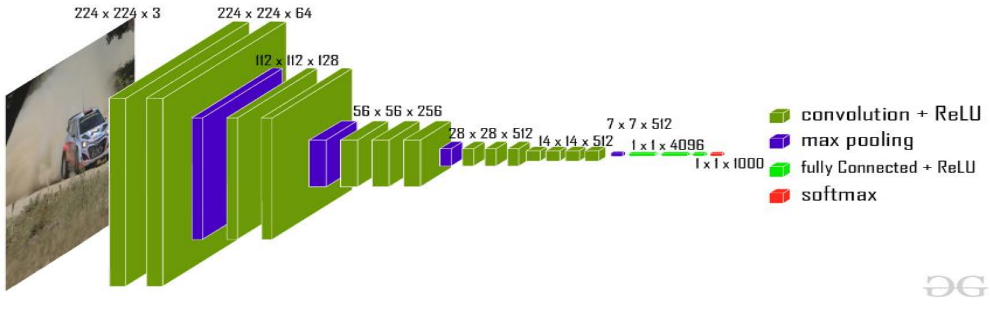
```
class MyDataset(Dataset):
    def __init__(self, file_path, df,classes,transforms=None, training=True):
        self.df = df
        self.classes=classes
        self.transforms = transforms
        self.training = training
        self.Path_to_image=file_path
    def __len__(self):
        return len(self.df)
    def __getitem__(self, idx):
        img_path = self.Path_to_image + '/' + self.df.loc[idx, 'id']
        img = Image.open(img_path).convert('RGB')
        if self.training:
            labels = self.df.loc[idx,self.df.columns[1]]
            labels = self.classes.index(labels)
            labels =torch.as_tensor(labels, dtype=torch.int8)
            if self.transforms:
                img = self.transforms(img)
            return img, labels
        else:
            if self.transforms:
                img = self.transforms(img)
            return img
        return img
```

VIII. Xây dựng kiến trúc VGG16

1. Sơ lược về lý thuyết kiến trúc VGG16

- VGG được xây dựng hầu như dựa trên tất cả các đặc điểm của mạng thần kinh tích chập (CNN) gồm các lớp chính như lớp tích chập (Convolutional layers), lớp kết nối đầy đủ (Fullyconnected layers), lớp pooling (pooling layer giúp chắt lọc thông tin). Bên cạnh đó VGG được xây dựng với bộ lọc (filters) tích chập rất nhỏ. VGG có 2 phiên bản là VGG-16 và VGG-19 tương ứng với 16 và 19 lớp tích chập (convolutional layers).

- Mạng VGG - 16 bao gồm 13 lớp convolution đều có kernel 3x3 (sau mỗi lớp convolution là max-pooling downside xuống 0.5) và 3 lớp fully connected.



VGG-16 architecture

Link ảnh: https://www.geeksforgeeks.org/vgg-16-cnn-model/

- **Lớp tích chập (Convolution layer)**: Ảnh được chuyển qua một một chồng các lớp tích chập với bộ lọc có kích thước 3x3. Stride của tích chập và padding đầu vào được cố định là 1 pixel cho các lớp tích chập 3 x 3, điều này đảm bảo rằng độ phân giải không gian được giữ nguyên sau khi tích chập. Năm lớp max-pooling 2x2 với giá trị stride là 2. Việc dùng rất nhiều heading ở mỗi lớp tích chập nhằm hy vọng rằng sau khi đi qua một lớp tích chập sẽ trích xuất được nhiều loại đặc trưng của hình ảnh mà không chỉ là một loại đặc trưng

- **Lớp ẩn (Hidden Layers)**: Các lớp ẩn trong mạng VGG đều sử dụng hàm kích hoạt ReLU để lọc các giá trị nhỏ hơn 0.

- **Lớp kết nối đầy đủ (Fully-Connected Layers)**: Sau các lớp convolution và pooling thì dữ liệu được flatten và cho vào lớp fully connected. Mô hình có 3 lớp FC như sau: hai lớp đầu tiên có 4096 đơn vị mỗi lớp và lớp thứ ba chứa 1000 đơn vị cho mỗi lớp.

- Đầu ra của lớp Fully-Connected sau này sẽ đi qua hàm Softmax để nhận được vector phân phối xác suất cho các lớp mà cụ thể đối với bài toán này là 31 lớp.

2. Cài đặt

- Xây dựng model: Bằng cách tạo một khung model vgg16 có sẵn do torch.vision cung cấp sau đó training model và lưu lại bộ tham số tối ưu nhất cho model.

- Cụ thể các lớp của VGG16 như sau:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=31, bias=True)
  )
)
```

3. Huấn luyện

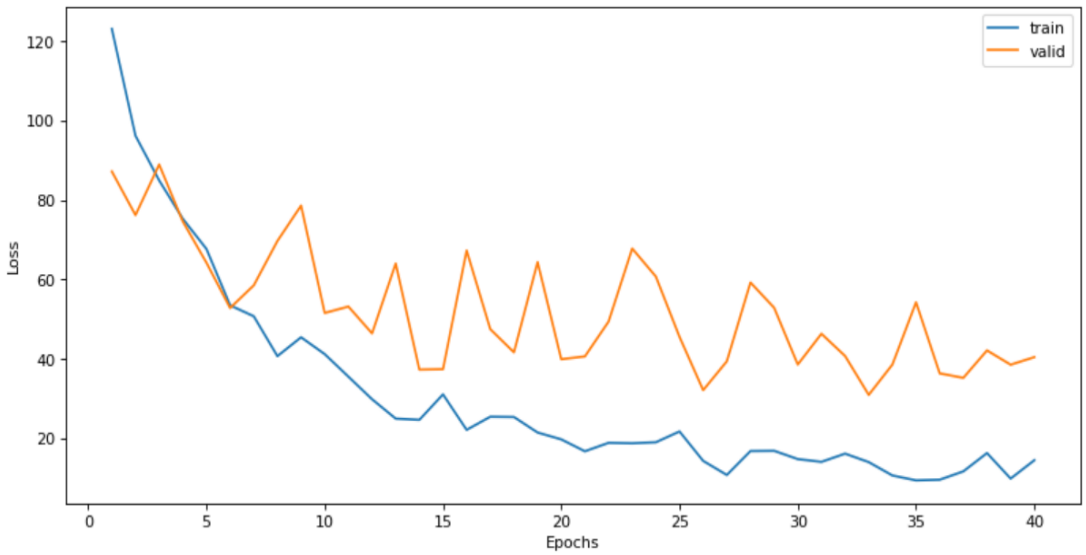
Huấn luyện mô hình VGG16:

```
[26]: history_vgg16 = train_model(model_vgg16,
                                  'vgg16',
                                  training_data_loader,
                                  valid_data_loader,
                                  critertion,
                                  optimizer_vgg16,
                                  num_epochs,
                                  device)
```

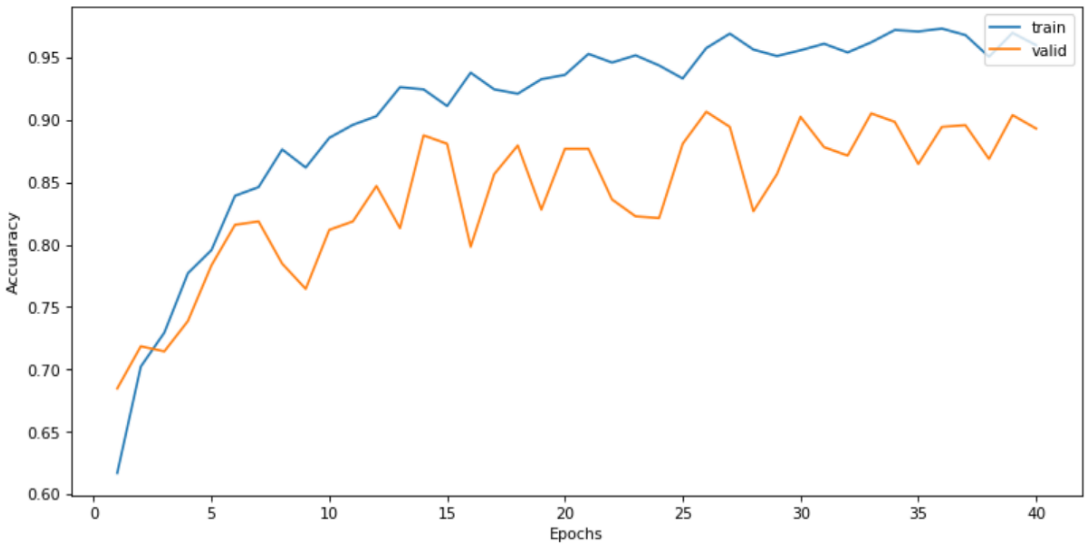
```
Epoch 1/40:
Training loss: 123.159  Accuracy: 0.617
Valid loss: 87.238      Accuracy: 0.685
Model save with 87.238 loss at Epoch 1

============================================================
Complete epoch 1/40     Training loss: 123.159     Trainning accuracy: 0.617   Valid loss: 87.238    Valid accuracy: 0.685
============================================================
Epoch 2/40:
Training loss: 96.231  Accuracy: 0.702
Valid loss: 76.249     Accuracy: 0.719
Model save with 76.249 loss at Epoch 2

============================================================
Complete epoch 2/40     Training loss: 96.231     Trainning accuracy: 0.702   Valid loss: 76.249    Valid accuracy: 0.719
============================================================
Epoch 3/40:
Training loss: 84.996  Accuracy: 0.730
Valid loss: 88.985     Accuracy: 0.714

============================================================
Complete epoch 3/40     Training loss: 84.996     Trainning accuracy: 0.730   Valid loss: 88.985    Valid accuracy: 0.714
============================================================
Epoch 4/40:
Training loss: 75.329  Accuracy: 0.777
```

4. Đánh giá

- Biểu đồ hàm chi phí:



- Biểu đồ Accuracy:



5. Dự đoán trên tập test

Nhận dạng bằng model VGG16:

```
result =pd.DataFrame()
for imgs in test_data_loader:
    imgs = imgs.to(device)
    predictions = model_vgg16(imgs)
    score = nn.Softmax(1)(predictions).detach().cpu().numpy()
    label= [classes[i] for i in np.argmax(score,axis=1)]
    output = pd.DataFrame(label, columns=['label'])
    result = pd.concat([result, output], ignore_index=True)

submission_df = result.copy()
submission_df['id'] = test_df.id
submission_df = submission_df[['id','label']]

submission_df.to_csv('submission-faces-recognition-vgg16.csv', index=False)
```

A1 | id

	A	B
1	id	label
2	Robert Downey Jr_87.jpg	Robert Downey Jr
3	Lisa Kudrow_64.jpg	Lisa Kudrow
4	Ellen Degeneres_34.jpg	Ellen Degeneres
5	Billie Eilish_3.jpg	Billie Eilish
6	Hrithik Roshan_35.jpg	Hrithik Roshan
7	Vijay Deverakonda_39.jpg	Vijay Deverakonda
8	Tom Cruise_21.jpg	Tom Cruise
9	Alia Bhatt_41.jpg	Priyanka Chopra
10	Elizabeth Olsen_36.jpg	Elizabeth Olsen
11	Charlize Theron_29.jpg	Charlize Theron
12	Natalie Portman_74.jpg	Natalie Portman
13	Priyanka Chopra_6.jpg	Priyanka Chopra
14	Priyanka Chopra_73.jpg	Priyanka Chopra
15	Natalie Portman_99.jpg	Priyanka Chopra
16	Zac Efron_6.jpg	Zac Efron
17	Marmik_5.jpg	Marmik
18	Camila Cabello_9.jpg	Camila Cabello
19	Elizabeth Olsen_23.jpg	Elizabeth Olsen
20	Claire Holt_28.jpg	Claire Holt

submission-faces-recognition-vg

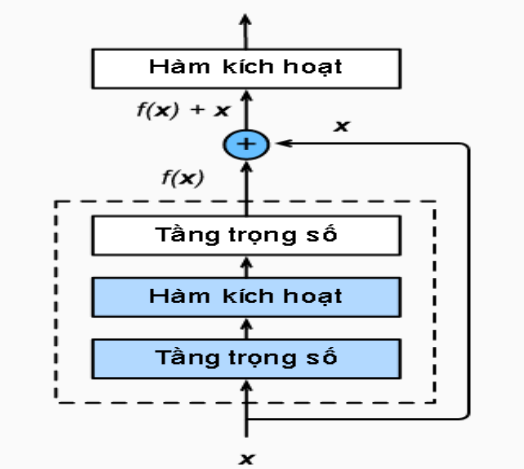
IX. Xây dựng kiến trúc ResNet-50

1. Sơ lược về lý thuyết kiến trúc ResNet-50

- Khi mạng CNN ra đời thì nó là một công cụ hiệu quả cho các tác vụ liên quan đến hình ảnh, và thường những kiến trúc CNN có tính hiệu quả cao khi số lượng lớp của nó từ ít đến trung bình. Một câu hỏi đặt ra rằng liệu nếu càng tăng số lớp cho mô hình CNN thì có đi kèm với việc hiệu suất càng tăng hay không?

- Thật không may khi câu trả lời là không. Khi càng tăng số lớp trên kiến trúc CNN cổ điển sẽ dễ dẫn đến vấn đề vanishing/exploding gradinents vì khi rất nhiều gradients quá nhỏ hoặc quá lớn được nhân lại với nhau thì các giá trị độ dốc ở những lớp càng thấp (hay nói cách khác những lớp càng sâu) sẽ bị biến mất do xấp xỉ 0 hoặc bùng nổ do có giá trị quá lớn.

- Khi đó kết nối tắt ra đời với tác dụng bổ sung thông tin của những layers trước đó cho các layers sâu hơn.



Link ảnh: https://d2l.aivivn.com/chapter_convolutional-modern/resnet_vn.html

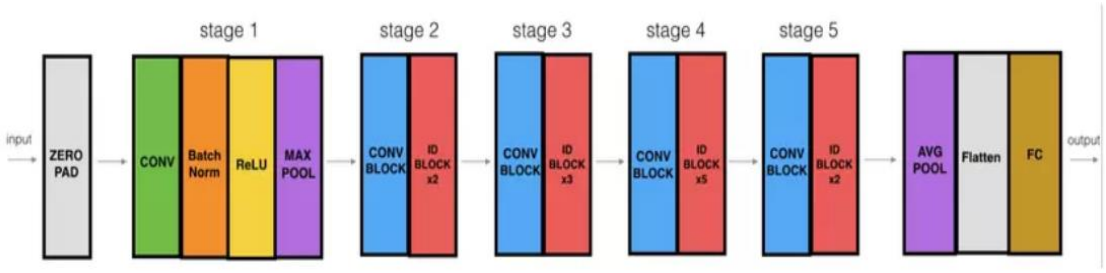
- Ý tưởng chính là: Mỗi lớp khi được thêm vào sẽ kèm theo một khối phần dư (residual block) với hy vọng rằng thông qua việc kết nối giữa đầu vào và đầu ra của lớp đó thì mô hình sẽ kết nối được nhiều thông tin hơn và hiệu quả ít nhất bằng mô hình cổ điển ban đầu khi chưa tăng độ sâu.

- Có hai loại Residual Block:

  + Identity Block (ID Block): input sẽ được giữ nguyên kích thước và thực hiện kết nối tắt.

  + Convolutional Block (Conv Block): input sẽ được thay đổi kích thước trước khi thực hiện kết nối tắt.

- **Mô hình ResNet-50:**



Link ảnh: https://viblo.asia/p/gioi-thieu-mang-resnet-vyDZOa7R5wj

+ Ở giai đoạn đầu:

Ảnh đầu vào sẽ được đi qua một lớp tích chập có kernel size lớn để có thể giữ lại thông tin càng nhiều càng tốt, sau đó được nén kích thước thông qua việc đi qua một lớp max pooling.

Kernel size thường là 7x7, cùng với multi heading sẽ giúp giữ được nhiều thông tin trên toàn bộ ảnh.

+ Tiếp theo ở giai đoạn 2, 3, 4, 5 sẽ gồm có hai loại khối residual block đã đề cập ở trên, dù loại khối nào thì trong nó cũng sẽ có 3 lớp tích chập liên tiếp nhau.

+ Giai đoạn hai sẽ gồm một khối Conv Block và hai khối ID Block xếp chồng lên nhau.

+ Giai đoạn ba sẽ gồm một khối Conv Block và ba khối ID Block xếp chồng lên nhau.

+ Giai đoạn bốn sẽ gồm một khối Conv Block và năm khối ID Block xếp chồng lên nhau.

+ Giai đoạn năm sẽ gồm một khối Conv Block và hai khối ID Block xếp chồng lên nhau.

+ Cuối cùng, sau khi đi hết 5 giai đoạn, đầu ra sẽ đi qua average pooling và flatten để đi vào lớp cuối cùng là Fully Connected.

- Tổng các lớp của 5 giai đoạn sẽ là 50 lớp do đó kiến trúc mới có tên là ResNet-50, và mọi người còn hay biết đến cái tên ResNet-52 là do tính luôn cả hai lớp ở khối cuối cùng là Average Pooling và Fully Connected.

- Sau khi ra khỏi Fully Connected thì kết quả sẽ được đi qua hàm Softmax để nhận vector phân phối xác suất tương ứng cho các lớp.

2. Cài đặt

- Xây dựng model: Bằng cách tạo một khung model resnet50 có sẵn do torch.vision cung cấp sau đó training model và lưu lại bộ tham số tối ưu nhất cho model.

- Cụ thể các lớp của ResNet-50 như sau:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=31, bias=True)
)
```

3. Huấn luyện

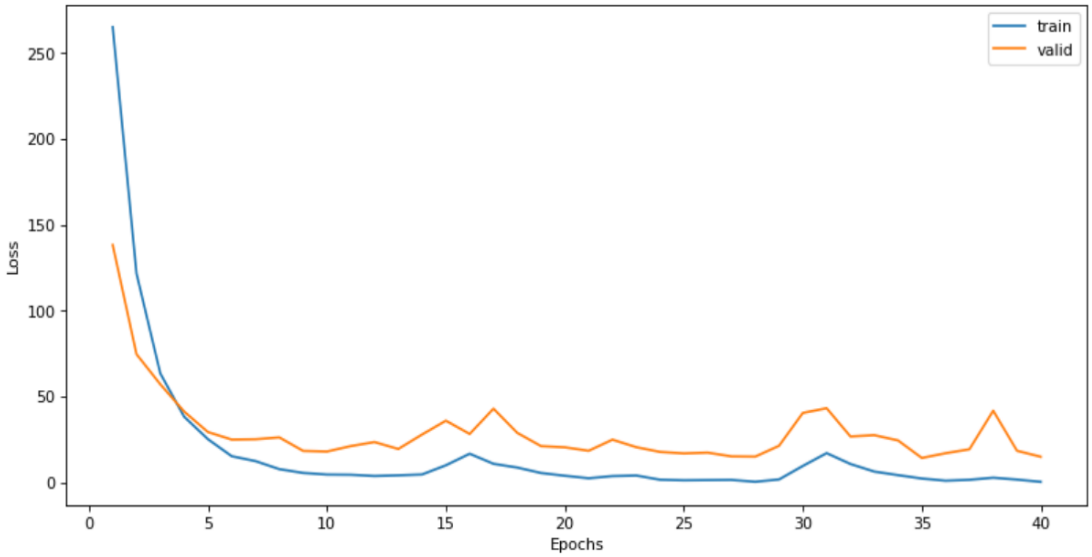
Huấn luyện mô hình RESNET50:

```
history_resnet50 = train_model(model_resnet50,
                               'resnet50',
                               training_data_loader,
                               valid_data_loader,
                               critertion,
                               optimizer_resnet50,
                               num_epochs,
                               device)
```

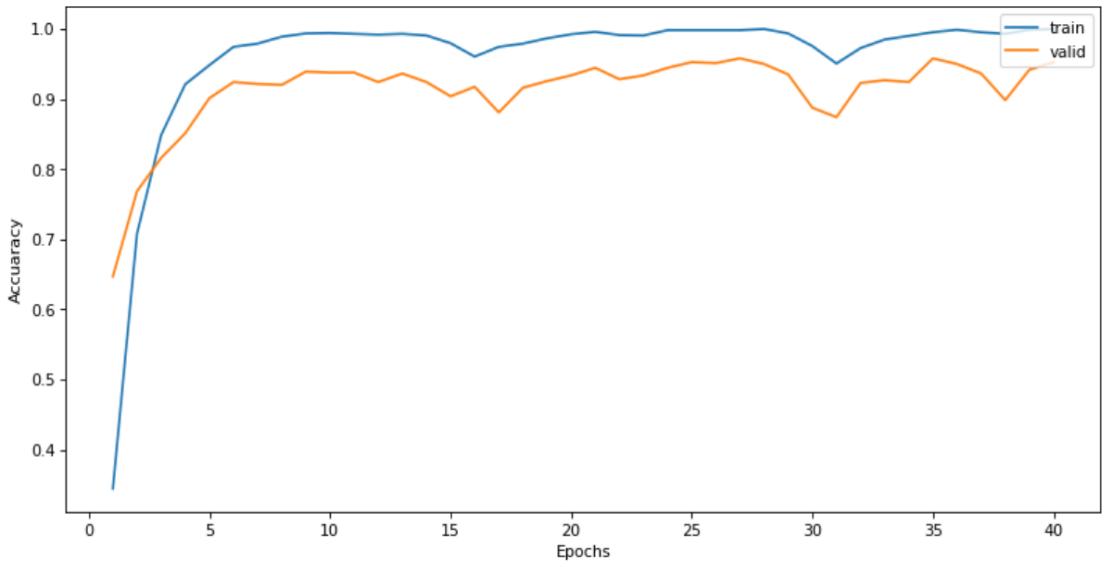
```
Epoch 1/40:
Training loss: 265.215  Accuracy: 0.344
Valid loss: 138.275     Accuracy: 0.647
Model save with 138.275 loss at Epoch 1

============================================================
Complete epoch 1/40     Training loss: 265.215     Trainning accuracy: 0.344   Valid loss: 138.275    Valid accuracy: 0.647
============================================================
Epoch 2/40:
Training loss: 121.598  Accuracy: 0.708
Valid loss: 74.653      Accuracy: 0.769
Model save with 74.653 loss at Epoch 2

============================================================
Complete epoch 2/40     Training loss: 121.598     Trainning accuracy: 0.708   Valid loss: 74.653    Valid accuracy: 0.769
============================================================
Epoch 3/40:
Training loss: 63.441  Accuracy: 0.849
Valid loss: 57.124     Accuracy: 0.816
Model save with 57.124 loss at Epoch 3

============================================================
Complete epoch 3/40     Training loss: 63.441     Trainning accuracy: 0.849   Valid loss: 57.124    Valid accuracy: 0.816
============================================================
Epoch 4/40:
Training loss: 38.292  Accuracy: 0.921
Valid loss: 41.292     Accuracy: 0.851
Model save with 41.292 loss at Epoch 4

============================================================
Complete epoch 4/40     Training loss: 38.292     Trainning accuracy: 0.921   Valid loss: 41.292    Valid accuracy: 0.851
============================================================
Epoch 5/40:
Training loss: 25.166  Accuracy: 0.948
```

4. Đánh giá

- Biểu đồ hàm chi phí:



- Biểu đồ Accuracy:



5. Dự đoán trên tập Test

Nhận dạng bằng model RESNET50:

```
result =pd.DataFrame()
for imgs in test_data_loader:
    imgs = imgs.to(device)
    predictions = model_resnet50(imgs)
    score = nn.Softmax(1)(predictions).detach().cpu().numpy()
    label= [classes[i] for i in np.argmax(score,axis=1)]
    output = pd.DataFrame(label, columns=['label'])
    result = pd.concat([result, output], ignore_index=True)

submission_df = result.copy()
submission_df['id'] = test_df.id
submission_df = submission_df[['id','label']]

submission_df.to_csv('submission-faces-recognition-resnet50.csv', index=False)
```

C1

	A	B
1	id	label
2	Robert Downey Jr_87.jpg	Robert Downey Jr
3	Lisa Kudrow_64.jpg	Lisa Kudrow
4	Ellen Degeneres_34.jpg	Ellen Degeneres
5	Billie Eilish_3.jpg	Billie Eilish
6	Hrithik Roshan_35.jpg	Hrithik Roshan
7	Vijay Deverakonda_39.jpg	Vijay Deverakonda
8	Tom Cruise_21.jpg	Tom Cruise
9	Alia Bhatt_41.jpg	Alia Bhatt
10	Elizabeth Olsen_36.jpg	Elizabeth Olsen
11	Charlize Theron_29.jpg	Charlize Theron
12	Natalie Portman_74.jpg	Natalie Portman
13	Priyanka Chopra_6.jpg	Priyanka Chopra
14	Priyanka Chopra_73.jpg	Anushka Sharma
15	Natalie Portman_99.jpg	Natalie Portman
16	Zac Efron_6.jpg	Zac Efron
17	Marmik_5.jpg	Marmik
18	Camila Cabello_9.jpg	Camila Cabello
19	Elizabeth Olsen_23.jpg	Elizabeth Olsen
20	Claire Holt_28.jpg	Claire Holt

submission-faces-recognition-re

X. Tổng kết

- Kết quả của mô hình VGG16:

	Loss	Accuracy
Train	14.033	96.2%
Validation	30.976	90.5%

- Kết quả của mô hình ResNet-50:

	Loss	Accuracy
Train	2.256	99.5%
Validation	14.232	95.8%

- Link đến các model sau khi huấn luyện:
https://drive.google.com/drive/folders/1QPSSdDflIrgYj9plrQo67wIN0vFnXBo0?usp=share_link

XI. Tham khảo

ResNet50: https://viblo.asia/p/gioi-thieu-mang-resnet-vyDZOa7R5wj

VGG16: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

Skip-connection: https://d2l.aivivn.com/chapter_convolutional-modern/resnet_vn.html