TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Ứng dụng nhận dạng khuôn mặt đeo khẩu trang dùng CNN

Giảng viên hướng dẫn: Từ Lãng Phiêu

Sinh viên: Võ Ngọc Phú - 3120410404
Nguyễn Hoàng Phúc - 3120410409

TP. HỒ CHÍ MINH, THÁNG 2/2024

Mục lục

1 Giới thiệu

1.1 Ứng dụng phát hiện đeo khẩu trang

Khẩu trang là một vật dụng thiết yếu trong cuộc sống hiện nay khi chất lượng không khi trong các đô thị hiện nay đang có tình trạng ô nhiễm, ngoài ra khẩu trang còn giúp hạn chế lây lan dịch bệnh thông qua không khí, đặc biệt là đại dịch covid-19 vừa mới xảy ra gần đây. Chính vì vậy mà nếu ta có một công cụ giúp nhận diện xem 1 người có đang đeo khẩu trang hay không sẽ rất hữu ích trong một vài trường hợp.

1.2 Mục đích của ứng dụng

Mục đích chính của bải toán là hỗ trợ kiểm tra tuân thủ quy định về đeo khẩu trang, có thể là một khu vực bắt buộc đeo khẩu trang để đảm bảo sự an toàn vệ sinh như nhà bếp, nhà ga, bệnh viện,... hoặc một khu vực cấm đeo khẩu trang để đảm bảo an ninh như ngân hàng, kho lưu trữ,...
Bài toán có thể được tích hợp vào camera an ninh để ghi lại các trường hợp đeo khẩu trang giúp đảm bảo an toàn và sức khỏe cộng đồng, giảm nguy cơ lây nhiễm trong các môi trường công cộng, hoặc cải thiện quản lý an ninh trong các tổ chức, cơ quan.

1.3 Phạm vi của báo cáo

1.3.1 Phạm vi nghiên cứu

Bài toán sử dụng ngôn ngữ Python và hai thành phần chính là thư viện opencv và thư viện keras:

- Opencv: lấy luồng stream từ camera và trích xuất ra các khuôn mặt có trong khung hình.
- Keras: từ dữ liệu được trích xuất, đưa vào model đã được xây dựng và huấn luyện để phân loại khuôn mặt có đeo khẩu trang hay không.

1.3.2 Phạm vi ứng dụng

- Xử lý hình ảnh từ camera để kiểm tra tuân thủ quy định về đeo khẩu trang trong các khu vực công cộng, hoặc tăng cường an ninh trong các tổ chức, cơ quan..
- Hình ảnh từ camera nên có đủ độ sáng và khoản cách giữa khuông mặt đến camera phải phù hợp
- Bài toán chỉ xác định có hoặc không đeo khẩu trang mà không xác định các đặt điểm khác

2 Cơ sở lý thuyết

2.1 Thư viện OpenCV

OpenCV (viết tắt của Open Source Computer Vision Library) là một thư viện mã nguồn mở chuyên dùng trong xử lý ảnh và thị giác máy tính. Công nghệ cung cấp các công cụ và thư viện để phân tích và xử lý ảnh, video từ việc xác định các đối tượng trong ảnh đến việc nhận diện khuôn mặt hoặc theo dõi chuyển động khác.

OpenCV được viết bằng C và hỗ trợ nhiều ngôn ngữ và nền tảng, bao gồm C, Python, Java và Android. Nó có thể được sử dụng trên nhiều hệ điều hành khác nhau, bao gồm Windows, Linux, macOS, iOS và Android. OpenCV cũng có cách triển khai được tối ưu hóa cao cho các nền tảng dựa trên ARM, chẳng hạn như Raspberry Pi, khiến nó rất phù hợp để sử dụng trong các hệ thống nhúng.
Các ứng dụng của OpenCV:

- Hình ảnh street view
- Kiểm tra và giám sát tự động
- Robot và xe hơi tự lái
- Phân tích hình ảnh y học
- Tìm kiếm và phục hồi hình ảnh/video
- Phim – cấu trúc 3D từ chuyển động
- Nghệ thuật sắp đặt tương tác

Các tính năng và các module phổ biến của OpenCV:

- Xử lý và hiển thị Hình ảnh/ Video/ I/O (core, imgproc, highgui)
- Phát hiện các vật thể (objdetect, features2d, nonfree)
- Geometry-based monocular hoặc stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning clustering (ml, flann)
- CUDA acceleration (gpu)

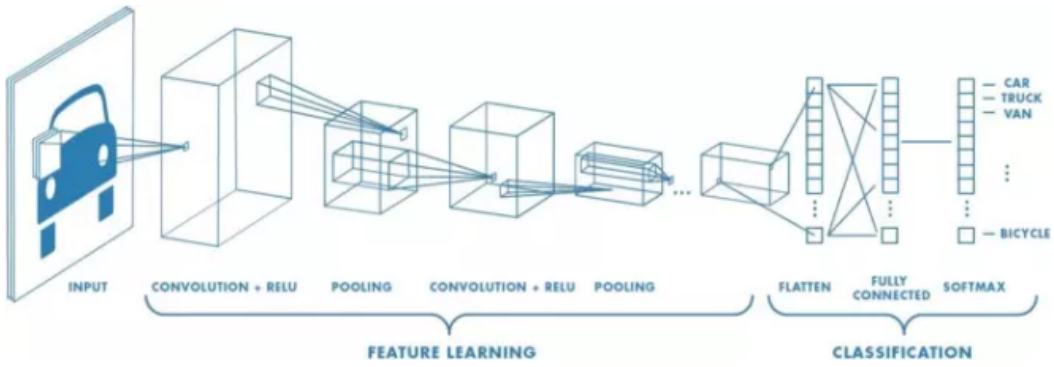
Trong ứng dụng này, ta sẽ tận dụng module cv2.dnn của thư viện opencv để load một model phát hiện khuôn mặt đã được huấn luyện có sẵn.

2.2 CNN (Convolutional Neural Network)

Lĩnh vực học máy đã trải qua một sự thay đổi đáng kể trong thời gian gần đây, với sự bùng nổ của Mạng Nơ-ron Nhân tạo (ANN). Những mô hình tính toán được lấy cảm hứng từ sinh học này có khả năng vượt xa hiệu suất của các dạng trước đó của trí tuệ nhân tạo trong các nhiệm vụ học máy phổ biến. Một trong những dạng kiến trúc ANN ấn tượng nhất là Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập), đây là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như

hiện nay.
CNN được sử dụng nhiều trong các bài toán phân loại hình ảnh, nhận dạng các object trong ảnh.

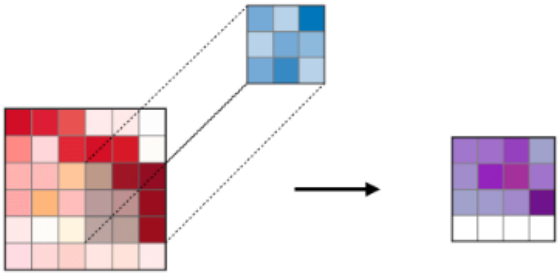
Cấu trúc của mạng CNN
Mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernals), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1. Hình dưới đây là toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.



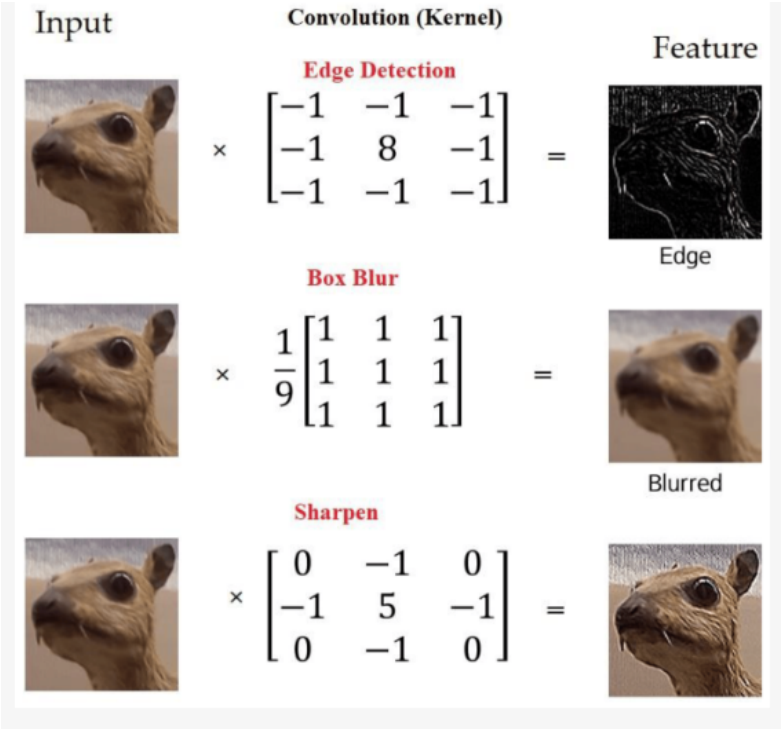
Hình 1: Cấu trúc mạng CNN

- Lớp tích chập - Convolution Layer
  Tích chập là lớp đầu tiên để trích xuất các đặc tính từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các đặc tính của hình ảnh bằng cách sử dụng 1 bộ lọc (kernel hoặc filter) để thực hiện phép tích chập
  Quá trình tích chập bao gồm các bước sau:

  1. Di chuyển kernel qua toàn bộ bức ảnh theo từng bước (stride).
  2. Tại mỗi vị trí, tính tích chập của kernel với các giá trị pixel trong vùng cửa sổ của ảnh (có kích thước tương ứng với kích thước của kernel).
  3. Kết quả của mỗi phép tích chập tạo ra một giá trị mới trong ma trận đầu ra gọi là feature map hoặc activation map. Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.



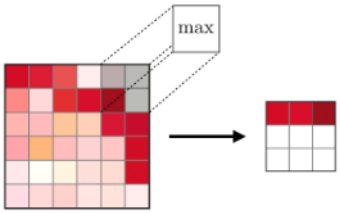
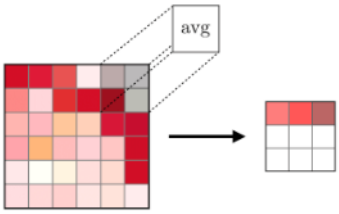
Hình 2: Lớp tích chập



Hình 3: Các bộ lọc khác nhau

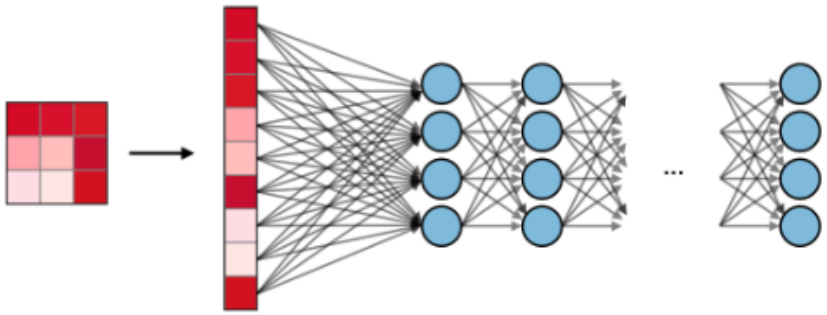
- Lớp gộp - Pooling Layer
  Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau.

Hình 4: Hai loại pooling

Kiểu	Max pooling	Average pooling
Chức năng	Từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng	Từng phép pooling tính trung bình các giá trị trong khu vực mà nó đang được áp dụng
Minh họa		
Nhận xét	• Bảo toàn các đặc trưng đã phát hiện • Được sử dụng thường xuyên	• Giảm kích thước feature map • Được sử dụng trong mạng LeNet

- Lớp Fully Connected (FC)
  Tầng kết nối đầy đủ (FC) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.

Hình 5: Lớp Fully Connected (FC)



- Đường viền - Padding
  Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:

  Chèn thêm các số 0 vào 4 đường biên của hình ảnh (padding).
  Cắt bớt hình ảnh tại những điểm không phù hợp với kernel.

- Hàm phi tuyến - ReLU
  - ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là: f (x) = max (0, x).

- Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.
- Có 1 số hà phi tuyến khác như tanh, sigmoid cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

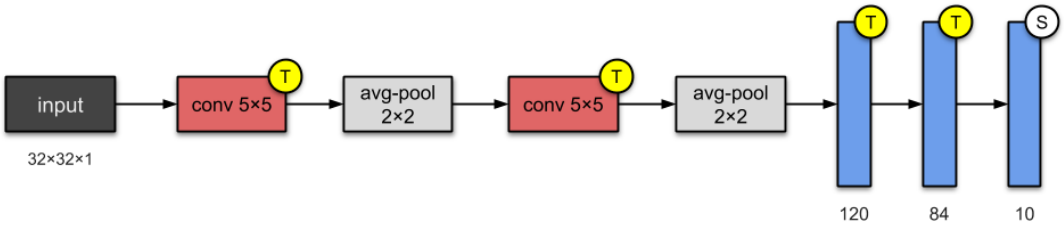
Các mạng CNN tiểu biểu

- LeNet-5(1998)
  LeNet-5 là kiến trúc đầu tiên áp dụng mạng tích chập 2 chiều của giáo sư Yan Lecun, cha đẻ của kiến trúc CNN. Model ban đầu khá đơn giản và chỉ bao gồm 2 convolutional layers + 3 fully-connected layers. Mặc dù đơn giản nhưng nó có kết quả tốt hơn so với các thuật toán machine learning truyền thống khác trong phân loại chữ số viết tay như SVM, kNN.

  Trong kiến trúc mạng nơ ron đầu tiên, để giảm chiều dữ liệu, Yan Lecun sử dụng Sub-Sampling Layer là một Average-Pooling Layer (các layer nhằm mục đích giảm chiều dữ liệu mà không thay đổi đặc trưng chúng ta còn gọi là Sub-Sampling Layer). Kiến trúc này khó hội tụ nên ngày nay chúng được thay thế bằng Max-Pooling.

  Đầu vào của mạng LeNet có kích thước nhỏ (chỉ 32x32) và ít layers nên số lượng tham số của nó chỉ khoảng 60 nghìn.

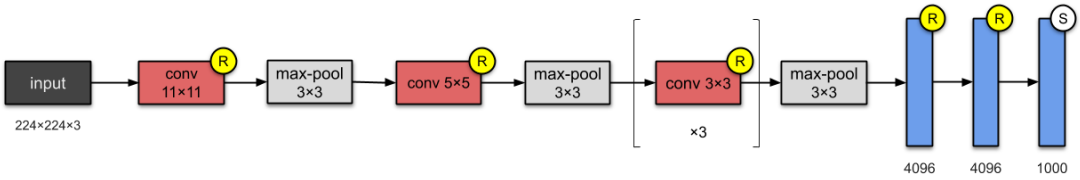
Hình 6: Mạng CNN Lenet-5



- AlexNet(2012)
  AlexNet là mạng CNN được giới thiệu vào năm 2012 bởi Alex Krizhevsky và dành chiến thắng trong cuộc thi ImageNet với cách biệt khá lớn so với vị trí thứ hai. Lần đầu tiên Alex net đã phá vỡ định kiến trước đó cho rằng các đặc trưng được học từ mô hình sẽ không tốt bằng các đặc trưng được tạo thủ công (thông qua các thuật toàn SUFT, HOG, SHIFT). Ý tưởng của AlexNet dựa trên LeNet của Yan Lecun và cải tiến ở các điểm:
  - Tăng kích thước đầu vào và độ sâu của mạng.
  - Sử dụng các bộ lọc (kernel hoặc filter) với kích thước giảm dần qua các layers để phù hợp với kích thước của đặc trưng chung và đặc trưng riêng.
  - Sử dụng local normalization để chuẩn hóa các layer giúp cho quá trình hội tụ nhanh hơn.
  - Lần đầu tiên sử dụng activation là ReLU (Rectified Linear Unit) thay cho Sigmoid.

- Resnet50(2015)
  ResNet là kiến trúc sớm nhất áp dụng batch normalization. Mặc dù là một mạng rất sâu

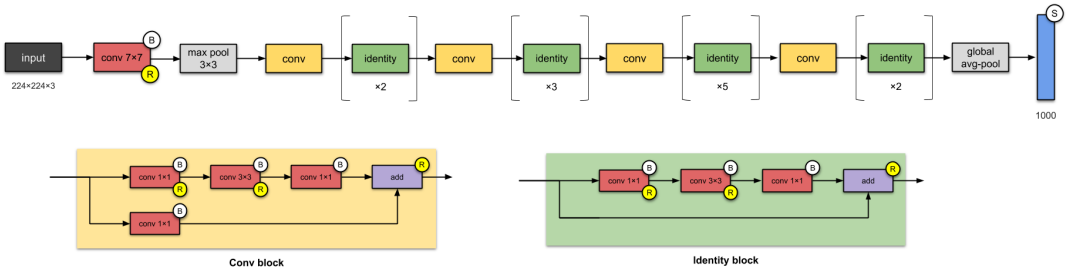
Hình 7: Mạng CNN AlexNet



khi có số lượng layer lên tới 152 nhưng nhờ áp dụng những kỹ thuật đặc biệt mà ta sẽ tìm hiểu bên dưới nên kích thước của ResNet50 chỉ khoảng 26 triệu tham số. Kiến trúc với ít tham số nhưng hiệu quả của ResNet đã mang lại chiến thắng trong cuộc thi ImageNet năm 2015.

Những kiến trúc trước đây thường cải tiến độ chính xác nhờ gia tăng chiều sâu của mạng CNN. Nhưng thực nghiệm cho thấy đến một ngưỡng độ sâu nào đó thì độ chính xác của mô hình sẽ bão hòa và thậm chí phản tác dụng và làm cho mô hình kém chính xác hơn. Khi đi qua quá nhiều tầng độ sâu có thể làm thông tin gốc bị mất đi thì các nhà nghiên cứu của Microsoft đã giải quyết vấn đề này trên ResNet bằng cách sử dụng kết nối tắt.

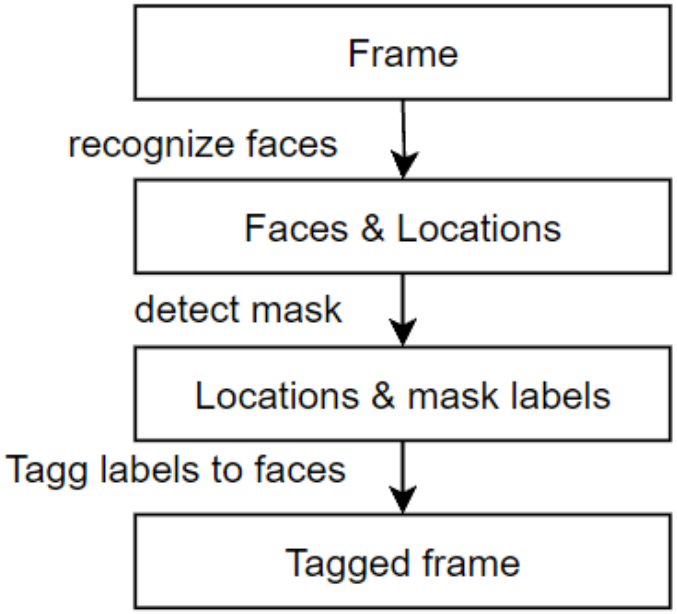
Hình 8: Mạng CNN ResNet50



2.3 Sự kết hợp giữa OpenCV và CNN trong việc nhận diện khuôn mặt đeo khẩu trang

1. Đọc luồng hình ảnh từ camera và phát hiện khuôn mặt bằng OpenCV
2. Nếu có khuôn mặt được phát hiện thì đưa nó vào mạng CNN để phân loại có hoặc không đeo khẩu trang
3. Dùng OpenCV để đánh viền khuông mặt và hiển thị nhãn tương ứng từ CNN

Hình 9: Workflow của ứng dụng



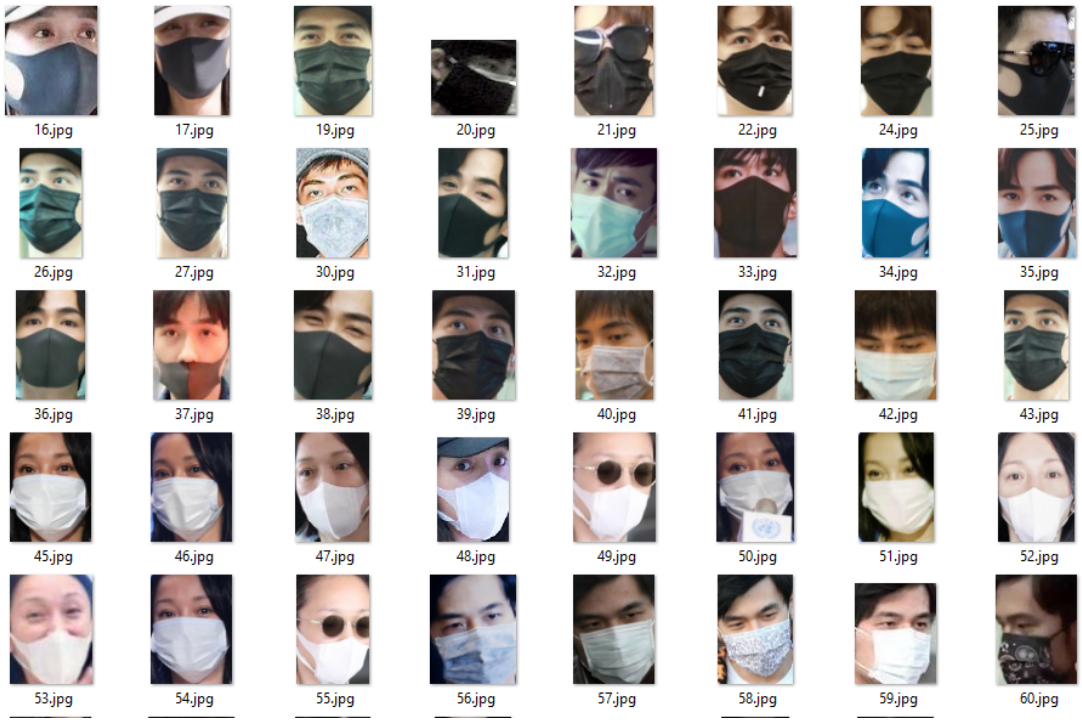
3 Thiết kế và triển khai ứng dụng

3.1 Thu thập dữ liệu và tiền xử lý

Thu thập dữ liệu: Dữ liệu dùng để huấn luyện mô hình này được phân làm 2 class: có đeo khẩu trang(with-mask) và khôgn đeo khẩu trang(without-mask). Ta sẽ thu thập các hình ảnh khuôn mặt có đeo khẩu trang và không đeo khẩu trang và tổng hợp chúng vào 2 thư mục:

- thư mục with-mask với 1915 bức ảnh
- thư mục without-mask 1918 bức ảnh

Hình 10: một vài hình ảnh with-mask



Tiền xử lý dữ liệu:

Để tiền xử lý dữ liệu ta sẽ có các bước sau đây:

1. Tải dữ liệu từ thư mục gốc và gán vào hai mảng là data và labels. mảng data chứa hình ảnh còn mảng labels chứa nhãn của hình ảnh tương ứng.
2. Encode mảng labels thành dạng mảng nhị phân mà mô hình mạng CNN có thể hiểu được
3. Biến đổi hai mảng thành numpy array
4. Chia mỗi 2 mảng thành 2 tập: tập train và tập test, tỉ lệ giữa train và test là 8/2 và thứ tự item trong các tập sẽ được xáo trộn ngẫu nhiên. trainX, testX sẽ là 2 tập dữ liệu hình ảnh, trainY, textY 2 sẽ là tập nhãn.

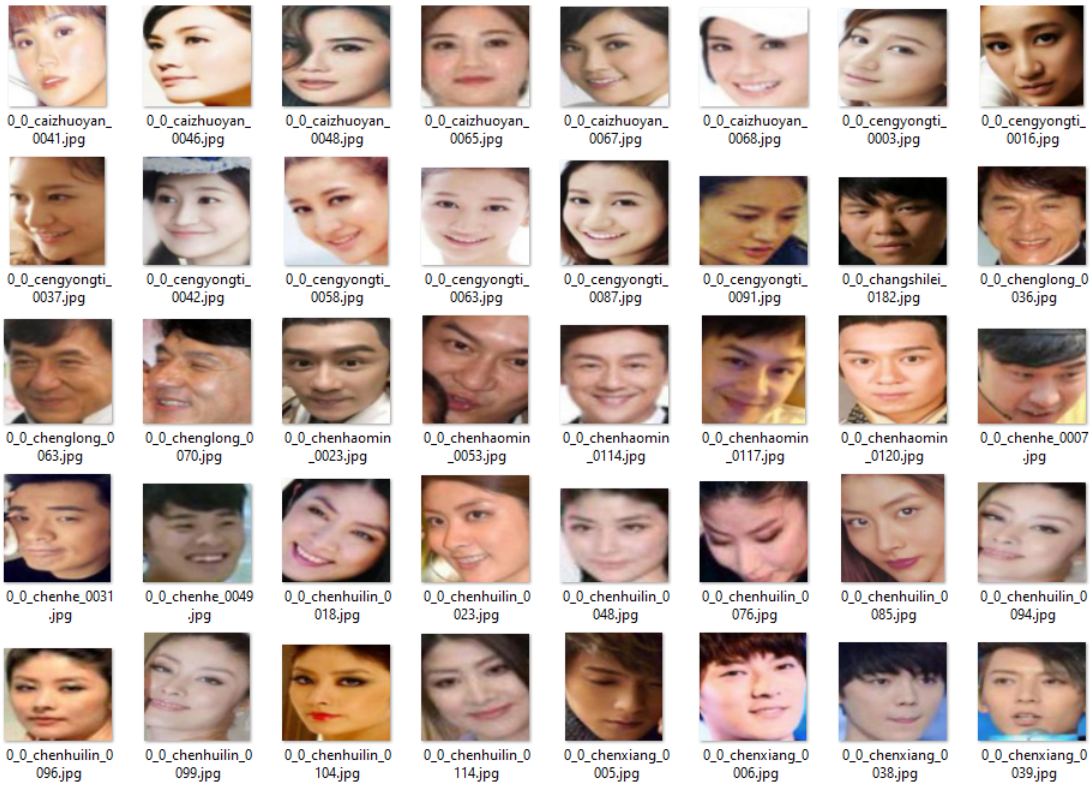
Đoạn code thực hiện tiền xử lý dữ liệu:

```
DIRECTORY = os.path.join(os.getcwd(), "dataset")
CATEGORIES = ["with_mask", "without_mask"]

data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
```

Hình 11: một vài hình ảnh without-mask




```
  for img in os.listdir(path):
    img_path = os.path.join(path, img)
    image = load_img(img_path, target_size=(224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)
    data.append(image)
    labels.append(category)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
  test_size=0.20, stratify=labels, random_state=42)
```

3.2 Xây dựng mô hình CNN sử dụng Keras

Keras: Keras là một thư viện mã nguồn mở phổ biến trong lĩnh vực học sâu (deep learning) và máy học (machine learning). Ban đầu được phát triển bởi François Chollet, Keras là một giao diện lập trình ứng dụng (API) dễ sử dụng, linh hoạt và mạnh mẽ cho việc xây dựng và huấn luyện các mạng neural networks.
Các đặc điểm của Keras:

- Dễ sử dụng: Keras cung cấp một giao diện lập trình đơn giản và dễ hiểu, giúp người dùng tập trung vào việc xây dựng mô hình mạng neural mà không cần quan tâm nhiều đến chi tiết cài đặt.
- Modular: Keras được thiết kế để có tính linh hoạt cao và có cấu trúc modular, cho phép người dùng dễ dàng xây dựng và kết hợp các lớp (layers), hàm kích hoạt (activation functions), và các phần tử khác của một mạng neural.
- Hỗ trợ nhiều backend: Keras có thể chạy trên nhiều backend khác nhau, bao gồm TensorFlow, Theano và Microsoft Cognitive Toolkit (CNTK), cho phép người dùng chọn lựa backend phù hợp với nhu cầu cụ thể.
- Hỗ trợ nhiều loại mô hình: Keras hỗ trợ xây dựng và huấn luyện nhiều loại mô hình học sâu, bao gồm mạng neural feedforward, mạng neural convolutional (CNNs), mạng neural hồi quy (RNNs), và nhiều loại mô hình khác.
- Tích hợp với TensorFlow: Kể từ TensorFlow 2.0, Keras đã trở thành một phần của TensorFlow, giúp việc sử dụng Keras trở nên dễ dàng hơn khi làm việc với TensorFlow.

Trong đề tài này, ta sẽ sử dụng một kiến trúc mạng CNN có sẵn được tích hợp trong Keras, đó chính là mạng MobileNet. MobileNet là một kiến trúc mạng CNN được thiết kế đặc biệt để có thể chạy trên các thiết bị di động có tài nguyên hạn chế như điện thoại thông minh và các thiết bị nhúng. Thay vì sử dụng các lớp tích chập thông thường, MobileNet sử dụng các lớp tích chập động (depthwise separable convolution), giúp giảm đáng kể số lượng tham số và phép tính cần thiết, trong khi vẫn giữ được hiệu suất của mạng. MobileNet cung cấp nhiều phiên bản với các cấp độ độ sâu khác nhau, từ MobileNetV1 đến MobileNetV3. MobileNet có thể dễ dàng được triển khai trên các nền tảng như TensorFlow, Keras, và TensorFlow Lite

Dưới đây là đoạn code khởi tạo model sử dụng MobileNet bằng Keras:

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model

baseModel = MobileNetV2(weights="imagenet", include_top=False,
  input_tensor=Input(shape=(224, 224, 3)))

headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
```

```
16headModel = Dropout(0.5)(headModel)
17headModel = Dense(2, activation="softmax")(headModel)
18
19model = Model(inputs=baseModel.input, outputs=headModel)
20
21for layer in baseModel.layers:
22   layer.trainable = False
```

Ta sẽ load model MobileNet từ Keras, loại bỏ layer FC của nó và pooling để giảm bớt số lượng tham số, sau đó thay layer FC(Fully Connected) để phù hợp với yêu cầu của bài toán. Ta sẽ giữ nguyên các trọng số của các lớp thuộc MobileNet để giữ lại các đặc trưng đã học từ dữ liệu, giúp tăng tốc quá trình huấn luyện và giảm nguy cơ overfitting.

3.3 Huấn luyện mô hình

Khai báo các super parameter:

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
```

Tối ưu model và huấn luyện:

```
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt,
  metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit(
  aug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

Hình 12: Quá trình huấn luyện mô hình

```
95/95 [==============================] - 110s 1s/step - loss: 0.4056 - accuracy: 0.8622 - val_loss: 0.1460 - val_accuracy: 0.9844
Epoch 2/20
95/95 [==============================] - 114s 1s/step - loss: 0.1504 - accuracy: 0.9674 - val_loss: 0.0747 - val_accuracy: 0.9883
Epoch 3/20
95/95 [==============================] - 95s 996ms/step - loss: 0.1079 - accuracy: 0.9717 - val_loss: 0.0577 - val_accuracy: 0.9870
Epoch 4/20
95/95 [==============================] - 107s 1s/step - loss: 0.0787 - accuracy: 0.9792 - val_loss: 0.0460 - val_accuracy: 0.9922
Epoch 5/20
95/95 [==============================] - 109s 1s/step - loss: 0.0683 - accuracy: 0.9806 - val_loss: 0.0411 - val_accuracy: 0.9909
Epoch 6/20
95/95 [==============================] - 107s 1s/step - loss: 0.0590 - accuracy: 0.9835 - val_loss: 0.0378 - val_accuracy: 0.9896
Epoch 7/20
95/95 [==============================] - 112s 1s/step - loss: 0.0551 - accuracy: 0.9832 - val_loss: 0.0357 - val_accuracy: 0.9909
Epoch 8/20
95/95 [==============================] - 97s 1s/step - loss: 0.0504 - accuracy: 0.9865 - val_loss: 0.0341 - val_accuracy: 0.9909
Epoch 9/20
95/95 [==============================] - 99s 1s/step - loss: 0.0417 - accuracy: 0.9875 - val_loss: 0.0314 - val_accuracy: 0.9922
Epoch 10/20
95/95 [==============================] - 100s 1s/step - loss: 0.0425 - accuracy: 0.9871 - val_loss: 0.0313 - val_accuracy: 0.9909
Epoch 11/20
95/95 [==============================] - 104s 1s/step - loss: 0.0404 - accuracy: 0.9881 - val_loss: 0.0302 - val_accuracy: 0.9909
Epoch 12/20
95/95 [==============================] - 103s 1s/step - loss: 0.0484 - accuracy: 0.9838 - val_loss: 0.0308 - val_accuracy: 0.9909
Epoch 13/20
95/95 [==============================] - 103s 1s/step - loss: 0.0336 - accuracy: 0.9904 - val_loss: 0.0282 - val_accuracy: 0.9922
Epoch 14/20
95/95 [==============================] - 103s 1s/step - loss: 0.0311 - accuracy: 0.9921 - val_loss: 0.0277 - val_accuracy: 0.9922
Epoch 15/20
95/95 [==============================] - 109s 1s/step - loss: 0.0334 - accuracy: 0.9881 - val_loss: 0.0329 - val_accuracy: 0.9909
Epoch 16/20
95/95 [==============================] - 101s 1s/step - loss: 0.0275 - accuracy: 0.9911 - val_loss: 0.0340 - val_accuracy: 0.9909
Epoch 17/20
95/95 [==============================] - 104s 1s/step - loss: 0.0334 - accuracy: 0.9891 - val_loss: 0.0271 - val_accuracy: 0.9909
Epoch 18/20
95/95 [==============================] - 107s 1s/step - loss: 0.0278 - accuracy: 0.9895 - val_loss: 0.0253 - val_accuracy: 0.9922
Epoch 19/20
95/95 [==============================] - 105s 1s/step - loss: 0.0264 - accuracy: 0.9911 - val_loss: 0.0262 - val_accuracy: 0.9935
Epoch 20/20
95/95 [==============================] - 128s 1s/step - loss: 0.0238 - accuracy: 0.9927 - val_loss: 0.0253 - val_accuracy: 0.9935
```

3.4 Triển khai ứng dụng với OpenCV

Để sử dụng mô hình sau khi huấn luyện, ta sẽ lưu trữ nó:

```
model.save("mask_detector.model", save_format="h5")
```

Đọc các frame ảnh từ webcam và resize chúng với chiều rộng tối đa là 400px:

```
while True:
frame = vs.read()
frame = imutils.resize(frame, width=400)
```

Đưa vào hàm xử lý chính:

```
(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
```

Hàm này sẽ làm các công việc sau:

- Phát hiện và trích xuất các khuôn mặt có trong frame

```
 blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
   (104.0, 177.0, 123.0))
faceNet.setInput(blob)
detections = faceNet.forward()
 box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
 face = frame[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
faces.append(face)
locs.append((startX, startY, endX, endY))
```

- Nếu có khuôn mặt, thực hiện dự đoán có hay không đeo khẩu trang

```
if len(faces) > 0:
  faces = np.array(faces, dtype="float32")
  preds = maskNet.predict(faces, batch_size=32)
```

- Trả về kết quả dự đoán và tọa độ của khuôn mắt được dự đoán tương ứng

```
return (locs, preds)
```

Sau khi có kết quả, ta đánh dấu lại và hiển thị ra màn hình

```
for (box, pred) in zip(locs, preds):
 # unpack the bounding box and predictions
  (startX, startY, endX, endY) = box
  (mask, withoutMask) = pred

```

```
    # determine the class label and color we'll use to draw
    # the bounding box and text
    label = "Mask" if mask > withoutMask else "No Mask"
    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

    # include the probability in the label
    label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

    # display the label and bounding box rectangle on the output frame
    cv2.putText(frame, label, (startX, startY - 10),
          cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
    cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

# show the output frame
cv2.imshow("Frame", frame)
```

4 Đánh giá và kết quả

4.1 Phương pháp đánh giá hiệu suất của mô hình

Để đánh giá hiệu suất của mô hình ta sẽ sử dụng:

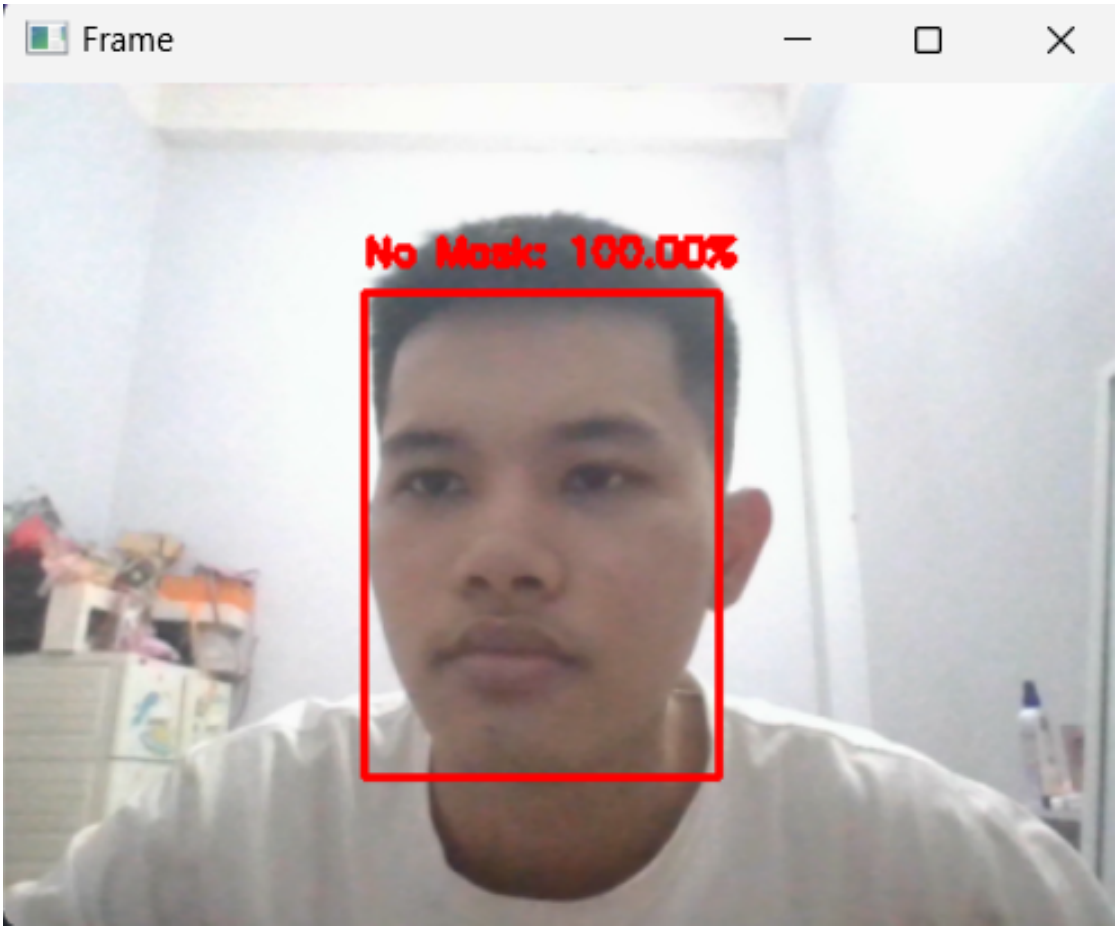
Accuracy (Độ chính xác), là tỷ lệ giữa số lượng dự đoán đúng và tổng số lượng mẫu trong tập kiểm tra, đây là phép đo phổ biến nhất cho các bài toán phân loại.

Loss (Hàm mất mát): Đánh giá mức độ mất mát của mô hình trên tập kiểm tra, được sử dụng trong quá trình huấn luyện để điều chỉnh các tham số của mô hình

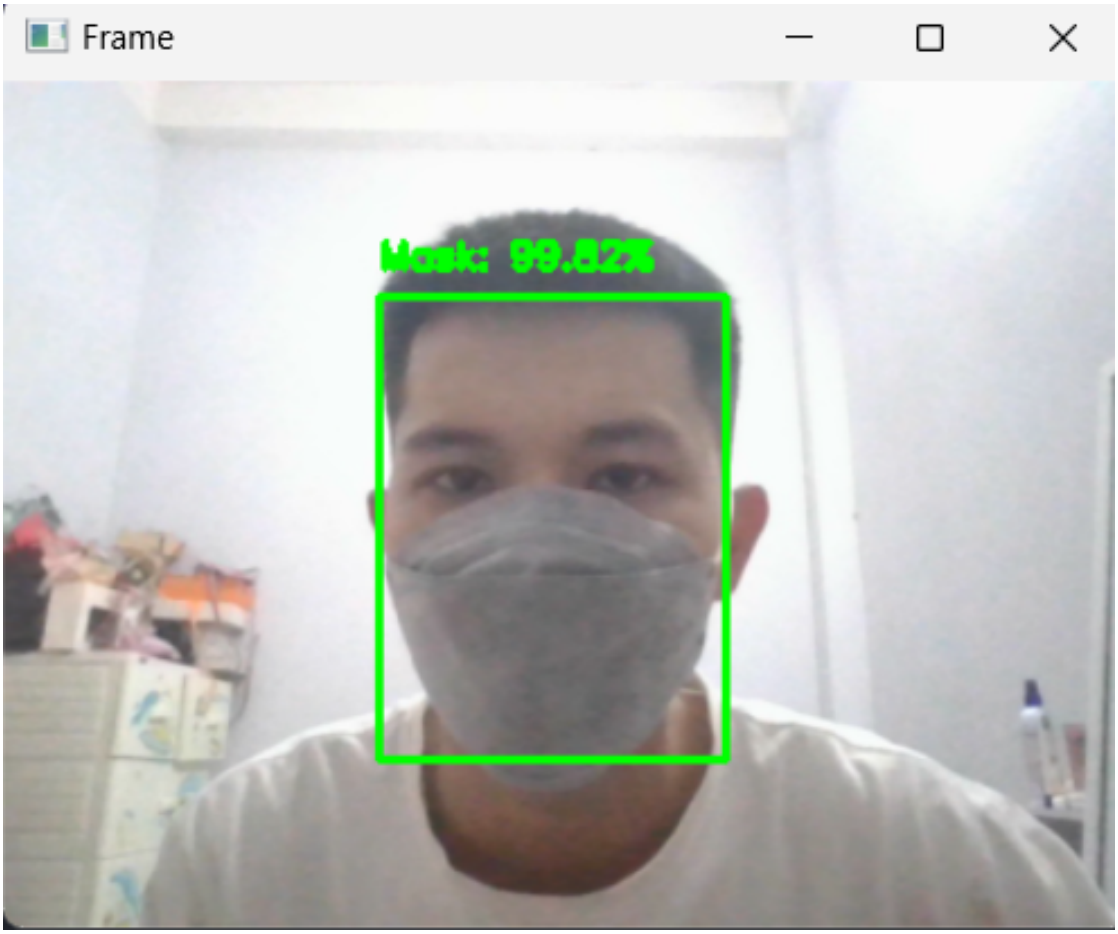
4.2 Kết quả đạt được và đánh giá hiệu suất của ứng dụng

Demo kết quả của ứng dụng:

Hình 13



Hình 14



Biểu đồ accuracy và loss trên epoch:

Hình 15

Ta có thể thấy train accuracy và validate accuracy đều rất cao, chứng mình đây là một mô hình tốt.

5 Nhìn nhận và hướng phát triển

5.1 Ưu điểm và hạn chế của phương pháp

Ưu điểm:

- Phương pháp có hiệu suất tốt đối với bài toán thị giác máy tính
- Nguồn tài liệu để tham khảo lớn, dễ tiếp cận và công cụ hỗ trợ (như TensorFlow, PyTorch) cho việc nghiên cứu và triển khai CNNs.
- Tìm năng rất lớn khi nghiên cứu sâu.

Nhược điểm:

- CNNs thường yêu cầu lượng dữ liệu lớn để huấn luyện một cách hiệu quả
- Việc hiểu và triển khai các mô hình CNNs có thể đòi hỏi kiến thức toán học và lập trình sâu, đặc biệt là đối với các kiến trúc sâu như ResNet, Inception, và Transformer.
- Các mô hình CNNs thường là black box, điều này làm cho việc diễn giải và hiểu cách mà mô hình ra quyết định trở nên khó khăn

5.2 Các cải tiến và hướng phát triển trong tương lai của ứng dụng

Ứng dụng đang còn mở mức rất sơ khai, cần nhiều sự điều chỉnh để có thể áp dụng vào các công việc trong thực tế, để có thể đạt được điều này cần:

Phát triển thêm giao diện để người dùng có thể dễ dàng thao tác các hành động như: chuyển đổi qua lại các camera khác nhau, điều chỉnh các thông số phù hợp, ...

Ngoài việc phát hiện xem người đó có đeo khẩu trang hay không, có thể mở rộng tính năng để phát hiện các vấn đề khác liên quan đến việc đeo khẩu trang, chẳng hạn như việc đeo khẩu trang sai cách, khẩu trang bẩn, hoặc việc đeo khẩu trang trong môi trường nhiễm bẩn.

Phát triển tính năng quản lý dữ liệu để theo dõi và phân tích số liệu về việc đeo khẩu trang, giúp cung cấp thông tin hữu ích cho quản lý an toàn và sức khỏe cộng đồng.

Xây dựng ứng dụng di động hoặc ứng dụng web để người dùng có thể sử dụng trên các thiết bị di động, giúp tiện lợi hơn trong việc kiểm tra và đảm bảo tuân thủ quy định về đeo khẩu trang.

6 Kết luận

Mạng nơ-ron tích chập(CNN) là một kỹ thuật hiệu quả trong việc xử lý nhận diện hình ảnh, cụ thể trong đề tài này là nhận diện khuôn mặt đeo khẩu trang sẽ hưu ích trong việc đảm bảo an toàn và sức khỏe cộng đồng. Mạng CNN tuy có những hạn chế về độ khó, phức tạp, yêu cầu một lượng lớn dữ liệu nhưng nếu ta nghiên cứu sâu thì sẽ mở ra một tiềm năng rất lớn.

Tài liệu

[1] **https://github.com/balajisrinivas/Face-Mask-Detection**

[2] **https://topdev.vn/blog/opencv-la-gi-hoc-computer-vision-khong-kho/**

[3] **https://datagen.tech/guides/face-recognition/face-detection-with-opencv-2-quick-tutorials/**

[4] **https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2**

[5] **https://stanford.edu/ shervine/l/vi/teaching/cs-230/cheatsheet-convolutional-neural-networks**

[6] **https://phamdinhkhanh.github.io/2020/05/31/CNNHistory.html**