ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN MÔN HỌC SÂU

Mạng nơ-ron nhận diện khuôn mặt dựa trên Squeeze Net

LÊ QUỐC KHÁNH
Khanh.lq241785e@sis.hust.edu.vn

Ngành Kỹ thuật Điều khiển & Tự động hóa
Chuyên ngành Điều khiển tự động

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

Chữ ký của GVHD

KHOA: Tự động hóa

HÀ NỘI, 1/2026

Tóm tắt nội dung đồ án

Đồ án tập trung nghiên cứu và giải quyết bài toán nhận diện khuôn mặt trên tập dữ liệu nhỏ (Small Data) gồm 30 lớp đối tượng, với yêu cầu mô hình phải nhỏ gọn để có thể triển khai trên các thiết bị giới hạn tài nguyên. Phương pháp chính được sử dụng là Học chuyển giao (Transfer Learning) dựa trên kiến trúc mạng SqueezeNet 1.1.

Đồ án đã áp dụng các kỹ thuật xử lý ảnh đầu vào như: Lật ảnh, xoay ảnh, chuẩn hóa và sử dụng hàm mất mát CrossEntropy. Kết quả thực nghiệm cho thấy mô hình đã giảm thiểu đáng kể sự chênh lệch giữa tập Train và Test, đạt độ chính xác kiểm thử ổn định, đồng thời duy trì tốc độ xử lý nhanh phù hợp cho các ứng dụng thời gian thực. Các kiến thức và kỹ năng đạt được bao gồm: kỹ năng xử lý dữ liệu ảnh, kỹ thuật tinh chỉnh (fine-tuning) mạng CNN sâu, và kỹ năng sử dụng thư viện PyTorch.

Sinh viên thực hiện
Ký và ghi rõ họ tên

MỤC LỤC

CHƯƠNG 1. TỔNG QUÁT

1.1 Đặt vấn đề

Trong bối cảnh công nghệ thị giác máy tính phát triển mạnh mẽ, bài toán nhận diện khuôn mặt đóng vai trò quan trọng trong an ninh và xác thực. Tuy nhiên, việc triển khai các mô hình Deep Learning sâu trên thiết bị nhúng gặp nhiều trở ngại về tài nguyên tính toán. Các mô hình truyền thống như VGG hay ResNet thường có kích thước lớn và tốn nhiều bộ nhớ. Do đó, việc tìm kiếm một kiến trúc mạng nhẹ (Lightweight CNN) nhưng vẫn đảm bảo độ chính xác cao trên tập dữ liệu hạn chế là một bài toán cấp thiết. Nhiều nghiên cứu, như của Mahmoud Ali và cộng sự (2021), đã chỉ ra rằng việc sử dụng các mạng nơ-ron nhẹ như SqueezeNet để trích xuất đặc trưng mang lại hiệu quả cao trong cân bằng giữa tốc độ và độ chính xác.

1.2 Bài toán

Xây dựng hệ thống phân loại 30 đối tượng khuôn mặt từ bộ dữ liệu quy mô nhỏ. Xử dụng mạng nơ-ron có dung lượng nhỏ, khối lượng tính toán và tham số cừa phải để có thể ứng dụng lên các hệ thống nhúng

1.3 Lựa chọn mô hình

Đồ án lựa chọn **SqueezeNet 1.1** với các ưu điểm vượt trội:

- Kích thước mô hình dưới 5MB, nhỏ hơn 50 lần so với AlexNet.
- Kiến trúc **Fire Module** độc đáo giúp giảm thiểu tham số nhưng vẫn giữ được độ sâu của mạng.
- Khả năng tổng quát hóa tốt: Không chỉ hiệu quả trên mặt người, kiến trúc này còn được chứng minh thành công trong các bài toán nhận diện khuôn mặt động vật (như nhận diện cừu) trong các nghiên cứu gần đây.

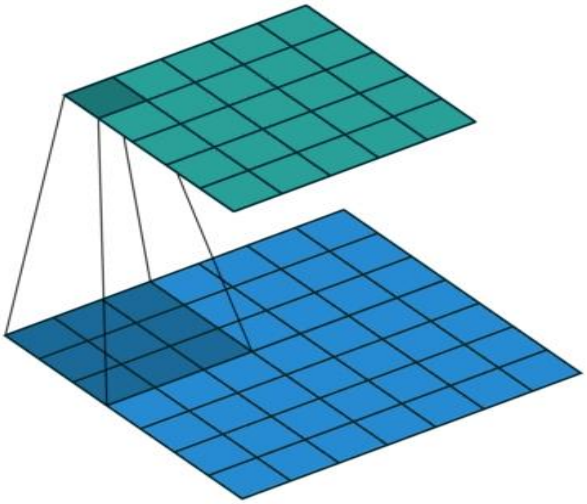
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Kiến trúc mạng SqueezeNet và Fire Module

2.1.1 Chiến lược thiết kế

a) Thu nhỏ mạng bằng cách thay thế các bộ lọc 3x3 bằng các bộ lọc 1x1.

- Conventional 3x3 thông thường được thay thế bằng convolution filters 1x1.
- Filter 1x1 có số tham số ít hơn 9 lần so với filter 3x3.



b) Giảm số lượng đầu vào cho các filter 3x3 còn lại.

- Số lượng đầu vào ít hơn cho các lớp conv dẫn đến số lượng tham số ít hơn.
- Đạt được điều này bằng cách chỉ sử dụng các filter 1x1 trước conv layer 3x3.
- Tổng số tham số trong conv layer 3x3 = (number of input channels) *(number of filters)* (3*3)

c) Lấy mẫu giảm ở giai đoạn cuối của mạng để các convolution layer có activation maps lớn.

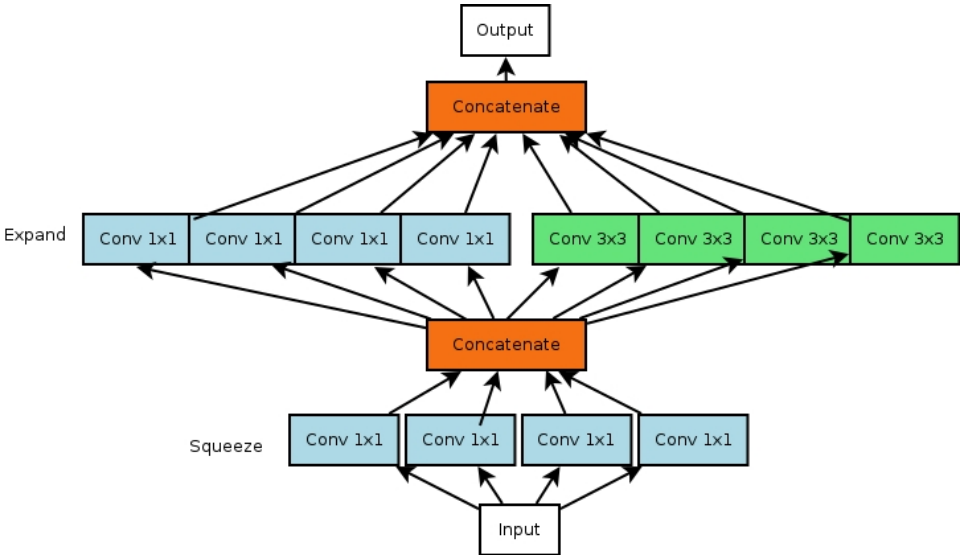
- Tận dụng tối đa số lượng tham số ít và tối đa hóa độ chính xác.
- Việc trì hoãn quá trình lấy mẫu giảm ở giai đoạn cuối của mạng nơ-ron tạo ra các bản đồ kích hoạt/đặc trưng lớn hơn.
- Khác biệt so với các kiến trúc truyền thống hơn như mạng VGG sử dụng phương pháp lấy mẫu giảm sớm.
- Việc sử dụng bản đồ kích hoạt lớn hơn sẽ mang lại độ chính xác phân loại cao hơn với cùng số lượng tham số.

2.1.2 Fire Module

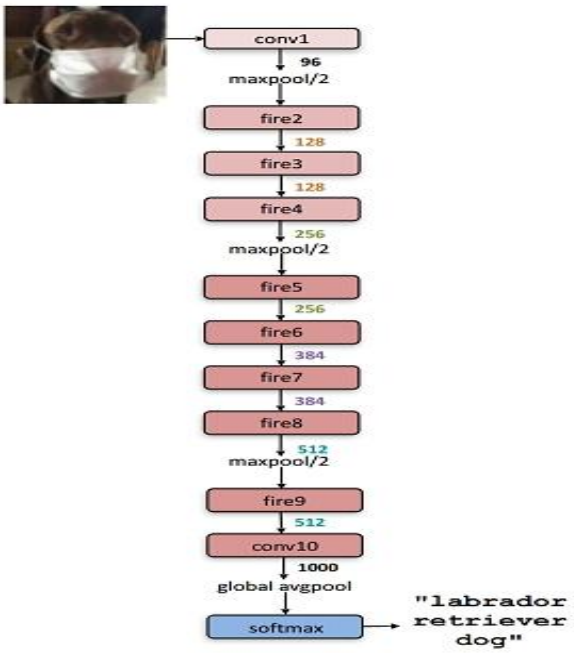
Fire Module:

- Là khối cấu tạo được sử dụng trong SqueezeNet
- Sử dụng các chiến lược thiết kế a, b và c
- Bao gồm các lớp nén (squeeze layers) chỉ có filter 1x1

- Bao gồm các lớp mở rộng (expand layers) kết hợp giữa convolution 1x1 và 3x3
- Số lượng filter trong squeeze layers phải ít hơn số lượng filter trong expand layers



2.1.3 Kiến trúc SqueezeNet



Layers breakdown:

- layer 1: regular convolution layer
- layer 2-9: fire module (squeeze + expand layer)
- layer 10: regular convolution layer
- layer 11: softmax layer

Thông số kỹ thuật:

- Tăng dần số lượng filter trên mỗi Fire Module
- Max pooling với stride = 2 sau layer 1, 4, 8
- Average pooling sau layer 10
- Downsampling chậm sau mỗi pooling layers

CHƯƠNG 3. XÂY DỰNG VÀ HUẤN LUYỆN MẠNG

3.1 Xây dựng mạng

Trong đồ án này ta sử dụng mạng SqueezeNet1.1 từ thư viện của Pytorch. Mặc dù mạng học sâu này có lượng thông số tương đối ít nhưng việc Train from scrach từ đầu với lượng data ít là khó khả thi nên ta sẽ Transfer Learning từ mạng đã train sẵn. Ta sẽ đóng băng phần trọng số phần đầu của mạng (phần lấy các đặc trưng), ta chỉ train và tinh chỉnh các trọng số trọng phần sau (phần phân loại) để thu được đầu ra là tỉ lệ của 30 nhãn từ data.

```
model=models.squeezenet1_1(weights=models.SqueezeNet1_1_Weights.DEF
AULT)

for param in model.features.parameters():
    param.requires_grad = False

num_classes = 30
in_channels = 512

model.classifier[1] = nn.Conv2d(in_channels, num_classes,
kernel_size=(1,1))

model = model.to(device)
```

3.2 Huấn luyện mạng

3.2.1 Chuẩn bị data

Như ta đã đề cập ở trên, do lượng data chúng ta có khá là nhỏ (30 nhãn, 20 ảnh mỗi nhãn) ~ 600 ảnh tổng.

Ta chia tệp dữ liệu thành 2 phần là training và test data với tỉ lệ là 7:3.

Với lượng data như này, ta sẽ sử dụng các phương pháp làm giàu như xoay, lật, tăng giảm tương phản, độ sáng, chuẩn hóa. Từ đó với mỗi lần học, mạng có thể có các ảnh khác nhau để lấy các đặc trưng, từ đó có thể chỉnh định được các trọng số.

Với tập test thì ta sẽ chỉ chuẩn hóa để có kết quả test khách quan nhất.

Bởi vì ảnh được chuẩn bị có thể có các kích thước khác nhau nên ta cũng cần phải resize lại ảnh về kích thước đầu vào của mạng là 224x224.

Ta cũng thêm các bước kiểm tra data xem data có đủ để chia không.

```
from torch.utils.data import DataLoader, Subset
from sklearn.model_selection import train_test_split
import numpy as np
from collections import Counter

data_dir = '/content/drive/MyDrive/DeepLearning/data'
```

```
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
])
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
])
temp_dataset = datasets.ImageFolder(data_dir)
targets = np.array(temp_dataset.targets)
class_counts = Counter(targets)
single_sample_classes = [cls for cls, count in class_counts.items()
if count < 2]
multi_sample_mask = ~np.isin(targets, single_sample_classes)
multi_sample_indices = np.where(multi_sample_mask)[0]
multi_sample_targets = targets[multi_sample_mask]
single_sample_indices = np.where(~multi_sample_mask)[0]

if len(multi_sample_indices) > 0 and
len(np.unique(multi_sample_targets)) > 1:
    train_multi_idx, test_idx = train_test_split(
        multi_sample_indices,
        test_size=0.3,
        random_state=42,
        stratify=multi_sample_targets
    )
else:
    print("Warning: Không đủ dữ liệu cho stratified split sau khi
lọc các lớp đơn lẻ. Thực hiện chia không stratified.")
    train_multi_idx, test_idx = train_test_split(
        np.arange(len(temp_dataset)),
        test_size=0.3,
        random_state=42
    )
if len(single_sample_indices) > 0:
    print(f"Cảnh báo: Có {len(single_sample_classes)} lớp chỉ có 1
mẫu, các mẫu này đã được thêm hoàn toàn vào tập huấn luyện.")
    train_idx = np.concatenate((train_multi_idx,
single_sample_indices))
else:
    train_idx = train_multi_idx
train_dataset = Subset(datasets.ImageFolder(data_dir,
transform=train_transforms), train_idx)
```

```
test_dataset = Subset(datasets.ImageFolder(data_dir,
transform=test_transforms), test_idx)

train_loader = DataLoader(train_dataset, batch_size=32,
shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=32,
shuffle=False, num_workers=2)
```

3.2.2 Huấn luyện và tinh chỉnh

Ta sẽ sử dụng hàm mất mát CrossEntropyLoss với phương pháp học Adam, learning rate là 0,001, số kỉ nguyên là 160 phù hợp để train với tập data ta đang có.

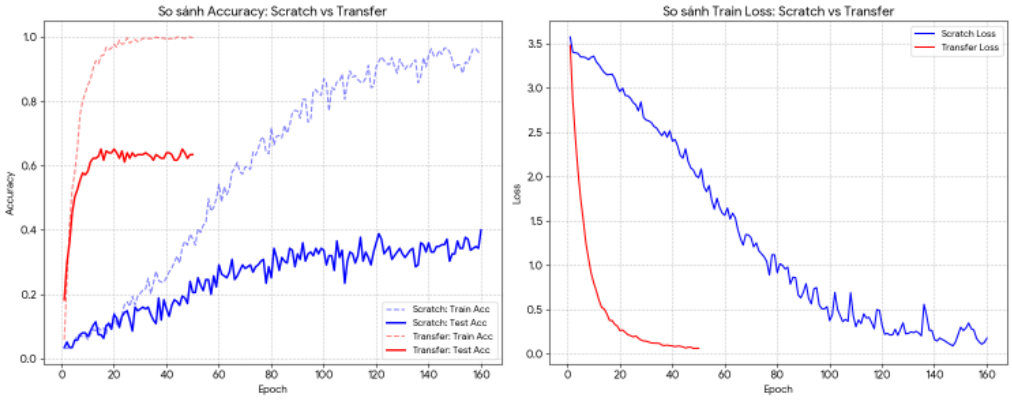
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 160

print("Bắt đầu huấn luyện...")

for epoch in range(num_epochs):

    model.train()
    running_loss = 0.0
    train_correct = 0
    train_total = 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

    train_acc = train_correct / train_total
    train_loss = running_loss / train_total

    model.eval()
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
```

```
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()
    test_acc = test_correct / test_total

    print(f'Epoch {epoch+1}/{num_epochs}:')
    print(f'   Train Loss: {train_loss:.4f} | Train Acc:
{train_acc:.4f}')
    print(f'   Test Acc:  {test_acc:.4f}  <-- Độ chính xác thực tế
trên tập kiểm tra')
print("Huấn luyện hoàn tất!")
torch.save(model.state_dict(),'/content/drive/MyDrive/DeepLearning/
face_recognition.pth')
```

Sau khi hoàn thành training, ta sẽ lưu các trọng số đã học được vào file pth để lần sau ta có thể dùng mạng đã train để nhận diện ảnh mà không cần train lại.

CHƯƠNG 4. KẾT QUẢ HUẤN LUYỆN VÀ NHẬN DIỆN

4.1 Kết quả huấn luyện



Đồ thị trên là kết quả huấn luyện bằng 2 phương pháp là học từ đầu và học kế thừa. Trong bài này, ta chỉ sử dụng kết quả của phương pháp học kế thừa, nhưng để so sánh thì ta thấy việc học từ đầu với việc cập nhật toàn bộ trọng số trong mạng rất khó. Đường đồ thị của phương pháp học từ đầu bị ramping chứng tỏ là việc học chưa hội tụ được đúng như ta mong muốn, so với đường Loss của phần học kế thừa rất mịn và hội tụ gần về 0.

Tuy nhiên độ chính xác trên tập test vẫn chỉ là khoảng 65%, việc đó cho thấy là tập dự liệu nhỏ này chưa đủ để giúp mạng học được hoàn toàn và chính xác

4.2 Ứng dụng vào nhận diện ảnh

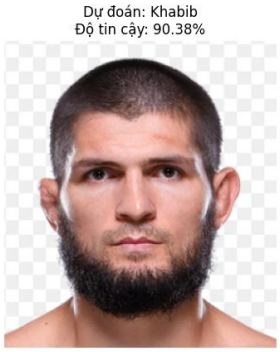
Từ file thông số ta đã lưu vào sau khi train thì ta có thể sử dụng để nhận diện 1 ảnh bất kì có nhãn trong 30 nhãn trong data train.

```
import torch
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image
import matplotlib.pyplot as plt
import os

# 1. Kết nối Google Drive
from google.colab import drive
if not os.path.exists('/content/drive'):
    drive.mount('/content/drive')

# ================ CẤU HÌNH ================
MODEL_PATH =
'/content/drive/MyDrive/DeepLearning/face_recognition_squeezenet_sp
lit.pth'
```

```
DATA_DIR = '/content/drive/MyDrive/DeepLearning/data' # Dùng để lấy
tên nhãn (Tên 30 người)
TEST_IMAGE_PATH = '/content/drive/MyDrive/DeepLearning/anh2.png' #
<--- Thay ảnh bạn muốn test vào đây
# ========================================

# 2. Thiết bị (GPU hoặc CPU)
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
print(f"Đang chạy trên: {device}")

# 3. Lấy lại danh sách nhãn (Class Names)
# Vì file .pth không lưu tên nhãn, ta cần đọc lại từ thư mục train
try:
    class_names = sorted([d for d in os.listdir(DATA_DIR) if
os.path.isdir(os.path.join(DATA_DIR, d))])
    print(f"Đã tải {len(class_names)} nhãn: {class_names}")
except FileNotFoundError:
    print("Lỗi: Không tìm thấy thư mục data để lấy tên nhãn. Bạn
cần tự định nghĩa list class_names.")
    # Ví dụ: class_names = ['Nguoi_A', 'Nguoi_B', ...]

# 4. Khởi tạo lại kiến trúc mạng (Giống hệt lúc train)
# Lưu ý: weights=None vì ta sẽ load weight của riêng ta
model = models.squeezenet1_1(weights=None)

# Sửa lại lớp cuối cùng cho khớp với 30 nhãn
model.classifier[1] = nn.Conv2d(512, len(class_names),
kernel_size=(1,1))

# 5. Load trọng số từ file .pth vào model
try:
    # map_location giúp load được kể cả khi bạn train bằng GPU
nhưng giờ chạy bằng CPU
    model.load_state_dict(torch.load(MODEL_PATH,
map_location=device))
    model = model.to(device)
    model.eval() # QUAN TRỌNG: Khóa các lớp Dropout/Batchnorm để dự
đoán chính xác
    print("-> Load model thành công!")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy file model tại {MODEL_PATH}")

# 6. Hàm xử lý và dự đoán
def predict_image(image_path):
    # Transform chuẩn cho SqueezeNet (giống lúc train)
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
```

```
    ])

    try:
        # Mở ảnh
        img_raw = Image.open(image_path).convert('RGB')

        # Tiền xử lý
        img_tensor = transform(img_raw).unsqueeze(0) # Thêm chiều
batch -> [1, 3, 224, 224]
        img_tensor = img_tensor.to(device)

        # Dự đoán
        with torch.no_grad():
            outputs = model(img_tensor)
            probs = torch.nn.functional.softmax(outputs, dim=1)
            confidence, idx = torch.max(probs, 1)

        label = class_names[idx.item()]
        score = confidence.item() * 100

        # Hiển thị
        plt.imshow(img_raw)
        plt.axis('off')

        plt.title(f"Dự đoán: {label}\nĐộ tin cậy: {score:.2f}%")
        plt.show()

        return label, score

    except Exception as e:
        print(f"Có lỗi khi đọc ảnh: {e}")

# --- CHẠY THỬ ---
predict_image(TEST_IMAGE_PATH)
```

Sau khi nhận diện thử 6 ảnh khác với các ảnh trong nhãn thì ta có được kết quả sau:



Với kết quả trên, ta thấy mạng chỉ nhận diện được 50% ảnh đúng với nhãn.

CHƯƠNG 5. KẾT LUẬN

5.1 Nhận xét

Từ các kết quả huấn luyện và nhận diện ảnh ở mục 4.1 và 4.2, ta có thể rút ra các nhận xét:

- Mạng Squeeze Net giúp mạng nơ-ron tiết kiệm được dung lương, giảm tham số tính toán nhiều hơn so với các mạng CNN hiện nay. (file lưu trữ chỉ gồm 2MB)
- Mạng có thể nhận diện được nhưng với kết quả tương đối
- Chất lượng nhận diện còn thấp là do lương data còn ít và chưa chuẩn (ảnh toàn thân, nửa thân)

5.2 Định hướng tương lai

Có thể áp dụng face detection vào khâu chuẩn bị data để tăng chất lương data, giúp mạng thu được nhiều đặc trưng hơn,

Có thể sử dụng các hàm mất mát chuyên dụng cho nhận diện mặt như ArcFace, AdaFace

Có thể sử dụng các cấu trúc mạng khác đặc thù hơn cho nhận diện mặt như MobileNet hay EfficientNet (cũng có dung lượng nhỏ và khối lượng tính toán vừa phải)

TÀI LIỆU THAM KHẢO

[1] Nguyễn Hoài Nam, Cơ sở của hệ mờ và mạng nơ ron, 2010.