ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN MÔN HỌC

THIẾT KẾ MẠNG NƠ-RON TÍCH CHẬP ỨNG DỤNG CHO NHẬN DIỆN KHUÔN MẶT

MAI ĐỨC HUY
huy.ms240422e@sis.hust.edu.vn

Ngành Kỹ thuật Điều khiển và Tự động hóa

Giảng viên hướng dẫn: PGS. TS. Nguyễn Hoài Nam

Chữ ký của GVHD

Khoa: Tự động hóa
Trường: Điện - Điện tử

Hà Nội, 01/2026

BỘ GIÁO DỤC & ĐÀO TẠO
ĐH BÁCH KHOA HÀ NỘI

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ
ĐỒ ÁN MÔN HỌC

Họ và tên sinh viên: MAI ĐỨC HUY
Khóa: K69 Trường: Điện – Điện tử Ngành: KT ĐK & TĐH

1. Tên đề tài
THIẾT KẾ MẠNG NƠ-RON TÍCH CHẬP ỨNG DỤNG CHO NHẬN DIỆN KHUÔN MẶT

2. Nội dung đề tài
Thiết kế một mô hình học sâu để nhận diện khuôn mặt trong ảnh

3. Thời gian giao đề tài: 17/12/2025

4. Thời gian hoàn thành: 31/12/2025

Ngày 07 tháng 01 năm 2026
CÁN BỘ HƯỚNG DẪN

Nguyễn Hoài Nam

TÓM TẮT ĐỒ ÁN

Đồ án sử dụng transfer learning MobileNetV2 để nhận diện khuôn mặt trên dataset đã được cung cấp

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. GIỚI THIỆU CHUNG

1.1 Giới thiệu dataset

Dataset bao gồm 600 bức ảnh khuôn mặt mà các thành viên trong lớp đã thu thập và tải lên teams. Mỗi thành viên trong lớp upload ảnh màu của 29 nhân vật khác nhau. Và nhiệm vụ của sinh viên là thiết kế model có thể nhận dạng khuôn mặt dựa trên tập dữ liệu đó.

1.2 MobilNetV2

1.2.1 Tổng quan

MobileNetV2 là một kiến trúc mạng nơ-ron tích chập (CNN) tối ưu cho thiết bị di động và nhúng, cải thiện hiệu năng và độ chính xác so với MobileNetV1, đồng thời giảm đáng kể chi phí tính toán và bộ nhớ.

1.2.2 Đóng góp chính

Inverted Residual Block

- Thay vì nối tắt giữa các lớp rộng (như ResNet), MobileNetV2 nối tắt giữa các lớp hẹp (bottleneck layers).
- Dòng dữ liệu được mở rộng, xử lý bằng convolution tách kênh (depthwise separable conv), rồi nén lại.
- Giúp tiết kiệm bộ nhớ và tăng hiệu quả lan truyền gradient.

Linear Bottleneck

- Loại bỏ hàm kích hoạt phi tuyến (ReLU) ở các lớp hẹp để tránh mất thông tin.
- Giữ lại tính biểu diễn mạnh mẽ của mạng trong không gian thấp chiều.

Hiệu quả tính toán cao

- Sử dụng depthwise separable convolution (3×3) $\rightarrow$ giảm số phép tính 8–9 lần so với convolution thường.
- Dễ triển khai trên phần cứng phổ biến và tối ưu bộ nhớ cho thiết bị di động.

1.2.3 Kiến trúc tổng thể

- Một lớp convolution đầu vào (32 filters)
- 19 khối bottleneck residual
- Dùng ReLU6 (giới hạn đầu ra từ 0 đến 6) để ổn định với tính toán số thấp.
- Các tham số có thể điều chỉnh:
  - Width multiplier (α): điều chỉnh độ rộng mạng.
  - Input resolution: điều chỉnh kích thước đầu vào để cân bằng độ chính xác – tốc độ.

Cấu hình chuẩn (α=1, 224×224):

- 3.4M tham số
- 300M Multiply-Adds

CHƯƠNG 2. QUÁ TRÌNH THỰC HIỆN

2.1 Gọi các thư viện

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader, random_split
import torchvision.models as models
```

Hình 2.1. Gọi các thư viện cần thiết

Gọi framework PyTorch, từ PyTorch gọi module torch.nn chứa các lớp xây dựng mạng nơ-ron. Module optim chứa các kỹ thuật cập nhật trọng số như Adam, SGD,... Gọi thư viện Torchvision (là thư viện mở rộng cho Pytorch) chuyên xử lý thị giác máy tính. Từ thư viện torchvision gọi module transforms để biến đổi hình ảnh. Thư viện os là thư viện Python làm việc với hệ thống (các file). Thư viện PIL là thư viện xử lý ảnh. Từ thư viện torch.utils.data gọi ra ba lớp: Dataset là lớp cơ sở để tạo một "custom dataset", DataLoader dùng để tạo batch dữ liệu, shuffle và load song song, random split để chia dataset thành các phần nhỏ. Gọi torchvision.models để sẻ dụng các mô hình CNN có sẵn, phục vụ cho việc transfer learning. Ngoài ra còn có thư viện matplotlib để phục vụ việc vẽ đồ thị.

2.2 Khai báo các hằng số

```
DATA_DIR = "/kaggle/input/face-recognition-mini-project"
CATEGORIES = [  "Brad Pitt",
                "Charles Leclerc",
                "Conor McGregor",
                "David Beckham",
                "Erling Haaland",
                "Faker",
                "General Vo Nguyen Giap",
                "Huy",
                "J97",
                "Jeff Bezo",
                "Jeffray",
                "Joji",
                "Khabib",
                "Leonardo DiCaprio",
                "Levi",
                "Messi",
                "Mixigaming",
                "Park Hang-seo",
                "Robert Downey Junior",
                "Ronaldo",
                "SonTungMTP",
                "Taylor Swift",
                "Tobey Maguire",
                "Tom Hanks",
                "Tom_Cruise",
                "Will Smith",
                "Zhao Lusi",
                "antony",
                "thayongnoi"]
IMG_SIZE = 300
BATCH_SIZE = 32
SEED = 42
```

Hình 2.2. Khai báo các hằng số

Tạo địa chỉ file chứa dataset của Kaggle. Tạo danh sách tên các nhãn. Cố định kích cỡ ảnh, kích thước batch, giá trị của seed. Các thuật toán random trên thực tế thì không random, thường sẽ nảy sinh từ 1 seed, nếu để mặc định thì seed sẽ được lấy từ thời gian hệ thống còn nếu tạo 1 seed trước thì các số sẽ được lấy ngẫu nhiên theo 1 quy luật đã định trước. Việc tạo seed để đảm bảo mỗi lần load lại code thì dataset sẽ được tạo lại theo đúng những gì đã phân chia từ trước. Tránh việc mỗi lần ta lại phải train và valid trên 2 bộ dữ liệu khác nhau, gây khó khăn cho việc kiểm tra mô hình và chỉnh sửa để tối ưu mô hình.

```
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")
```

Hình 2.3. Sử dụng CUDA để xử lý tính toán

2.3 Chuẩn bị Dataset

```
class FaceRecognitionMiniProject(Dataset):
    def __init__(self, data_dir, categories, transform=None):
        self.image_paths = []
        self.labels = []
        self.transform = transform

        for label, category in enumerate(categories):
            class_path = os.path.join(data_dir, category)
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                self.image_paths.append(img_path)
                self.labels.append(label)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert("RGB")
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

full_dataset = FaceRecognitionMiniProject(
    data_dir=DATA_DIR,
    categories=CATEGORIES,
    transform=transform
)
```

Hình 2.4. Chuẩn bị Dataset

Định nghĩa lớp FaceRecognitionMiniProject kế thừa từ Dataset của PyTorch bằng class FaceRecognitionMiniProject(Dataset). Dataset là một lớp cơ sở trong torch.utils.data dùng để tạo bộ dữ liệu tùy chỉnh.

Hàm def *__init__*(self, *data_dir*, categories, transform=None) là hàm khởi tạo của lớp bao gồm các tham số *data_dir* chứa đường dẫn đến thư mục chứa dataset, categories chứa danh sách tên các nhẫn đã được định nghĩa ở trên, transform là phép biến đổi ảnh được định nghĩa. Trong hàm này sẽ chứa đoạn code để đánh số cho các nhãn, do máy tính tính không thể hiểu được tên các nhãn là gì nên ta sẽ phải đánh số cho các nhãn và cho link đến thư mục chứa ảnh của nhãn đó. Vòng lặp bên trong sẽ duyệt qua các ảnh trong thư mục con đó và lưu link đến ảnh đó và lưu lại label cho chính ảnh đó. Vòng lặp lớn sẽ tiếp tục cho đến khi hết dữ liệu trong dataset.

Hàm *__len__*(self) sẽ trả về số lượng ảnh trong dataset.

Hàm *__getitem__*(self, idx) là cách để lấy mẫu dữ liệu theo chỉ sô idx. Trong hàm này đã thiết lập hàm mở ảnh từ đường dẫn và chuẩn hóa thành ảnh màu RGB. Nếu sử dụng transform thì sẽ áp dụng transform trước khi trả về ảnh và nhãn.

```python
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),  # đưa ảnh về [0, 1]
])
```

Hình 2.5. Chuẩn hóa ảnh

Ta sẽ chuẩn hóa ảnh về size 300x300, và chuyển ảnh thành 1 Tensor.

```python
total_size = len(full_dataset)
train_size = int(0.7 * total_size)
val_size   = int(0.3 * total_size)
test_size  = total_size - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(
    full_dataset,
    [train_size, val_size, test_size],
    generator=torch.Generator().manual_seed(SEED)
)

print(f"Train: {len(train_dataset)}")
print(f"Validation: {len(val_dataset)}")
print(f"Test: {len(test_dataset)}")

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

Hình 2.6. Load Dataset

Ta sẽ chia dataset đã load ở phía trên thành 3 tập train, valid, test theo tỷ lệ xác định. Trong đồ án này sử dụng tỷ lệ 7:3:0. DataLoader là lớp dùng để đưa dữ liệu ra theo từng batch nhỏ khi huấn luyện. Các tham số dataset là tập dữ liệu, batch_size là kích thước của batch đã chọn ở trên, shuffle là dữ liệu có được xáo trộn ngẫu nhiên mỗi epoch.

```python
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.RandomResizedCrop(227, scale=(0.9, 1.0)),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Hình 2.7. Data Augmentation

Các lệnh sẽ biến đổi hình ảnh như xoay, lật, cắt ảnh ngẫu nhiên, thay đổi màu ảnh để tránh mạng học vẹt.

2.4 Thiết kế model

```python
def get_model():
    weights = models.MobileNet_V2_Weights.IMAGENET1K_V2
    model = models.mobilenet_v2(weights=weights)
    for param in model.parameters():
        param.requires_grad = False
    model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(1280, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, num_classes),
    nn.Sigmoid())
    loss_fn = loss_fn = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr= 1e-3)
    return model.to(device), loss_fn, optimizer
```

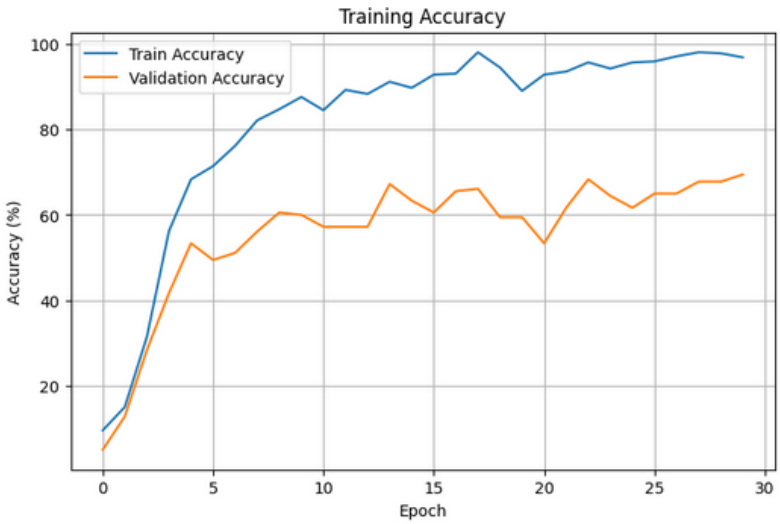
Hình 2.8. Thiết kế model

```
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                                  [-1, 1280, 7, 7]          --
|    └─Conv2dNormActivation: 2-1                   [-1, 32, 112, 112]        --
|    |    └─Conv2d: 3-1                           [-1, 32, 112, 112]        (864)
|    |    └─BatchNorm2d: 3-2                      [-1, 32, 112, 112]        (64)
|    |    └─ReLU6: 3-3                            [-1, 32, 112, 112]        --
|    └─InvertedResidual: 2-2                       [-1, 16, 112, 112]        --
|    |    └─Sequential: 3-4                       [-1, 16, 112, 112]        (896)
|    └─InvertedResidual: 2-3                       [-1, 24, 56, 56]          --
|    |    └─Sequential: 3-5                       [-1, 24, 56, 56]          (5,136)
|    └─InvertedResidual: 2-4                       [-1, 24, 56, 56]          --
|    |    └─Sequential: 3-6                       [-1, 24, 56, 56]          (8,832)
|    └─InvertedResidual: 2-5                       [-1, 32, 28, 28]          --
|    |    └─Sequential: 3-7                       [-1, 32, 28, 28]          (10,000)
|    └─InvertedResidual: 2-6                       [-1, 32, 28, 28]          --
|    |    └─Sequential: 3-8                       [-1, 32, 28, 28]          (14,848)
|    └─InvertedResidual: 2-7                       [-1, 32, 28, 28]          --
|    |    └─Sequential: 3-9                       [-1, 32, 28, 28]          (14,848)
|    └─InvertedResidual: 2-8                       [-1, 64, 14, 14]          --
|    |    └─Sequential: 3-10                      [-1, 64, 14, 14]          (21,056)
|    └─InvertedResidual: 2-9                       [-1, 64, 14, 14]          --
|    |    └─Sequential: 3-11                      [-1, 64, 14, 14]          (54,272)
|    └─InvertedResidual: 2-10                      [-1, 64, 14, 14]          --
|    |    └─Sequential: 3-12                      [-1, 64, 14, 14]          (54,272)
|    └─InvertedResidual: 2-11                      [-1, 64, 14, 14]          --
|    |    └─Sequential: 3-13                      [-1, 64, 14, 14]          (54,272)
|    └─InvertedResidual: 2-12                      [-1, 96, 14, 14]          --
|    |    └─Sequential: 3-14                      [-1, 96, 14, 14]          (66,624)
|    └─InvertedResidual: 2-13                      [-1, 96, 14, 14]          --
|    |    └─Sequential: 3-15                      [-1, 96, 14, 14]          (118,272)
|    └─InvertedResidual: 2-14                      [-1, 96, 14, 14]          --
|    |    └─Sequential: 3-16                      [-1, 96, 14, 14]          (118,272)
|    └─InvertedResidual: 2-15                      [-1, 160, 7, 7]           --
|    |    └─Sequential: 3-17                      [-1, 160, 7, 7]           (155,264)
|    └─InvertedResidual: 2-16                      [-1, 160, 7, 7]           --
|    |    └─Sequential: 3-18                      [-1, 160, 7, 7]           (320,000)
|    └─InvertedResidual: 2-17                      [-1, 160, 7, 7]           --
|    |    └─Sequential: 3-19                      [-1, 160, 7, 7]           (320,000)
|    └─InvertedResidual: 2-18                      [-1, 320, 7, 7]           --
|    |    └─Sequential: 3-20                      [-1, 320, 7, 7]           (473,920)
|    └─Conv2dNormActivation: 2-19                  [-1, 1280, 7, 7]          --
|    |    └─Conv2d: 3-21                          [-1, 1280, 7, 7]          (409,600)
|    |    └─BatchNorm2d: 3-22                     [-1, 1280, 7, 7]          (2,560)
|    |    └─ReLU6: 3-23                           [-1, 1280, 7, 7]          --
├─Sequential: 1-2                                  [-1, 29]                  --
|    └─Flatten: 2-20                               [-1, 1280]                --
|    └─Linear: 2-21                                [-1, 256]                 327,936
|    └─ReLU: 2-22                                  [-1, 256]                 --
|    └─Dropout: 2-23                               [-1, 256]                 --
|    └─Linear: 2-24                                [-1, 29]                  7,453
|    └─Sigmoid: 2-25                               [-1, 29]                  --
==========================================================================================
```

Hình 2.9. Cấu trúc của mạng

Ta sẽ sử dụng phần trích trọn đặc trưng của mạng đã được train trên tập ImageNet, đóng băng các chỉ số bằng lệnh param.requires_grad = False. Sau đó, ta sẽ chỉnh sửa phần phân loại của lớp phân loại để phù hợp với số đầu ra phù hợp với ứng dụng. Hàm mục tiêu được lựa chọn là hàm CrossEntropy và phương pháp tối ưu trọng số được chọn là Adam với tốc độ học là 1e-3.

2.5 Kết quả đạt được



Hình 2.10. Kết quả đạt được

Sau 30 epoch, ta đạt được kết quả như hình 2.10. Ta có thể thấy mô hình đã bị hiện tượng overfitting (quá khớp). Có thể do tập dữ liệu học quá nhỏ.

KẾT LUẬN

Qua đồ án môn học này em đã học được cách thiết kế một mạng nơ-ron tích chập học sâu. Tuy kết quả chưa được như mong muốn nhưng em cũng đã rút ra được 1 kinh nghiệm đắt giá: Phải đọc literature review trước khi thiết kế mạng để tìm hiểu trên thế giới họ đã sử dụng phương pháp nào để tối ưu mô hình, từ đó ta có thể học hỏi và sử dụng cho bài toán của ta. Những mạng học sâu này cần nhiều kinh nghiệm cũng như sự tính toán để đạt hiệu suất cao nhất, không đơn giản chỉ là chỉnh định tham số tham số của mạng như số lớp hay là loại lớp được sử dung. Đồ án này mang lại cho em cái nhìn tổng quan và hiểu được cách thiết kế, những vấn đề khi thiết kế một mô hình ứng dụng vào thực tế.