TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN NHẬP MÔN THỊ GIÁC MÁY TÍNH

PHÂN TÍCH CONVOLUTIONAL NEURAL NETWORKS VÀ NHẬN DIỆN BIỂN BÁO GIAO THÔNG

Người hướng dẫn: **TS. PHẠM VĂN HUY**

Người thực hiện: **TRẦN THỊ VẸN - 52100674**

Lớp: **21050301**

Khoá: **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN NHẬP MÔN THỊ GIÁC MÁY TÍNH

PHÂN TÍCH CONVOLUTIONAL NEURAL NETWORKS VÀ NHẬN DIỆN BIỂN BÁO GIAO THÔNG

Người hướng dẫn: **TS. PHẠM VĂN HUY**

Người thực hiện: **TRẦN THỊ VẸN - 52100674**

Lớp: **21050301**

Khoá: **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Trong suốt quá trình học tập và rèn luyện, chúng em đã nhận được rất nhiều sự giúp đỡ tận tình, sự quan tâm, chăm sóc của thầy Phạm Văn Huy. Ngoài ra, chúng em còn được thầy truyền đạt những kiến thức về xử lý ảnh hay ho và thú vị, thầy cô còn giúp sinh viên có được nhiều niềm vui trong việc học và cảm thấy thoải mái, … Chúng em xin chân thành cảm ơn các thầy cô rất nhiều trong suốt quá trình học tập này!

Bởi lượng kiến thức của chúng em còn hạn hẹp và gặp nhiều vấn đề trong quá trình học nên báo cáo này sẽ còn nhiều thiếu sót và cần được học hỏi thêm. Chúng em rất mong em sẽ nhận được sự góp ý của quý thầy cô về bài báo cáo này để chúng em rút kinh nghiệm trong những môn học sắp tới. Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô.

TP Hồ Chí Minh, ngày 05 tháng 01 năm 2024

Sinh viên:

Trần Thị Vẹn – 52100674

BÁO CÁO CUỐI KÌ ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm của riêng tôi/chúng tôi và được sự hướng dẫn của thầy Phạm Văn Huy. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 05 tháng 01 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Trần Thị Vẹn

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Trong quá trình thực hiện phân tích một bài báo và ứng dụng mô hình vào thực tế, có rất nhiều bài học được rút ra:

- Sự quan trọng của dữ liệu: Tập dữ liệu đa dạng và đầy đủ là yếu tố quyết định cho hiệu suất của mô hình. Việc thu thập và chuẩn bị dữ liệu một cách cẩn thận là điều cần thiết.
- Kiến thức về CNNs: Hiểu rõ về cấu trúc và hoạt động của Convolutional Neural Networks là quan trọng để thiết kế và huấn luyện mô hình hiệu quả.
- Quy trình làm việc có cấu trúc: Quy trình từ chuẩn bị dữ liệu, xây dựng mạng CNN, huấn luyện, và kiểm tra mô hình cần phải được thực hiện một cách có cấu trúc và có kế hoạch.
- Thách thức trong việc xử lý ảnh thực tế: Các biển báo giao thông có thể xuất hiện trong các điều kiện ánh sáng và môi trường khác nhau, làm tăng độ phức tạp của bài toán nhận diện.
- Ứng dụng thực tiễn của CNNs: Hiểu rõ cách áp dụng CNNs vào các bài toán thực tế như nhận diện biển báo giao thông giúp nắm vững ứng dụng của công nghệ trong thế giới thực.

MỤC LỤC

DANH MỤC HÌNH ẢNH, BẢNG BIỂU VÀ ĐỒ THỊ

CHƯƠNG 1 - PHÂN TÍCH DỰA TRÊN BÀI BÁO

Thông tin của bài báo cần phân tích lý thuyết lẫn thực nghiệm:

- An Introduction to Convolutional Neural Networks

Đường dẫn: https://arxiv.org/pdf/1511.08458.pdf

Tác giả: Keiron O'Shea, Ryan Nash

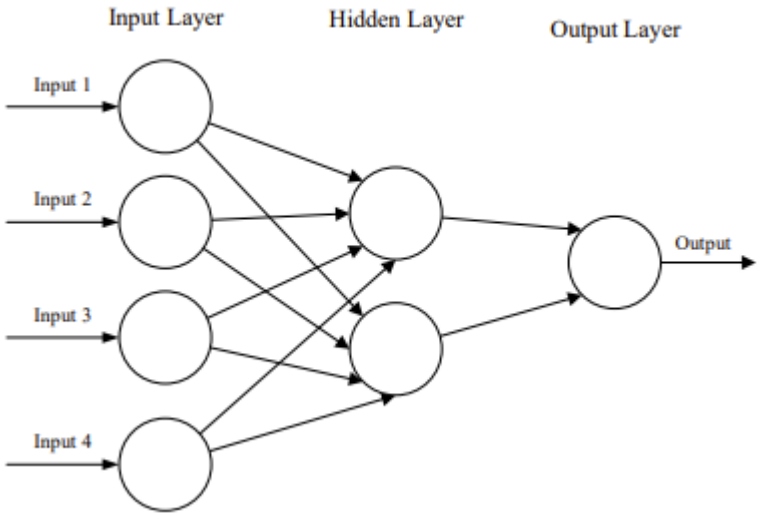
Thời gian: 2 Dec 2015 (version 2)

Cùng với sự bùng nổ của Mạng Nơ-ron Nhân Tạo (ANN), lĩnh vực học máy đã trải qua một sự biến động đáng kể trong thời gian gần đây. Một trong những dạng kiến trúc ANN ấn tượng nhất là Mạng Nơ-ron Convolutional (CNN). CNNs chủ yếu được sử dụng để giải quyết các nhiệm vụ nhận dạng mẫu đa dạng trên hình ảnh và với kiến trúc chính xác nhưng đơn giản của chúng, cung cấp một phương pháp đơn giản để bắt đầu với ANNs.

1.1 Một số thông tin giữa ANNs và CNNs

Các Mạng Nơ-ron Nhân Tạo (ANNs) là các hệ thống xử lý tính toán được lấy cảm hứng nặng nề từ cách hoạt động của các hệ thống thần kinh sinh học (như não người). ANNs chủ yếu bao gồm một số lượng lớn các nút tính toán kết nối với nhau một cách phức tạp (được gọi là nơ-ron), các nơ-ron này hoạt động theo một cách phân tán để cùng học từ đầu vào nhằm tối ưu hóa đầu ra cuối cùng.

Cấu trúc cơ bản của một ANN có thể được mô hình như được thể hiện trong Hình 1. Chúng ta sẽ tải đầu vào, thường là dưới dạng một vector đa chiều, vào lớp đầu vào sẽ phân phối nó tới các lớp ẩn. Các lớp ẩn sau đó sẽ ra quyết định dựa trên các lớp trước và đánh giá cách một sự thay đổi ngẫu nhiên bên trong nó làm hại hoặc cải thiện đầu ra cuối cùng, và điều này được gọi là quá trình học. Việc có nhiều lớp ẩn được xếp chồng lên nhau thường được gọi là học sâu.



Hình 1.1: A simple three layered feedforward neural network (FNN)

Mạng Nơ-ron Convolutional (CNNs) tương tự như ANN truyền thống ở chỗ chúng bao gồm các nơ-ron tự tối ưu hóa thông qua quá trình học. Mỗi nơ-ron vẫn nhận một đầu vào và thực hiện một phép toán (như một tích vô hướng tiếp theo bởi một hàm phi tuyến) - cơ sở của vô số ANN. Từ các vector hình ảnh đầu vào thô đến đầu ra cuối cùng của điểm số lớp, toàn bộ mạng vẫn sẽ biểu diễn một hàm điểm cảm thấu duy nhất (trọng số). Lớp cuối cùng sẽ chứa các hàm mất mát liên quan đến các lớp, và tất cả các mẹo và thủ thuật thông thường được phát triển cho ANN truyền thống vẫn áp dụng.

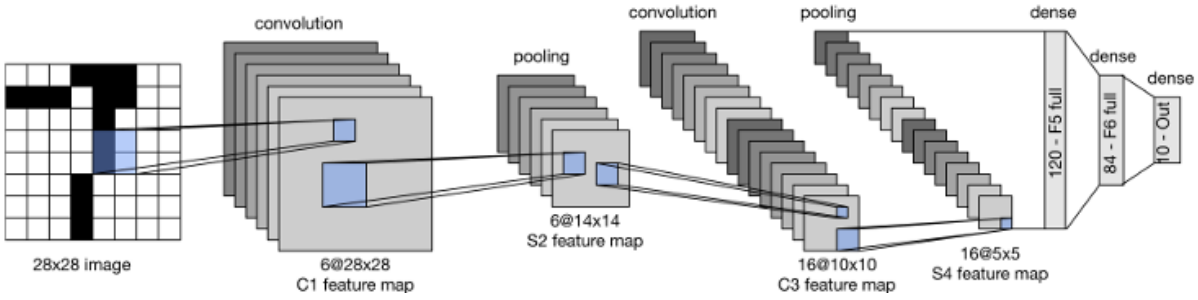
Sự khác biệt đáng chú ý duy nhất giữa CNN và ANN truyền thống là CNN chủ yếu được sử dụng trong lĩnh vực nhận dạng mẫu trong hình ảnh. Điều này cho phép chúng ta mã hóa các đặc điểm cụ thể của hình ảnh vào kiến trúc, làm cho mạng phù hợp hơn cho các nhiệm vụ tập trung vào hình ảnh - đồng thời giảm thiểu thêm các tham số cần thiết để thiết lập mô hình.

Một trong những hạn chế lớn nhất của các dạng truyền thống của ANN là chúng thường gặp khó khăn với độ phức tạp tính toán cần thiết để tính toán dữ liệu hình ảnh. Các tập dữ liệu chuẩn để đánh giá học máy như cơ sở dữ liệu MNIST về các chữ số viết tay phù hợp với hầu hết các dạng của ANN, nhờ vào số chiều hình ảnh tương đối nhỏ

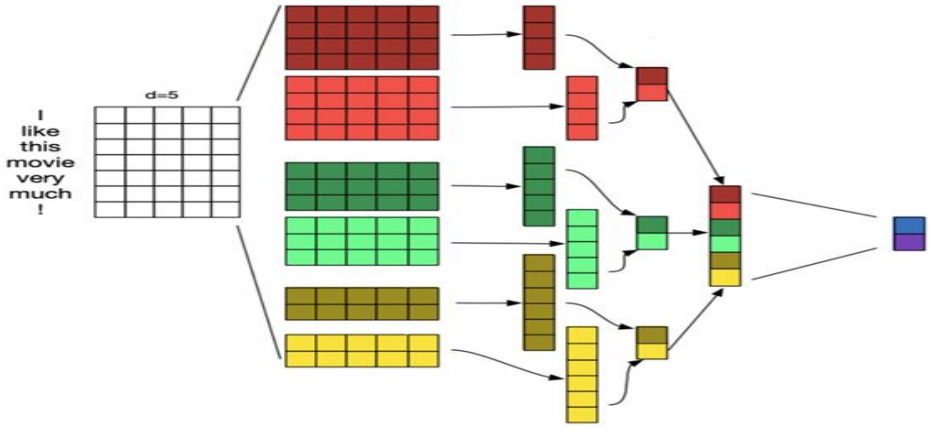
chỉ là 28 × 28. Với tập dữ liệu này, một nơ-ron đơn trong lớp ẩn đầu tiên sẽ chứa 784 trọng số (28×28×1, lưu ý rằng MNIST được chuẩn hóa thành giá trị đen và trắng), điều này có thể quản lý được đối với hầu hết các dạng của ANN.

Nếu bạn xem xét một đầu vào hình ảnh màu sắc lớn hơn với kích thước 64 × 64, số lượng trọng số trên một nơ-ron đơn của lớp đầu tiên tăng đáng kể lên 12,288. Hãy cũng lưu ý rằng để xử lý quy mô đầu vào này, mạng cũng sẽ cần phải lớn hơn nhiều so với một mạng được sử dụng để phân loại các chữ số MNIST đã được chuẩn hóa màu sắc. Khi đó, bạn sẽ hiểu được nhược điểm của việc sử dụng các mô hình như vậy.

1.2 Mô hình kiến trúc của CNN



Hình 1.1: Mô hình LeNet cho bài toán phân loại ảnh trên bộ dữ liệu MNIST



Hình 1.2: TextCNN cho bài toán phân loại văn bản trên bộ dữ liệu NTC-SCV

Trong xử lý ảnh, mỗi ảnh có hàng ngàn pixels, mỗi pixel được xem như 1 feature, vậy nếu như một bức ảnh có kích thước 1000 * 1000, thì sẽ có 1.000.000 features. Với

normal feed-forward neural net- works, mỗi layer là full-connected với previous input layer. Trong normal feed-forward neural network, với mỗi layer, có 1.000.000 pixels, mỗi pixel lại kết nối full-connected với 1.000.000 pixels ở layer trước, tức sẽ có 10^{12} tham số. Đây là còn số quá lớn để có thể tính được vào thời điểm đó, bởi vì với mô hình có nhiều tham số, sẽ dễ bị overfitted và cần lượng lớn data cho việc training, ngoài ra còn cần nhiều memory và năng lực tính toán cho việc training và prediction.

Vì vậy, sự ra đời CNN giúp xây dựng các model giải quyết hiệu quả với dữ liệu ảnh. Có 2 đặc tính của image hình thành nên cách hoạt động của CNN trên image, đó là feature localization và feature independence of location.

- Feature localization: mỗi pixel hoặc feature có liên quan với các pixel quanh nó.
- Feature Independence of location: mỗi feature dù nó có nằm ở đâu trong bức ảnh, thì nó vẫn mang giá trị của feature đó. CNN xử lý vấn đề có quá nhiều tham số với Shared parameters (feature inde- pendence of location) của Locally connected networks (feature loclization), được gọi là Convolution Net.

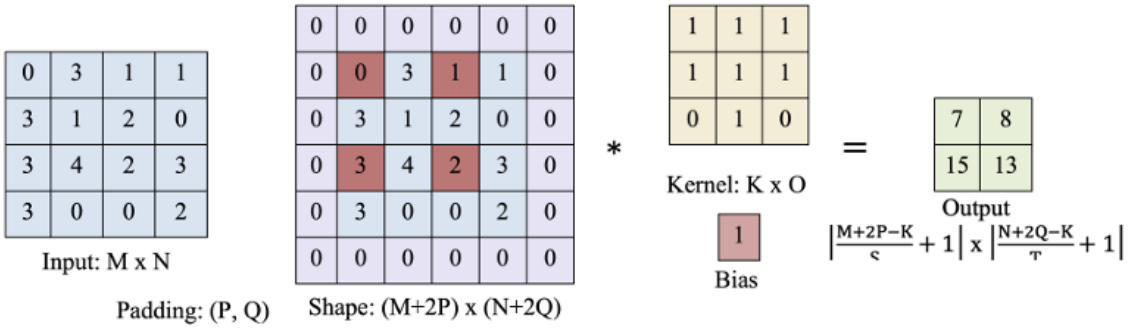
Locally connected layer: Trong hidden layer đầu tiên, mỗi node sẽ kết nối tới một cụm nhỏ pixels của input image chứ không phải toàn bộ image, gọi là small portion. Theo cách này, ta sẽ có ít kết nối hơn, vì thế ít tham số hơn giữa input và hidden layer đầu tiên.

Shared parameters: Có nhưng khu vực, mà việc tìm ra feature là giống nhau về cách làm, vì vậy ta có thể dùng chung bộ parameter, trong hình trên là phía phải bên trên và phía trái bên dưới. Tức ta chia sẻ bộ parameter giữa những vị trí khác nhau trong bức ảnh.

CNN có 3 thành phần chính:

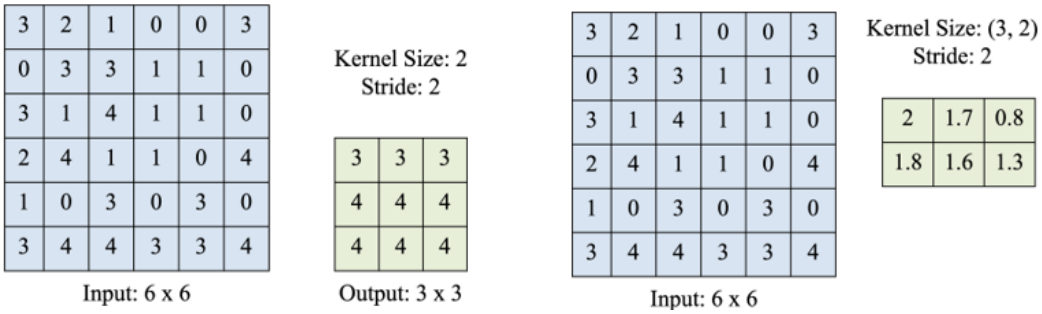
- Convulution layer: gồm Filter, Padding và Stride. Filter là locally connected network, mỗi filter (kernel) sẽ học được nhiều feature trong images. Ma trận ảnh đầu vào có thể sẽ được padding thêm các giá trị padding index vào ngoài các hàng và các cột. Mỗi filter sẽ di chuyển quanh bức ảnh với bước nhảy được cấu

hình trước là Stride.



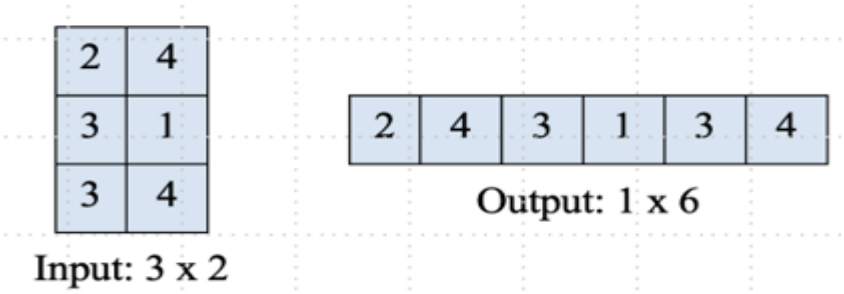
Hình 1.3: Convolutional Operation, Filter, Padding, Stride

- Pooling layer: bao gồm max-pooling và average-pooling layer tương ứng. Max-pooling layer chọn giá trị lớn nhất từ cửa sổ tính toán, còn Average-pooling layer sẽ tính giá trị trung bình của các giá trị mỗi kernel.



Hình 1.4: Max Pooling và Average Pooling

- Fully-connected layer: flatten convolution layer cuối cùng duỗi các ma trận nhiều chiều trong ảnh và fully connect neuron cho output layer.



Hình 1.5: Flatten Layer

CHƯƠNG 2 - PHÂN TÍCH BỘ DỮ LIỆU MNIST

Phần này xây dựng mô hình phân loại cho dữ liệu ảnh và văn bản sử dụng mạng CNN. Một trong những mô hình mạng CNN đầu tiên và nổi bật đó là LeNet được mô tả như hình 1.1.

Bộ dữ liệu MNIST

Đầu tiên, mô hình phân loại văn bản sử dụng LeNet được xây dựng trên tập dữ liệu phân loại chữ số MNIST, với 70.000 ảnh và nhãn tương ứng thuộc 10 classes từ 0 đến 9.

2.1 Tải về bộ dữ liệu

```
# Tải dữ liệu
ROOT = './data'
train_data = datasets.MNIST(root=ROOT, train=True, download=True)
test_data = datasets.MNIST(root=ROOT, train=False, download=True)
```

2.2 Tiền xử lý dữ liệu

- Chia tỉ lệ các tập training: validation = 0.9 : 0.1
- Chuẩn hoá dữ liệu và chuyển sang tensor sử dụng torchvison.transform

```
# Tiền xử lý dữ liệu
# Tỷ lệ dữ liệu sử dụng cho tập validation
VALID_RATIO = 0.9

# Tính số lượng mẫu dữ liệu được sử dụng cho tập huấn luyện
n_train_examples = int(len(train_data) * VALID_RATIO)
# Tính số lượng mẫu dữ liệu được sử dụng cho tập validation
n_valid_examples = len(train_data) - n_train_examples

# Chia tập dữ liệu huấn luyện thành tập huấn luyện và tập validation
train_data, valid_data = data.random_split(train_data,
[n_train_examples, n_valid_examples])

# Tính giá trị trung bình và độ lệch chuẩn từ tập dữ liệu huấn luyện
mean = train_data.dataset.data.float().mean() / 255
std = train_data.dataset.data.float().std() / 255

# Chuẩn hóa dữ liệu huấn luyện
train_transforms = transforms.Compose([
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])

# Chuẩn hóa dữ liệu kiểm tra
test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[mean], std=[std])
])

# Áp dụng các phép biến đổi cho tập dữ liệu
train_data.dataset.transform = train_transforms
valid_data.dataset.transform = test_transforms

# Kích thước batch
BATCH_SIZE = 256

# Tạo DataLoader cho tập dữ liệu huấn luyện và validation
train_dataloader = data.DataLoader(train_data, shuffle=True,
batch_size=BATCH_SIZE)
valid_dataloader = data.DataLoader(valid_data, batch_size=BATCH_SIZE)
```

2.3 Xây dựng mô hình LeNet

```
# Xây dựng mô hình LeNet
class LeNetClassifier(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        # Convolutional layer đầu tiên: 1 kênh đầu vào, 6 kênh đầu ra,
kernel size là 5x5, padding=2
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6,
kernel_size=5, padding=2)
        # Pooling layer đầu tiên: kernel size là 2x2
        self.avgpool1 = nn.AvgPool2d(kernel_size=2)
        # Convolutional layer thứ hai: 6 kênh đầu vào, 16 kênh đầu ra,
kernel size là 5x5
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16,
kernel_size=5)
        # Pooling layer thứ hai: kernel size là 2x2
        self.avgpool2 = nn.AvgPool2d(kernel_size=2)
        # Flatten layer để chuyển đổi feature maps thành vector
        self.flatten = nn.Flatten()
```

```
        # Fully connected layer 1: đầu vào có kích thước 16 * 5 * 5, đầu
ra có kích thước 120
        self.fc_1 = nn.Linear(16 * 5 * 5, 120)
        # Fully connected layer 2: đầu vào có kích thước 120, đầu ra có
kích thước 84
        self.fc_2 = nn.Linear(120, 84)
        # Fully connected layer 3: đầu vào có kích thước 84, đầu ra có
kích thước num_classes
        self.fc_3 = nn.Linear(84, num_classes)

    def forward(self, inputs):
        # Đầu vào đi qua convolutional layer đầu tiên, sau đó qua
pooling layer đầu tiên và hàm kích hoạt ReLU
        outputs = self.conv1(inputs)
        outputs = self.avgpool1(outputs)
        outputs = F.relu(outputs)
        # Tiếp tục với convolutional layer thứ hai, pooling layer thứ
hai và hàm kích hoạt ReLU
        outputs = self.conv2(outputs)
        outputs = self.avgpool2(outputs)
        outputs = F.relu(outputs)
        # Chuyển đổi feature maps thành vector
        outputs = self.flatten(outputs)
        # Đưa qua fully connected layer 1 và hàm kích hoạt ReLU
        outputs = F.relu(self.fc_1(outputs))
        # Tiếp tục với fully connected layer 2 và hàm kích hoạt ReLU
        outputs = F.relu(self.fc_2(outputs))
        # Cuối cùng, đưa qua fully connected layer 3 để có kết quả cuối
cùng
        outputs = self.fc_3(outputs)
        return outputs
```

2.4 Huấn luyện mô hình

```
# Huấn luyện mô hình
def train(model, optimizer, criterion, train_dataloader, device,
epoch=0, log_interval=50):
    # Chuyển mô hình sang chế độ huấn luyện
    model.train()
    total_acc, total_count = 0, 0
    losses = []
    start_time = time.time()
```

```
    # Duyệt qua từng batch trong tập dữ liệu huấn luyện
    for idx, (inputs, labels) in enumerate(train_dataloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()  # Đặt gradients về 0 trước khi lan truyền
ngược

        predictions = model(inputs)  # Dự đoán với mô hình

        loss = criterion(predictions, labels)  # Tính hàm mất mát
        losses.append(loss.item())  # Lưu giá trị hàm mất mát

        loss.backward()  # Lan truyền ngược
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)  # Cắt
tỉ lệ gradients

        optimizer.step()  # Cập nhật các tham số của mô hình

        # Tính toán accuracy
        total_acc += (predictions.argmax(1) == labels).sum().item()
        total_count += labels.size(0)

        # In thông tin sau mỗi log_interval batch
        if idx % log_interval == 0 and idx > 0:
            elapsed = time.time() - start_time
            print("| epoch {:3d} | {:5d}/{:5d} batches | accuracy
{:8.3f}".format(
                epoch, idx, len(train_dataloader), total_acc /
total_count
            ))

            total_acc, total_count = 0, 0
            start_time = time.time()

    # Tính toán accuracy và loss của epoch
    epoch_acc = total_acc / total_count
    epoch_loss = sum(losses) / len(losses)
    return epoch_acc, epoch_loss

# Đánh giá mô hình
```

```
def evaluate(model, criterion, valid_dataloader):
    # Chuyển mô hình sang chế độ đánh giá
    model.eval()
    total_acc, total_count = 0, 0
    losses = []

    # Không tính gradients trong quá trình đánh giá
    with torch.no_grad():
        # Duyệt qua từng batch trong tập dữ liệu validation
        for idx, (inputs, labels) in enumerate(valid_dataloader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            predictions = model(inputs)  # Dự đoán với mô hình

            loss = criterion(predictions, labels)  # Tính hàm mất mát
            losses.append(loss.item())  # Lưu giá trị hàm mất mát

            # Tính toán accuracy
            total_acc += (predictions.argmax(1) == labels).sum().item()
            total_count += labels.size(0)

    # Tính toán accuracy và loss của epoch
    epoch_acc = total_acc / total_count
    epoch_loss = sum(losses) / len(losses)
    return epoch_acc, epoch_loss
```

```
# Training
num_classes = len(train_data.dataset.classes)  # Số lượng lớp trong tập
dữ liệu
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')  #
Sử dụng GPU nếu có, nếu không thì sử dụng CPU

# Khởi tạo mô hình LeNet
lenet_model = LeNetClassifier(num_classes)
lenet_model.to(device)  # Di chuyển mô hình sang GPU (nếu có)

# Hàm mất mát và thuật toán tối ưu
criterion = nn.CrossEntropyLoss()  # Hàm mất mát: Cross Entropy Loss
optimizer = optim.Adam(lenet_model.parameters())  # Thuật toán tối ưu:
Adam
```

```
num_epochs = 10  # Số lượng epoch
save_model = './model'  # Đường dẫn để lưu trữ mô hình

train_accs, train_losses = [], []  # Danh sách để lưu trữ accuracy và
loss của quá trình huấn luyện
eval_accs, eval_losses = [], []  # Danh sách để lưu trữ accuracy và loss
của quá trình đánh giá
best_loss_eval = 100  # Đặt giá trị loss tốt nhất cho việc so sánh

# Vòng lặp qua các epoch
for epoch in range(1, num_epochs+1):
    epoch_start_time = time.time()

    # Huấn luyện
    train_acc, train_loss = train(lenet_model, optimizer, criterion,
train_dataloader, device, epoch)
    train_accs.append(train_acc)
    train_losses.append(train_loss)

    # Đánh giá
    eval_acc, eval_loss = evaluate(lenet_model, criterion,
valid_dataloader)
    eval_accs.append(eval_acc)
    eval_losses.append(eval_loss)

    # Lưu mô hình tốt nhất
    if eval_loss < best_loss_eval:
        torch.save(lenet_model.state_dict(), save_model +
'/lenet_model.pt')

    # In thông tin về accuracy và loss sau mỗi epoch
    print("-" * 59)
    print("| End of epoch {:3d} | Time: {:5.2f}s | Train Accuracy
{:8.3f} | Train Loss {:8.3f} | Valid Accuracy {:8.3f} | Valid Loss
{:8.3f} ".format(
        epoch, time.time()- epoch_start_time, train_acc, train_loss,
eval_acc, eval_loss
    ))
    print("-" * 59)

# Load mô hình tốt nhất
```

```
lenet_model.load_state_dict(torch.load(save_model + '/lenet_model.pt'))
lenet_model.eval()
```

```
| End of epoch   8 | Time: 27.98s | Train Accuracy    0.984 | Train Loss    0.039 | Valid Accuracy    0.985 | Valid Loss    0.047
-----------------------------------------------------------
| epoch   9 |    50/  211 batches | accuracy    0.990
| epoch   9 |   100/  211 batches | accuracy    0.990
| epoch   9 |   150/  211 batches | accuracy    0.990
| epoch   9 |   200/  211 batches | accuracy    0.989
-----------------------------------------------------------
| End of epoch   9 | Time: 27.59s | Train Accuracy    0.986 | Train Loss    0.034 | Valid Accuracy    0.984 | Valid Loss    0.054
-----------------------------------------------------------
| epoch  10 |    50/  211 batches | accuracy    0.992
| epoch  10 |   100/  211 batches | accuracy    0.989
| epoch  10 |   150/  211 batches | accuracy    0.989
| epoch  10 |   200/  211 batches | accuracy    0.990
-----------------------------------------------------------
| End of epoch  10 | Time: 27.64s | Train Accuracy    0.992 | Train Loss    0.031 | Valid Accuracy    0.985 | Valid Loss    0.045
-----------------------------------------------------------
```

```
LeNetClassifier(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (avgpool1): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (avgpool2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc_1): Linear(in_features=400, out_features=120, bias=True)
  (fc_2): Linear(in_features=120, out_features=84, bias=True)
  (fc_3): Linear(in_features=84, out_features=10, bias=True)
)
```

2.5 Đánh giá mô hình trên tập test

```
# Đánh giá mô hình trên tập test
test_data.transform = test_transforms
test_dataloader = data.DataLoader(test_data, batch_size=BATCH_SIZE)

test_acc, test_loss = evaluate(lenet_model, criterion, test_dataloader)
print("Accuracy trên tập test:", test_acc)
```

```
Accuracy trên tập test: 0.9853
```

CHƯƠNG 3 - CƠ SỞ LÝ THUYẾT VÀ DÙNG CNN NHẬN BIẾT BIỂN BÁO GIAO THÔNG

Hiện nay, sản phẩm về nhận diện biển báo giao thông trong thời gian thực đã đạt được sự phát triển đáng kể và được áp dụng trong nhiều ứng dụng thực tế. Nhiều sản phẩm đã đạt được độ chính xác cao trong việc nhận dạng các biển báo thông qua sự kết hợp giữa các công nghệ học máy và học sâu, đồng thời cải thiện đáng kể về tốc độ xử lý

trong thời gian thực, đáp ứng yêu cầu của các ứng dụng thực tế như xe tự hành, giám sát giao thông, v.v.

Tuy nhiên, mặc dù đã có sự phát triển tích cực, sản phẩm nhận diện biển báo giao thông trong thời gian thực vẫn đối mặt với một số thách thức. Điều kiện ánh sáng yếu, biển báo bị che khuất, đa dạng biển báo trong các quốc gia khác nhau, và các tình huống phức tạp như biển báo di động hay biển báo nằm trong khuôn hình động là những thách thức cần được giải quyết để cải thiện hiệu suất và độ tin cậy của hệ thống. Nên khi nghiên cứu đề tài nhận diện biển báo cấm trong môn xử lý ảnh số. Tôi đã nghiên cứu qua các kiến thức lý thuyết sau:

3.1 Phép toán Bitwise

Bitwise operations bao gồm các phép AND, OR, XOR, NOT. Bảng chân lý của các phép toán này được thể hiện ở hình dưới đây:

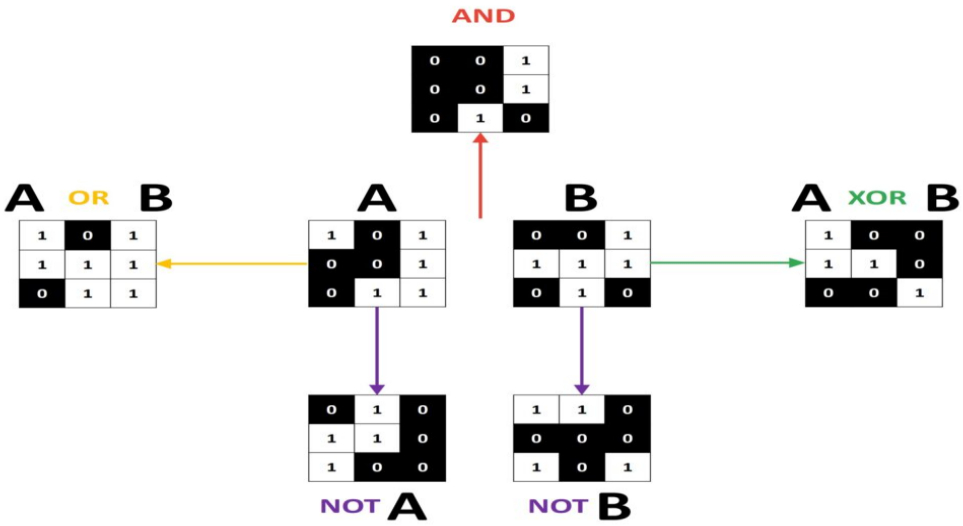
Khi thực hiện trên ảnh, X, Y sẽ đại diện cho giá trị pixel của ảnh khi đó:

- AND có giá trị 1 khi 2 pixel có giá trị lớn hơn 0.
- OR có giá trị 1 nếu một trong 2 pixel có giá trị lớn hơn 0.
- XOR có giá trị 1 nếu một trong 2 pixel có giá trị lớn hơn 0, nhưng đồng thời pixel còn lại phải có giá trị 0.
- NOT Đảo ngược giá trị pixel.

Với Bitwise Operations, phép toán này chỉ thực hiện trên ảnh nhị phân. Do đó để thực hiện ta cần đưa ảnh về nhị phân với quy luật đơn giản là pixel nào có giá trị lớn hơn 0 thì sẽ mang giá trị 1 và còn lại những pixel nào có giá trị 0 thì vẫn giữ nguyên giá trị là 0.

mask_red = cv2.bitwise_or(mask_red1, mask_red2)

color_result = cv2.bitwise_and(image, image, mask=mask_color)



Hình 3.1 Các thuật toán Bitwise

3.2 Chuyển đổi màu hệ màu

3.2.1 Chuyển đổi không gian màu RGB sang HSV

RGB là không gian màu rất phổ biến được dùng trong đồ họa máy tính và nhiều thiết bị kĩ thuật số khác. Ý tưởng chính của không gian màu này là sự kết hợp của 3 màu sắc cơ bản : màu đỏ (R, Red), xanh lục (G, Green) và xanh lơ (B, Blue) để mô tả tất cả các màu sắc khác.

HSV và cũng gần tương tự như HSL là không gian màu được dùng nhiều trong việc ch ỉnh sữa ảnh, phân tích ảnh và một phần của lĩnh vực thị giác máy tính. Hệ không gian này dựa vào 3 thông số sau để mô tả màu sắc H = Hue: màu sắc, S = Saturation: độ đậm đặc, sự bảo hòa, V = value: giá trị cường độ sáng.

hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

- Lấy R,G,B từ ảnh và tính R’ ; G’, B’ nằm trong khoảng [0;1]

R '= R / 255

G '= G / 255

B '= B / 255

- Tìm giá trị min và max trong 3 kênh màu:

Cmax = max (R ', G ', B ')

Cmin = min (R ', G ', B ')

Δ = Cmax – Cmin

- Tính toán Hue:

+ Nếu Cmax = R, Hue = 60 * ((G - B) / (Δ))

+ Nếu Cmax = G, Hue = 60 * ((B - R) / (Δ)) + 120

+ Nếu Cmax = B, Hue = 60 * ((R - G) / (Δ)) + 240

+ Nếu Cmax = 0, Hue = 0

- Tính toán Saturation:

+ Cmax = 0, Saturation = 0

+ Cmax khác 0, Saturation = (Δ) / Cmax

- Tính toán Value:

Value = Cmax

3.2.2 Chuyển đổi ảnh màu RGB thành ảnh xám (grayscale)

Để chuyển từ ảnh màu sang ảnh xám, ta thực hiện theo công thức sau

Độ sáng điểm ảnh = 0.2989*Red + 0.5870*Green + 0.1140*Blue

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

3.3 Simple Thresholding

Hàm sử dụng là threshold , tham số đầu tiên là 1 ảnh xám, tham số thứ 2 là giá trị ngưỡng, tham số thứ 3 maxval là giá trị được gán nếu giá pixel lớn hơn giá trị ngưỡng, tham số thứ 4 là loại phân ngưỡng. Tùy theo các loại phân ngưỡng mà pixel được gán giá trị khác nhau:

- THRESH_TOZERO

  Nếu giá trị pixel lớn hơn ngưỡng thì giữ nguyên giá trị

  Ngược lại gán bằng 0

- THRESH_TOZERO_INV

  Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng 0

Ngược lại giữ nguyên

- THRESH_BINARY

  Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng maxval

  Ngược lại bằng gán bằng 0

- THRESH_BINARY_INV

  Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng 0

  Ngược lại bằng gán bằng maxval

- THRESH_TRUNC

  Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng ngưỡng

  Ngược lại giữ nguyên giá trị

- ADAPTIVE THRESHOLDING

  Phương thức tính giá trị trung bình của các n điểm xung quanh pixel đó rồi trừ cho C chứ không lấy ngưỡng cố định (n thường là số lẻ, còn C là số nguyên bất kỳ)

- ADAPTIVE_THRESH_MEAN_C

  Giá trị ngưỡng là giá trị trung bình của vùng lân cận trừ đi hằng số C

- ADAPTIVE_THRESH_GAUSSIAN_C

Giá trị ngưỡng là tổng theo trọng số gausian của các giá trị lân cận trừ đi hằng số C

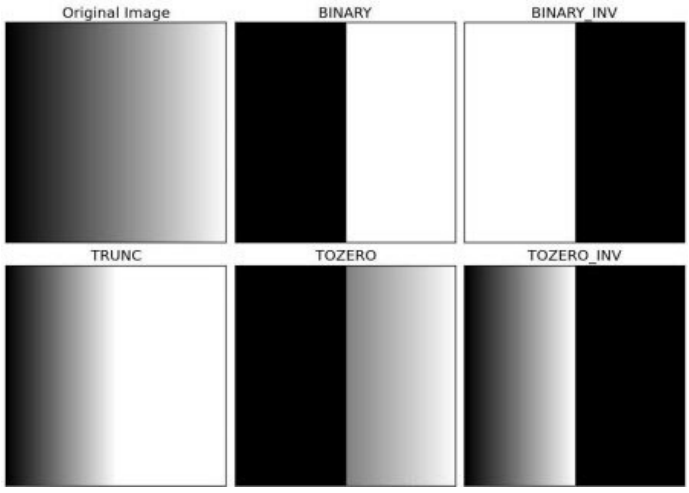
_, thresholded = cv2.threshold(image, lower_threshold, 255, cv2.THRESH_TOZERO)

_, thresholded_inv = cv2.threshold(thresholded, upper_threshold, 255, cv2.THRESH_TOZERO_INV)

_, adavetiveThresholded = cv2.AdaptiveThreshold(image, maxValue, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blockSize, C)

_, adavetiveThresholded = cv2.AdaptiveThreshold(image, maxValue,

cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, blockSize, C)



Hình 3.2: Minh họa trực quan với ngưỡng bằng 127

3.4 Gaussian Blur

Gaussian Blur là một kỹ thuật xử lý ảnh phổ biến trong lĩnh vực thị giác máy tính và xử lý ảnh số. Nó được sử dụng để làm mờ hoặc làm trơn ảnh bằng cách áp dụng một bộ lọc Gaussian lên mỗi điểm ảnh trong ảnh đầu vào.

Bộ lọc Gaussian là một bộ lọc phi tuyến tính có tính chất làm mờ ảnh dựa trên hàm Gaussian. Hàm Gaussian là một hàm có đặc trưng hình dạng hình chuông và phân phối xác suất Gauss. Bằng cách áp dụng bộ lọc Gaussian, các điểm ảnh gần nhau sẽ có tác động lên nhau và tạo ra hiệu ứng làm mờ ảnh.

Gaussian Blur được sử dụng trong nhiều ứng dụng xử lý ảnh như giảm nhiễu, làm trơn, tạo hiệu ứng mờ, tạo ảnh nền, trích xuất đặc trưng và nhiều ứng dụng khác. Nó giúp làm giảm nhiễu và loại bỏ các chi tiết không mong muốn trong ảnh, tạo ra hiệu ứng mờ mịn và tạo cảm giác mềm mại, tăng khả năng trích xuất các đặc trưng quan trọng trong ảnh.

GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])

blurred = cv2.GaussianBlur(binary_image, (5, 5), 0)

- src -Nó được sử dụng để nhập một Hình ảnh.
- dst -Nó là một biến lưu trữ Hình ảnh đầu ra.
- ksize -Nó định nghĩa Kích thước Kernel Gaussian [chiều rộng chiều cao]. Chiều cao và chiều rộng phải là số lẻ (1,3,5,…) và có thể có các giá trị khác nhau. Nếu ksize được đặt thành [0,0], thì ksize được tính từ giá trị sigma.
- sigmaX – Dẫn xuất tiêu chuẩn hạt nhân dọc theo trục X. (hướng nằm ngang).
- sigmaY – Dẫn xuất tiêu chuẩn hạt nhân dọc theo trục Y (hướng dọc). Nếu sigmaY = 0 thì giá trị sigmaX được lấy cho sigmaY.
- borderType – Đây là các ranh giới hình ảnh được chỉ định trong khi hạt nhân được áp dụng trên các đường viền hình ảnh. Loại đường viền có thể có là:

+ cv.BORDER_CONSTANT
+ cv.BORDER_REPLICATE
+ cv.BORDER_DEFAULT…

3.5 findContours()

Contour được hiểu đơn giản là một đường cong liên kết toàn bộ các điểm liên tục (dọc theo đường biên) mà có cùng màu sắc hoặc giá trị cường độ. Contour rất hữu ích trong phân tích hình dạng, phát hiện vật thể và nhận diện vật thể. Một số lưu ý khi sử dụng contour.

Để độ chính xác cao hơn thì nên sử dụng hình ảnh nhị phân (chỉ gồm 2 màu đen và trắng). Do đó trước khi phát hiện contour thì nên áp dụng threshold hoặc thuật toán canny để chuyển sang ảnh nhị phân.

Hàm **findContour** và **drawContour** sẽ thay đổi hình ảnh gốc. Do đó nếu bạn muốn hình ảnh gốc sau khi tìm được contour, hãy lưu nó vào một biến khác.

Trong openCV, tìm các contours như là tìm các vật thể màu trắng từ nền màu đen. Do đó hãy nhớ rằng, object cần tìm nên là màu trắng và background nên là màu đen.

contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

Có 3 tham số trong hàm cv2.findContours():

+ Ảnh gốc

+ Phương pháp trích xuất contours

+ Phương pháp xấp xỉ contour.

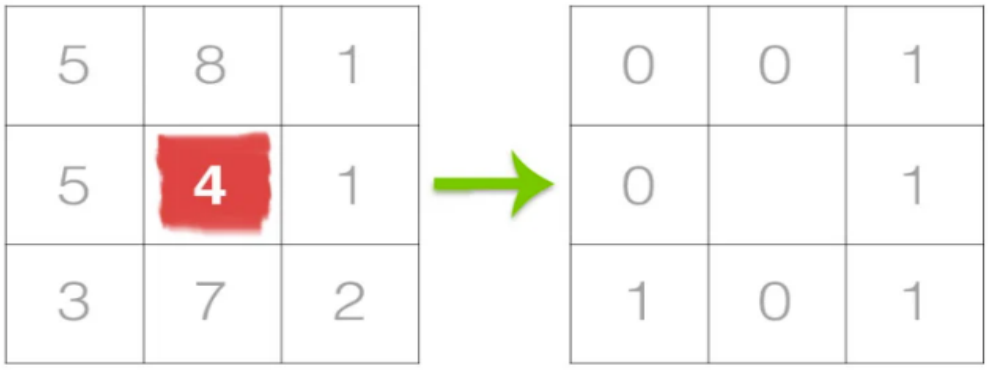
Kết quả trả ra là hình ảnh và contours. Trong đó contours là một list của toàn bộ các contours xác định trong hình ảnh. Mỗi một contour là một numpy array của các tọa độ của các điểm biên trong object.

3.6 Local Binary Pattern (LBP)

LBP là 1 toán tử cấu trúc đơn giản và hiệu quả gắn nhãn các pixel của ảnh bằng cách đánh giá các vùng lân cận của mỗi pixel và xem xét kết quả là một số nhị phân. LBP được dùng để đo độ tương phản cục bộ ảnh.

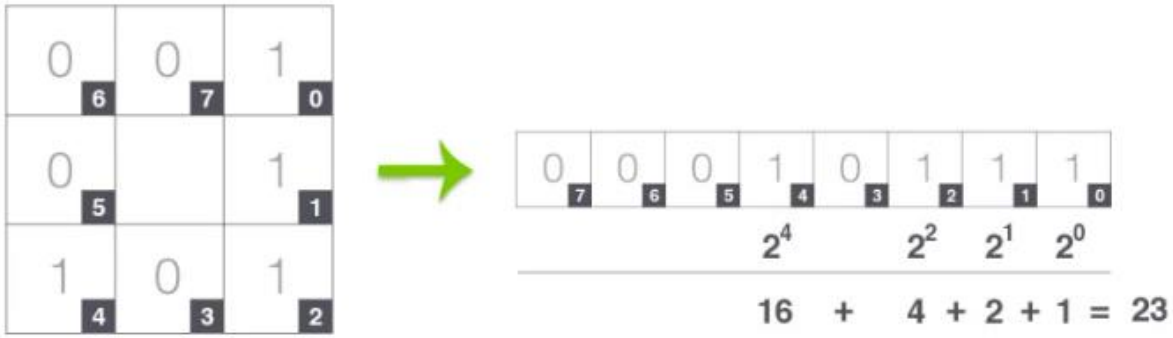
Các bước tiến hành trích xuất đặc trưng theo phương pháp LBP:

- Bước 1: duyệt lần lượt từng pixel trên ảnh (theo cột -> theo hàng), với pixel đang xét, ta áp dụng bước 2->4.
- Bước 2: xét lần lượt 8 pixel lân cận (hàng xóm - neighbor) của pixel đang duyệt (trung tâm - center). Mỗi pixel hàng xóm sẽ ứng với một bit trong một chuỗi 8-bit. Chuỗi 8-bit này ban đầu sẽ bằng: 00000000. Chuỗi 8-bit này sẽ được cập nhật theo mô tả ở bước 3.
- Bước 3: nếu mức sáng tại pixel hàng xóm >= mức sáng tại pixel trung tâm: vote bit ở vị trí tương ứng lên 1 trong chuỗi 8-bit đề cập ở bước 2.
- Bước 4: Sau khi hoàn tất bước 2 và 3, ta sẽ có mội chuỗi 8-bit (vd: 00101100) -> đổi giá trị nhị phân này sang thập phân để lưu trữ (vd: 00101100 nhị phân = 44 thập phân).
- Bước 5: lặp hết toàn ảnh (bước 1->4), ta sẽ có kết quả đầu ra bằng kích thước với ảnh đầu vào. Mỗi giá trị trên ảnh đầu ra là đặc trưng LBP.



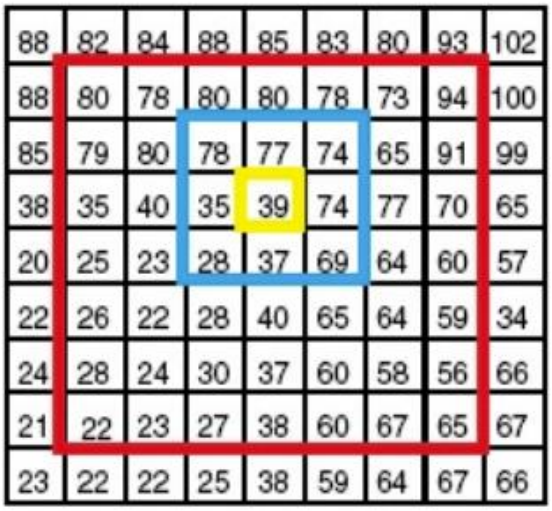
Hình 3.3: Ví dụ về LBP

Ví dụ: trong ảnh ta xét pixel 4, các vùng lân cận của nó là: 5 – 8 – 1 – 1 – 2 – 7 – 3 – 5. Ta sẽ đi tìm trọng số (gọi là X) của mỗi điểm ảnh để tính toán giá trị nhị phân cần thiết. Với các vùng lân cận (tạm gọi là A), ta so sánh mỗi pixel với pixel đang xét (tạm gọi là B) (ở đây là pixel mang label = 4), nếu: A < B => X = 1 ngược lại nếu A > B => X = 0. Ta có ma trận trọng số như sau:



Hình 3.4: Ma trận trọng số

Sau khi có ma trận trọng số, ta sẽ có dãy nhị phân: 00010111 => số nhị phân: 23.

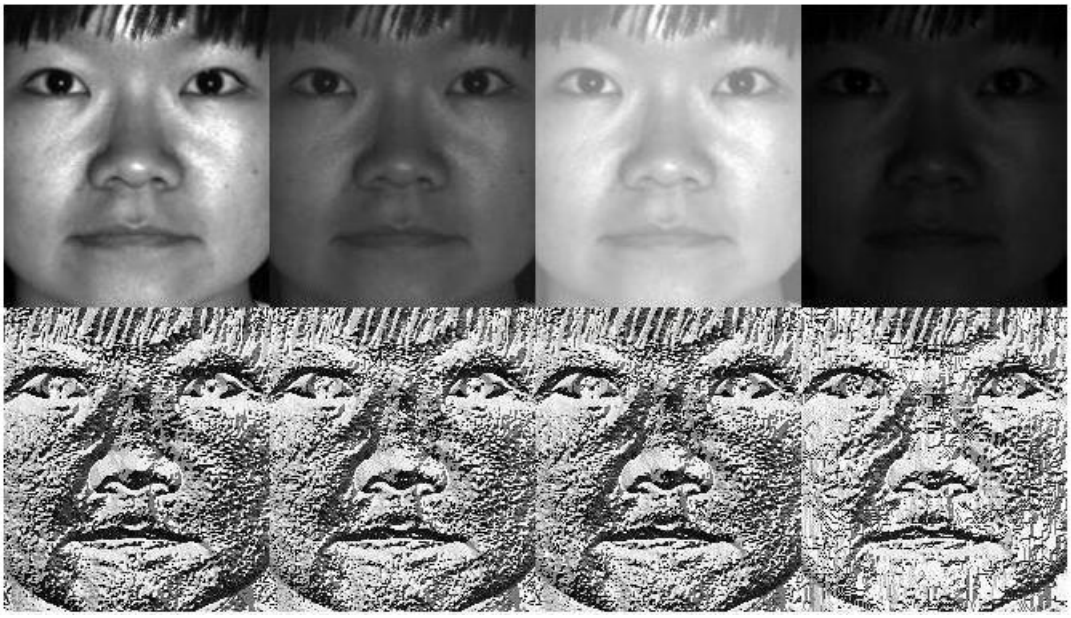


Hình 3.5: Ví dụ về trích xuất đặc trưng

Thường các điểm B được xác định nằm trong vùng màu đỏ, ta thường bỏ đi các pixel ở rìa ngoài cùng, bởi nếu xét các pixel này ta sẽ không có đủ vùng lân cận, trong 1 bức ảnh thì loại bỏ đi 1 pixel ở rìa là không đáng kể, và các pixel này cũng không gây ảnh hưởng tới bức ảnh do các đối tượng được mô tả trong ảnh thường không thuộc pixel này.

Ví dụ ta xét điểm B là 39 (ô màu vàng) thì vùng lân cận của nó là vùng màu xanh. Với mỗi điểm ta xét ta sẽ có 1 giá trị nhị phân, các giá trị này sẽ tạo thành 1 bức ảnh xám.

Đây là kết quả sau khi sử dụng LBP, với cùng 1 gương mặt sử dụng các độ sáng tối, khác nhau ta sẽ nhận được kết quả giống nhau.



Hình 3.6: Ảnh trước và sau trích xuất đặc trưng

LBP ta có thể tạm hiểu (để nhớ như sau):

- Local: thể hiện tính chất cục bộ địa phương, đó là khi ở bước 2 ta xét pixel lân cận -> mỗi đặc trưng trong output sẽ mang đặc trưng đại diện cục bộ.
- Binary Patterns: các mẫu hình nhị phân -> cách nhị phân hóa mô tả ở bước 3, 4

Tổng quát hóa phương pháp tiếp cận LBP trên, ta sẽ có các tham số sau:

- P: Số pixel lận cận pixel trung tâm (vd: P=8).
- R: Bán kính của pixel lân cận mà ta sẽ xét - cách pixel trung tâm bao nhiêu pixel (vd: R=1 nghĩa là liền kề).
- Thứ tự các pixel lân cận mã hóa vào chuỗi 8-bit sẽ theo chiều kim đồng hồ hay ngược chiều kim đồng hồ.
- Interpolation: do lấy pixel lân cận theo hình tròn, do đó tọa độ của các pixel lân cận khi tính toán ra sẽ là số thực -> quyết định lấy ra giá trị mức sáng của pixel đó theo cách nào: pixel gần nhất (nearest) hay có trọng số (bilinear).

3.7 Hough Circle Detection

Phương pháp Hough Circle Detection trong OpenCV là một kỹ thuật được sử dụng để phát hiện các vòng tròn trong ảnh. Đây là một ứng dụng cụ thể của phép biến đổi Hough, một kỹ thuật tổng quát hơn để phát hiện các hình dạng đơn giản như đường thẳng, vòng tròn, ellipses, v.v. trong không gian ảnh. Dưới đây là những điểm chính của phương pháp này:

Biến đổi Hough cho Vòng tròn: Mỗi điểm trên vòng tròn có thể được biểu diễn trong không gian Hough bằng một mặt cầu trong không gian ba chiều (với các trục là tâm vòng tròn (x,y) bán kính r). Khi các mặt cầu này giao nhau ở một điểm, điểm đó biểu diễn một vòng tròn trong không gian ảnh.

Dựa vào Gradient: OpenCV sử dụng phương pháp Hough Gradient, một biến thể của biến đổi Hough, dựa trên việc tính toán gradient (đạo hàm) của ảnh. Phương pháp này xác định biên (edges) của vòng tròn thông qua việc tìm cực trị trong không gian gradient.

Hàm cv2.HoughCircles trong OpenCV được sử dụng để thực hiện phát hiện vòng tròn. Hàm này yêu cầu một số tham số:

- Ảnh đầu vào: Thường là ảnh đã được xử lý qua bước phát hiện cạnh, chẳng hạn sử dụng Canny edge detector.
- Phương pháp: thông thường là cv2.HOUGH_GRADIENT.
- dp (Inverse Ratio of the Accumulator Resolution): Độ phân giải ngược của không gian tích lũy. Giá trị này quyết định độ chính xác của vòng tròn được phát hiện.
- minDist: Khoảng cách tối thiểu giữa các tâm của vòng tròn phát hiện được. param1 và param2: Các tham số cho phép điều chỉnh ngưỡng cho các giai đoạn phát hiện cạnh và tìm trung tâm.
- minRadius và maxRadius: Giới hạn bán kính nhỏ nhất và lớn nhất của các vòng tròn cần phát hiện.

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20, param1=50, param2 = 30, minRadius=0,maxRadius=0)

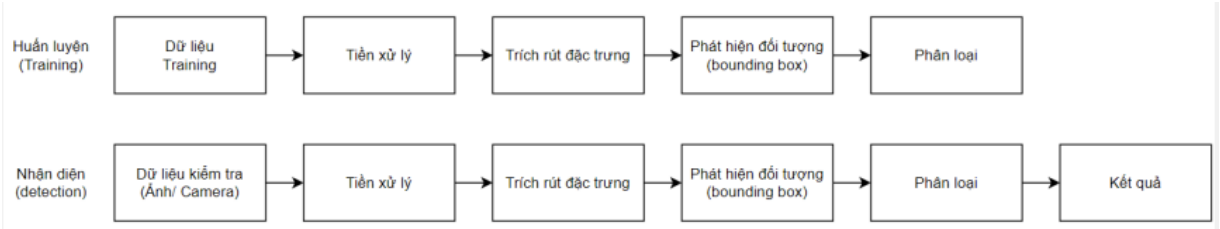
3.8 Vẽ hình chữ nhật (Drawing Rectangle)

Để vẽ một hình chữ nhật, bạn cần góc trên bên trái và góc dưới cùng bên phải của hình chữ nhật. Lần này ta sẽ vẽ một hình chữ nhật màu xanh lá cây ở góc trên cùng bên phải của hình ảnh.

img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

3.9 Phương pháp giải quyết đề tài

Một hệ thống nhận dạng đối tượng thông thường gồm các bước sau đây:



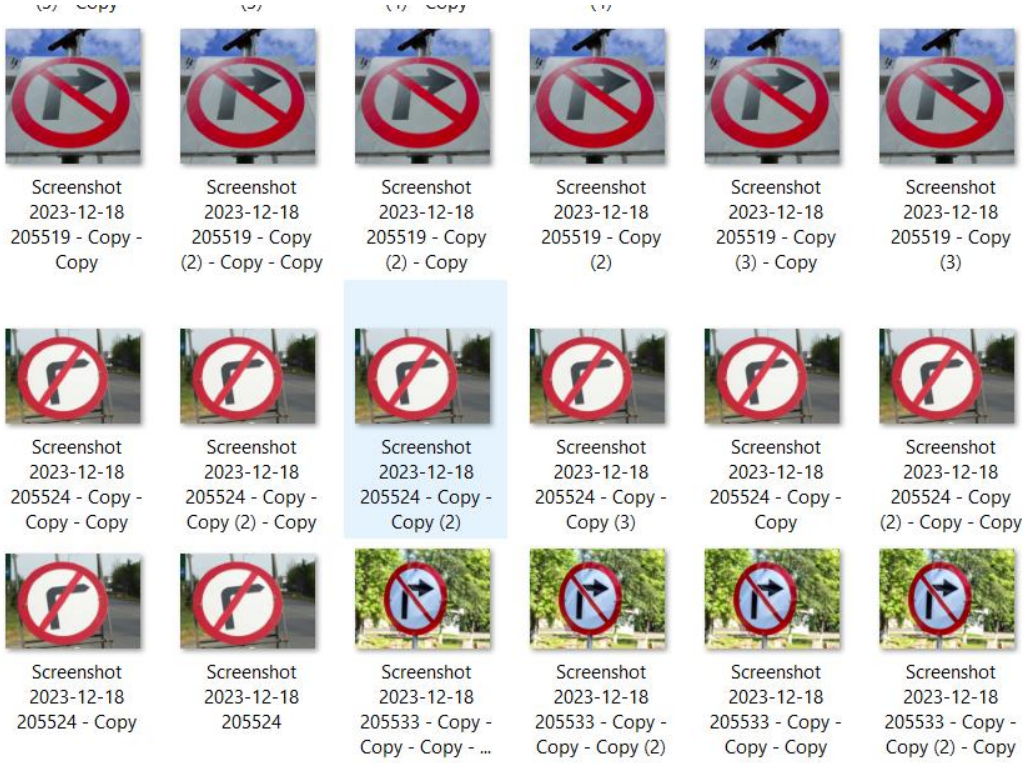
Hình 3.7: Sơ đồ hệ thống nhận diện đối tượng

- Tiền xử lý: Bước này nhằm mục đích lọc nhiễu, nâng cao chất lượng ảnh, trong bước này bao gồm các bước : căn chỉnh ảnh, chuẩn hóa ánh sáng
- Trích rút đặc trưng: LBP
- Phát hiện đối tượng và phân loại: ở bước này 1 phương pháp phát hiện và phân loại vật thể như CNN, SSD, YOLO, … được sử dụng.
- Kết quả là một ảnh với đối tượng được bounding box và kết quả được dự đoán.

3.9.1 Thu thập dữ liệu

Đầu tiên, mục tiêu của đề tài là nhận diện biển báo cấm trong hình. Vì thế sẽ có 15 ảnh để thử nghiệm (trong đó có 5 ảnh có 2-3 biển báo bên trong hình).

Ngoài tập ảnh thử nghiệm, thì em có thu thập trên kaggle tập dữ liệu để cho mô hình học để dự đoán được biển báo.

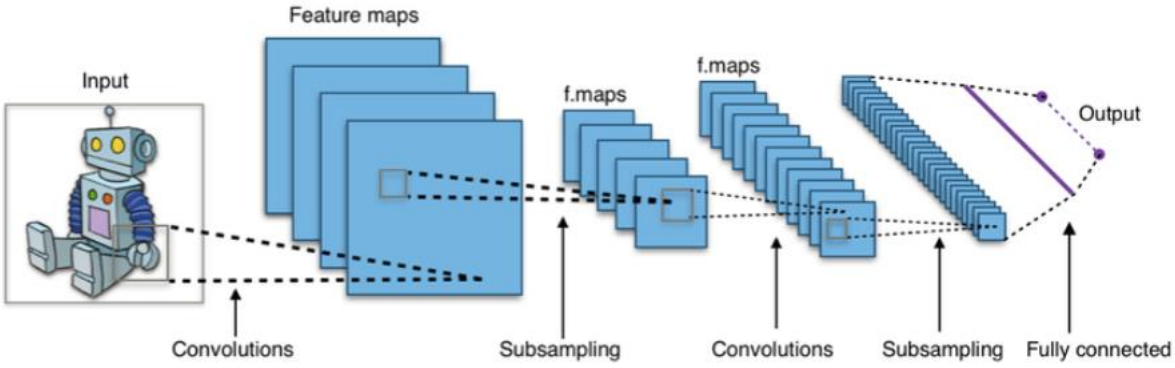


Hình 3.8: Biển báo được thu thập

Gồm 19 loại biển báo, được chụp với nhiều góc độ và kích thước khác nhau.

3.9.2 Mô hình học sâu CNN (modeltest.h5)

Trong quá trình huấn luyện mạng (traning) CNN tự động học các giá trị qua các lớp filter dựa vào cách thức mà bạn thực hiện. Ví dụ trong tác vụ phân lớp ảnh, CNNs sẽ cố gắng tìm ra thông số tối ưu cho các filter tương ứng theo thứ tự raw pixel > edges > shapes > facial > high-level features. Layer cuối cùng được dùng để phân lớp ảnh.



Hình 3.9: CNN models

Trong mô hình CNN có 2 khía cạnh cần quan tâm là tính bất biến (Location Invariance) và tính kết hợp (Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các gốc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể.

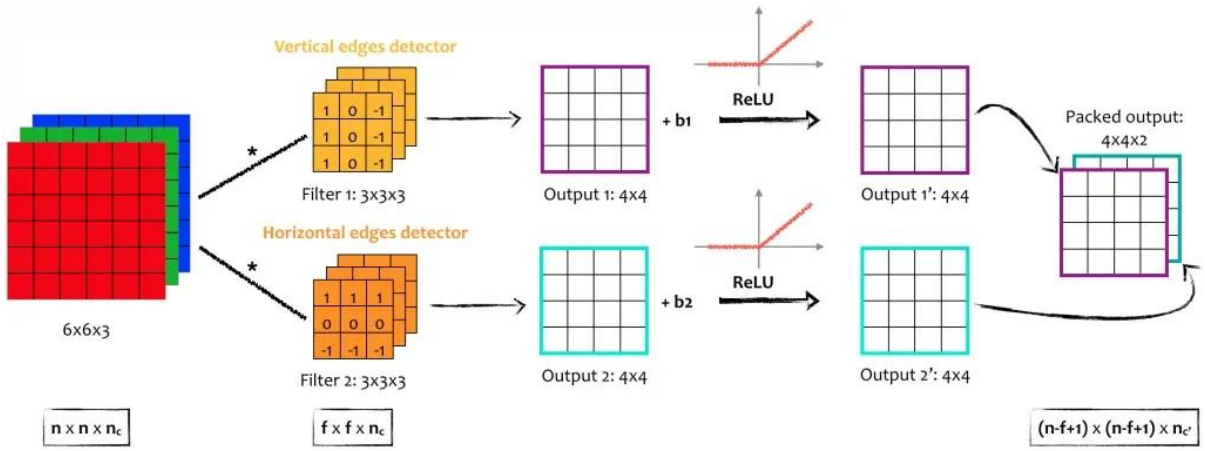
Pooling layer sẽ cho bạn tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling). Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter.

Đó là lý do tại sao CNNs cho ra mô hình với độ chính xác rất cao. Cũng giống như cách con người nhận biết các vật thể trong tự nhiên.

Hướng dẫn cách chọn tham số cho CNN

Nhằm lựa chọn được tham số phù hợp nhất cho CNN, bạn nên lưu ý đến số lượng các yếu tố sau đây: Filter size, pooling size, số convolution và số lần train test.

- Convolution layer: Nếu lớp này có số lượng lớn hơn,chương trình chạy của bạn sẽ càng được cải thiện và tiến bộ. Sử dụng layer với số lượng nhiều có thể giúp các tác động được giảm một cách đáng kể. Trong đa phần các trường hợp, chỉ cần khoảng 3 đến 5 lớp là bạn sẽ thu về kết quả như mong đợi.
- Filter size: Thông thường, các filter size sẽ có kích thước là 3×3 hoặc 5×5.
- Pooling size: Với các hình ảnh thông thường, bạn nên sử dụng loại kích thước 2×2. Nếu đầu vào xuất hiện dạng hình ảnh lớn hơn thì bạn nên chuyển sang dùng loại 4×4.
- Train test: Càng thực hiện nhiều lần train test, bạn càng có nhiều khả năng thu được các parameter tốt nhất, giúp mô hình “thông minh” và hiệu quả hơn.



Hình 3.10: Chọn tham số cho mô hình CNN

3.9.3 Các bước để train model

Chuẩn bị và thu thập dữ liệu

- Thiết lập: Khởi tạo hai danh sách trống data và labels để lưu trữ hình ảnh và nhãn tương ứng. Lặp qua các lớp: Duyệt qua 19 lớp biển báo giao thông, mỗi lớp được biểu diễn bởi một số từ 0 đến 18.
- Đọc ảnh: Trong mỗi thư mục chứa hình ảnh của một lớp cụ thể, đọc từng ảnh. Chuyển đổi ảnh sang màu RGB. Thay đổi kích cỡ ảnh thành 30x30 pixel.
- Chuyển ảnh thành một mảng NumPy. Thêm mảng ảnh vào danh sách data và nhãn tương ứng vào labels.

Xử lý dữ liệu

- Chuẩn hóa dữ liệu ảnh: Chuyển giá trị pixel của mỗi ảnh thành phạm vi [0, 1].
- Chia Dữ liệu: Chia dữ liệu thành hai tập, huấn luyện và kiểm tra, với tỷ lệ 80/20.
- One-Hot Encoding cho nhãn: Chuyển nhãn thành định dạng one-hot encoding để phù hợp với yêu cầu của mô hình học sâu.

Xây dựng mô hình học sâu

- Khởi tạo mô hình: Tạo một mô hình Sequential.
- Thêm các Lớp Mạng Nơ-ron Tích Chập (Conv2D): Thêm các lớp Conv2D với số lượng bộ lọc và kích thước kernel khác nhau, sử dụng hàm kích hoạt ReLU.

- Thêm các Lớp Pooling (MaxPool2D): Giảm kích thước đặc trưng sau mỗi lớp tích chập.
- Thêm Lớp Dropout: Ngăn chặn hiện tượng overfitting bằng cách 'tắt' ngẫu nhiên một số nút.
- Flatten Dữ liệu: Chuyển dữ liệu từ dạng ma trận sang vector.
- Thêm các Lớp Dense: Thêm lớp kết nối đầy đủ với hàm kích hoạt ReLU và softmax.

Tinh chỉnh và huấn luyện mô hình

- Biên dịch mô hình: Sử dụng hàm mất mát categorical_crossentropy, optimizer adam và đánh giá dựa trên accuracy.
- Huấn luyện mô hình: Huấn luyện mô hình với dữ liệu, số lần lặp (epoch) là 15 và kích thước mẻ (batch size) là 64. Sử dụng dữ liệu kiểm tra để đánh giá trong quá trình huấn luyện.

Lưu mô hình để dùng mô hình cho quá trình dự đoán

- Sau khi hoàn tất quá trình huấn luyện, lưu mô hình vào file modeltest.h5.

3.9.4 Các bước để trích xuất biển báo bỏ vào mô hình train

Khởi tạo và tải mô hình

- Sử dụng các thư viện cần thiết: OpenCV (cv2), NumPy (numpy) và Keras.
- Tải mô hình học sâu modeltest.h5 đã được huấn luyện trước ở các bước phía trên.

Định nghĩa các lớp biển báo

Tạo một từ điển prohibition_classes để ánh xạ nhãn số sang tên tương ứng của các biển báo giao thông. (Gồm 19 nhãn)

Xử lý hình ảnh đầu vào

- Tải hình ảnh từ đường dẫn được chỉ định. (gọi là ảnh đầu vào hay ảnh thử nghiệm).
- Chuyển đổi hình ảnh sang không gian màu HSV và xám.
- Áp dụng làm mờ Gaussian và ngưỡng thích nghi.

Phát hiện màu đỏ trong hình ảnh (vì biển báo cấm)

- Định nghĩa khoảng màu đỏ trong không gian màu HSV.
- Tạo mặt nạ để phát hiện các vùng màu đỏ trong ảnh.
- Áp dụng mặt nạ trên hình ảnh để tạo ra hình ảnh chỉ chứa các vùng màu đỏ.

Tính toán Local Binary Pattern (LBP)

- Chuyển đổi hình ảnh màu đỏ sang xám.
- Áp dụng thuật toán LBP để xác định kết cấu trên hình ảnh.

Phát hiện biển báo sử dụng Hough Circle Detection

Sử dụng Hough Circle Detection để tìm các vòng tròn trong hình ảnh, ứng với biển báo giao thông.

Xử lý và Phân loại từng biển báo

- Duyệt qua mỗi vòng tròn phát hiện được.
- Đối với mỗi vòng tròn, cắt ROI (vùng quan tâm) và áp dụng thuật toán LBP.
- Chuyển ROI sang định dạng RGB, thay đổi kích thước và chuẩn hóa.
- Sử dụng mô hình học sâu để phân loại ROI thành một trong các loại biển báo.

Ghi tên nhãn lên hình ảnh đã được đưa vào mô hình

Ghi chú tên của biển báo lên hình ảnh ở vị trí tương ứng.

Lưu và xuất hình ảnh

Lưu hình ảnh đã được xử lý và chú thích vào một file mới.

Quá trình này kết hợp xử lý ảnh truyền thống (như làm mờ, phát hiện cạnh, phân loại màu sắc, LBP, Hough Transform) và học sâu (phân loại sử dụng mô hình đã được huấn luyện) để nhận dạng và phân loại biển báo giao thông từ hình ảnh.

CHƯƠNG 4 - CÁC BƯỚC THỰC THI VÀ KẾT QUẢ THỰC NGHIỆM

4.1 Các bước thực thi

4.1.1 Thực thi để tạo ra mô hình học sâu

Bước 1: Sử dụng thư viện openCV và numpy và dùng Keras để tải mô hình học và dự đoán ảnh

```
from PIL import Image
import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

Bước 2: Chuẩn bị và thu thập dữ liệu

Khởi tạo danh sách dữ liệu và nhãn. Tạo hai danh sách trống, data để lưu trữ hình ảnh và labels để lưu trữ nhãn tương ứng. Định nghĩa số lượng lớp classes là 19, tương ứng với số lượng biển báo giao thông khác nhau cần được phân loại.

```
data = []
labels = []
classes = 19
```

Lấy đường dẫn hiện tại bằng os.getcwd().

Duyệt qua từng lớp biển báo (từ 0 đến 18). Mỗi lớp được lưu trữ trong một thư mục riêng biệt trong thư mục 'TrainData'.

```
cur_path = os.getcwd()
for i in range(classes):
    path = os.path.join(cur_path, 'TrainData', str(i))
    images = os.listdir(path)
```

Trong mỗi thư mục lớp, đọc từng hình ảnh. Mở hình ảnh và chuyển đổi sang màu RGB (convert('RGB')) để đảm bảo hình ảnh đầu vào là màu.

Thay đổi kích thước hình ảnh thành 30x30 pixel để chuẩn hóa kích thước đầu vào cho mô hình học sâu. Chuyển hình ảnh từ định dạng PIL Image sang mảng NumPy.

Thêm hình ảnh vào danh sách data và nhãn tương ứng vào labels. Bắt và in ra ngoại lệ nếu có lỗi khi tải hình ảnh.

```
for a in images:
    try:
        image_path = os.path.join(path, a)
        image = Image.open(image_path).convert('RGB')  # Keep the image in color
        image = image.resize((30, 30))
        image = np.array(image)

        data.append(image)
        labels.append(i)
    except Exception as e:
        print('Error loading image:', image_path, '\nError:', e)
```

Bước 3: Xử lý dữ liệu

Chuẩn hóa dữ liệu ảnh bằng cách chia cho 255.

Chuyển đổi nhãn sang định dạng one-hot encoding.

```
data = np.array(data, dtype='float32') / 255.0
labels = np.array(labels)
```

Bước 4: Chia dữ liệu thành tập huấn luyện và kiểm tra

Tách dữ liệu ra thành 20% tập test và 80% là tập train.

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
y_train = to_categorical(y_train, classes)
y_test = to_categorical(y_test, classes)
```

Bước 5: Xây dựng model CNN

Sử dụng kiến trúc Sequential từ TensorFlow và Keras, cho phép tạo một mô hình với các lớp được thêm vào tuần tự.

- Thêm hai lớp tích chập (Conv2D) với 32 bộ lọc, kích thước kernel là 5x5.
- Hàm kích hoạt relu (Rectified Linear Unit) được sử dụng để thêm phi tuyến tính.
- Lớp đầu tiên cần chỉ định input_shape, ở đây là ảnh 30x30 pixel với 3 kênh màu (RGB).
- MaxPool2D giảm kích thước đầu ra của lớp trước đó, giúp giảm thiểu số lượng tham số và tránh overfitting.
- Dropout(0.25) loại bỏ ngẫu nhiên 25% các nút, giúp ngăn chặn overfitting.

- Hai lớp tích chập khác với 64 bộ lọc và kích thước kernel 3x3, cũng sử dụng hàm kích hoạt relu.
- Flatten() chuyển đổi dữ liệu từ dạng ma trận sang vector trước khi đưa vào lớp kết nối đầy đủ (Dense).
- Lớp Dense với 256 nút và hàm kích hoạt relu.
- Dropout(0.5) loại bỏ 50% các nút để giảm overfitting.
- Lớp Dense cuối cùng với 19 nút, mỗi nút tương ứng với một lớp đầu ra (loại biển báo). Hàm kích hoạt softmax được sử dụng để phân loại đa lớp.

Mô hình này sử dụng các kỹ thuật tiêu chuẩn của mạng CNN, phù hợp cho các tác vụ phân loại hình ảnh, bao gồm các lớp tích chập để học các đặc trưng từ hình ảnh, pooling để giảm kích thước không gian, và các lớp kết nối đầy đủ để phân loại.

```
model = Sequential([
    Conv2D(32, (5, 5), activation='relu', input_shape=(30, 30, 3)),
    Conv2D(32, (5, 5), activation='relu'),
    MaxPool2D(2, 2),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPool2D(2, 2),
    Dropout(0.25),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(19, activation='softmax')
])
```

Bước 6: Tinh chỉnh và huấn luyện mô hình

Biên dịch mô hình sử dụng hàm mất mát categorical_crossentropy và optimizer adam. Huấn luyện mô hình với số lượng epochs là 15 và kích thước batch là 64.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=64, epochs=15, validation_data=(X_test, y_test))
```

Bước 7: Lưu model cho dự đoán sau khi trích xuất

```
model.save('modeltest.h5')
```

4.1.2 Thực thi trích xuất biển báo và dự đoán

Bước 1: Sử dụng thư viện openCV và numpy và dùng Keras để tải mô hình học và dự đoán ảnh

```
import cv2
import numpy as np
from keras.models import load_model

model = load_model('modeltest.h5')
```

Bước 2: Gán nhãn cho các tập dữ liệu học được

Tạo một từ điển prohibition_classes để ánh xạ từ số nguyên sang tên của các loại biển báo giao thông.

```
prohibition_classes = {
                      1: 'Speed limit (20km/h)',
                      2: 'Speed limit (30km/h)',
                      3: 'Speed limit (50km/h)',
                      4: 'Speed limit (60km/h)',
                      5: 'Speed limit (70km/h)',
                      6: 'Speed limit (80km/h)',
                      7: 'No car',
                      8: 'No 2.5 car',
                      9: 'No byCicle',
                      10: 'No passing',
                      11: 'No passing veh over 3.5 tons',
                      12: 'No vehicles',
                      13: 'Veh > 3.5 tons prohibited',
                      14: 'No entry',
                      15: 'Beware of ice/snow',
                      16: 'End of no passing',
                      17: 'No Turn Left',
                      18: 'No Turn Right',
                      19: 'Stop'}
```

Bước 3: Đọc ảnh đầu vào

```
# Load the original image
image_path = r'input1copy.png'
image = cv2.imread(image_path)
```

Bước 4: Chuyển đổi hình ảnh sang không gian màu HSV và xám.

Áp dụng làm mờ Gaussian và ngưỡng thích nghi trên hình ảnh xám.

```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
thresh = cv2.adaptiveThreshold(blur, 255, 1, 1, 11, 2)
```

Bước 5: Phát hiện màu đỏ trong ảnh để phát hiện biển báo cấm

Sử dụng phép toán bitwise AND để tạo ra hình ảnh chỉ chứa các vùng màu đỏ.

```
# Define range of RED color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
lower_red2 = np.array([170, 50, 50])
upper_red2 = np.array([180, 255, 255])

# Threshold the HSV image to get only red colors
mask1 = cv2.inRange(hsv, lower_red, upper_red)
mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
mask = mask1 + mask2

# Bitwise-AND mask and original image
red_hue_image = cv2.bitwise_and(image, image, mask=mask)
```

Bước 6: Tính toán Local Binary Pattern (LBP)

Tính toán LBP: Hàm này tính giá trị LBP cho mỗi pixel. LBP so sánh giá trị cường độ của mỗi pixel với các pixel xung quanh nó trong một khu vực lân cận nhỏ. Nếu pixel lân cận có giá trị cường độ cao hơn hoặc bằng pixel trung tâm, nó được gán giá trị 1, ngược lại là 0. Các giá trị nhị phân này sau đó được chuyển đổi thành giá trị thập phân, tạo ra một mô tả kết cấu tại điểm ảnh đó.

Chuyển đổi hình ảnh màu đỏ sang xám để chuẩn bị cho việc tính toán LBP.

- Tạo Mảng LBP: Mảng lbp_result được tạo ra với cùng kích thước như hình ảnh xám gray_red_hue. Điều này đảm bảo mỗi pixel trong hình ảnh xám sẽ có một giá trị LBP tương ứng.
- Áp dụng Hàm LBP: Hàm calculate_lbp_pixel được áp dụng lên từng pixel của hình ảnh xám. Điều này tạo ra một hình ảnh LBP hoàn chỉnh, nơi mỗi pixel mang thông tin kết cấu dựa trên các pixel xung quanh nó.

```
# Function to calculate LBP value for a pixel
def calculate_lbp_pixel(img, x, y):
    center = img[x, y]
    values = []
    # Define the neighborhood points
    points = [(x-1, y-1), (x-1, y), (x-1, y+1),
              (x, y+1), (x+1, y+1), (x+1, y),
              (x+1, y-1), (x, y-1)]
    for point_x, point_y in points:
        values.append(1 if img[point_x, point_y] >= center else 0)
    # Convert binary values to decimal
    lbp_value = sum([val * (2 ** idx) for idx, val in enumerate(values)])
    return lbp_value

rows, cols = gray_red_hue.shape
lbp_result = np.zeros_like(gray_red_hue, dtype=np.uint8)

for i in range(1, rows - 1):
    for j in range(1, cols - 1):
        lbp_result[i, j] = calculate_lbp_pixel(gray_red_hue, i, j)
```

Bước 7: Dùng HoughCircles để phát hiện ra hình tròn của biển báo

```
# Use Hough Circle Detection
circles = cv2.HoughCircles(
    gray_red_hue,
    cv2.HOUGH_GRADIENT,
    dp=1,
    minDist=50,
    param1=50,
    param2=30,
    minRadius=30,
    maxRadius=70
)
```

- gray_red_hue: Hình ảnh đầu vào ở chế độ xám. Thường thì người ta sử dụng Phép biến đổi Hough Circle trên hình ảnh xám để đơn giản hóa quá trình xử lý.
- cv2.HOUGH_GRADIENT: Phương pháp phát hiện được sử dụng. Phương pháp Gradient Hough là một biến thể hiệu quả hơn của Phép biến đổi Hough tiêu chuẩn để phát hiện đường tròn.
- dp=1: Tỉ lệ nghịch đảo giữa độ phân giải của bộ tích tụ và độ phân giải của hình ảnh. Trong trường hợp này, sử dụng cùng độ phân giải như hình ảnh đầu vào.
- minDist=50: Khoảng cách tối thiểu giữa các trung tâm của các đường tròn được phát hiện. Tham số này giúp tránh phát hiện nhiều đường tròn ở gần nhau.
- param1=50: Ngưỡng cao hơn trong hai ngưỡng được chuyển cho bộ lọc biên Canny. Nó được áp dụng cho hình ảnh đầu vào trước khi thực hiện Phép biến đổi Hough. Việc điều chỉnh giá trị này có thể ảnh hưởng đến chất lượng phát hiện đường tròn.
- param2=30: Ngưỡng của bộ tích tụ để phát hiện đường tròn. Đây là một tham số quan trọng ảnh hưởng đến độ nhạy của quá trình phát hiện đường tròn. Giá trị thấp sẽ dẫn đến việc phát hiện nhiều đường tròn hơn, bao gồm cả kết quả giả mạo, trong khi giá trị cao sẽ làm cho quá trình phát hiện nghiêm túc hơn.
- minRadius=30: Bán kính tối thiểu của các đường tròn cần phát hiện.
- maxRadius=70: Bán kính tối đa của các đường tròn cần phát hiện.

Bước 8: Xử lý khoanh và xử lý ảnh trước khi đưa vào mô hình

Đề yêu cầu màu khác nhau cho mỗi biển báo, nên đã gắng color index cho 2-3 biển báo khác nhau.

```
color_index = 0
colors = [(0, 255, 0), (0, 0, 255), (255, 0, 0)]
```

Kiểm tra xem có phát hiện được vòng tròn (biển báo) không. Nếu có, chuyển đổi dữ liệu vòng tròn thành kiểu số nguyên không dấu 16-bit.

```
if circles is not None:
    circles = np.uint16(np.around(circles))
```

Lấy tọa độ và bán kính của mỗi vòng tròn phát hiện được.

Tính toán kích thước ROI và vẽ hình chữ nhật xung quanh vòng tròn.

Khoanh những cái hình tròn nên khoanh và đổi thành vị trí color thích hợp như định nghĩa.

```
if counts[0] / np.sum(counts) < 0.8:
    cv2.rectangle(image, (x, y), (x+w, y+h), current_color, 2)
```

Ảnh trước khi được đưa vào dự đoán. Cắt ROI từ hình ảnh gốc.

Chuyển đổi màu từ BGR sang RGB, thay đổi kích thước và chuẩn hóa dữ liệu ảnh.

```
roi_color = image[y:y+h, x:x+w]
roi_color_rgb = cv2.cvtColor(roi_color, cv2.COLOR_BGR2RGB)
roi_color_resized = cv2.resize(roi_color_rgb, (30, 30))
roi_color_resized = roi_color_resized.astype('float32') / 255.0
image_array = np.expand_dims(roi_color_resized, axis=0)
```

Bước 9: Dự đoán ảnh

Sử dụng mô hình học sâu đã tải để dự đoán loại biển báo.

Lấy tên biển báo từ từ điển prohibition_classes.

```
# Dự đoán
pred_probabilities = model.predict(image_array)
pred_class = np.argmax(pred_probabilities, axis=1)[0]
sign = prohibition_classes[pred_class + 1]
```

Ghi chú tên biển báo lên hình ảnh ở vị trí tương ứng.

```
# Ghi chú tên biển báo lên ảnh
cv2.putText(image, sign, (x, y-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, current_color, 2)
```

Bước 10: In ra kết quả

```
output_path_prohibition_signs = 'Output.png'
cv2.imwrite(output_path_prohibition_signs, image)
```

4.2 Kết quả thử nghiệm

Đây là kết quả thử nghiệm sau khi thực thi.

Input: Ảnh test

Output: Ảnh có khoanh vùng và in tên biển báo

Hình 4.1: Ảnh đầu vào thử nghiệm 1

Hình 4.2: Ảnh kết quả thử nghiệm 1

Hình 4.3: Ảnh đầu vào thử nghiệm 2

Hình 4.4: Ảnh kết quả thử nghiệm 2

Hình 4.5: Ảnh đầu vào thử nghiệm 3

Hình 4.6: Ảnh kết quả thử nghiệm 3

Hình 4.7: Ảnh đầu vào thử nghiệm 4

Hình 4.8: Ảnh kết quả thử nghiệm 4

Ảnh test thử không có trong tập train sẽ được học sai:

Hình 4.9: Ảnh đầu vào thử nghiệm 5

Hình 4.10: Ảnh kết quả thử nghiệm 5

TÀI LIỆU THAM KHẢO

[1] Digital Image Processing, Rafael C. Gonzalez, Richard E. Woods, 2008.

[2] Image Processing, Analysis, and Machine Vision, Milan Sonka, Vaclav Hlavac, Roger Boyle, 2014.

[3] Computer Vision: Algorithms and Applications, Richard Szeliski, 2010.

[4] Digital Image Processing Using MATLAB, Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, 2009.

[5] Handbook of Medical Imaging: Processing and Analysis, Isaac N. Bankman, 2000.