TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN
MÔN HỌC MÁY

ĐỀ TÀI: PHÂN TÍCH SỰ CHUYỂN ĐỘNG VÀ TƯƠNG TÁC TRÊN KHUÔN MẶT ỨNG DỤNG THUẬT TOÁN KNN

Giảng viên hướng dẫn: Thầy Trần Anh Đạt

Sinh viên thực hiện: Hà Hải Việt - 2151061211

Nguyễn Việt Anh - 2151060233

Trần Quý Đức - 2151060303

Hà Nội, tháng 10 năm 2023

Mục lục

Lời mở đầu

Với sự phát triển của công nghệ đang diễn ra hàng ngày trong cuộc sống hiện tại, chúng ta không thể phủ nhận được sự quan trọng và những hiệu quả mà máy móc đã đem lại cho chúng ta. Dù trong bất cứ lĩnh vực nào, sự góp mặt của trí tuệ nhân tạo không chỉ giúp chúng ta hoàn thành công việc một cách dễ dàng mà còn đạt được những hiệu suất vượt bậc. Chính vì vậy kiến thức về "Machine Learning" đang được nhiều người tìm hiểu và ứng dụng trong cuộc sống.

Đây là một lĩnh vực khoa học tuy không mới, nhưng nó đã cho chúng ta thấy được những sự ảnh hưởng rõ rệt về việc trí tuệ nhân tạo đang ngày càng phát triển và có thể tiến xa hơn trong tương lai. Chính vì vậy, có thể xem rằng thời điểm này Machine Learning là một lĩnh vực "nóng" và nên dành thời gian quan tâm để phát triển nó một cách mạnh mẽ, bùng nổ hơn.

Thay vì phải code những phần mềm theo cách thức thủ công theo những yêu cầu cụ thể nhằm hoàn thiện một nhiệm vụ đề ra, ta có thể ứng dụng kiến thức về Machine Learning nhằm huấn luyện cho máy tự "học hỏi" thông qua việc sử dụng một lượng lớn dữ liệu cùng những thuật toán cho phép nó thực hiện các tác vụ đã được yêu cầu. Quá trình ứng dụng Machine Learning trong xử lý các tác vụ giúp cho quá trình thực hiện trở nên nhanh chóng và hiệu quả hơn rất nhiều lần do quá trình đó máy sẽ thực hiện tự động, tạo ra những mô hình cho phép phân tích dữ liệu trên quy mô lớn và phức tạp, đồng thời đưa ra

những kết quả cho tác vụ một cách nhanh chóng và có độ chính xác cao.

Chính sự hiệu quả trong công việc và học tập, cũng như những lợi ích mà nó đem lại cho chúng ta khiến cho Machine Learning ngày càng được chú trọng và quan tâm nhiều hơn. Chính vì vậy, nhóm chúng em đã quyết định lựa chọn đề tài: “Phân tích chuyển động và tương tác trên khuôn mặt ứng dụng thuật toán KNN” nhằm học tập cũng như ứng dụng kiến thức về Machine Learning.

Chúng em xin chân thành gửi lời cảm ơn tới các thầy cô giáo trong Khoa Công nghệ thông tin nói chung đã luôn tận tình giảng dạy, phân tích và truyền đạt cho chúng em những kiến thức cũng như kinh nghiệm quý báu trong suốt quá trình học tập và giảng dạy. Và nhóm em xin gửi lời cảm ơn tới thầy Trần Anh Đạt đã luôn tận tâm góp ý, chỉ dạy và hướng dẫn trong suốt quá trình nghiên cứu và học hỏi của chúng em.

CHƯƠNG I: GIỚI THIỆU VỀ HỌC MÁY VÀ THUẬT TOÁN KNN

1. Giới thiệu về học máy

1.1. Khái niệm về học máy

Học máy (Machine learning) là một lĩnh vực con của Trí tuệ nhân tạo (Artificial Intelligence) sử dụng các thuật toán cho phép máy tính có thể học hỏi từ các bộ dữ liệu để thực hiện các công việc thay vì được lập trình một cách rõ ràng, cung cấp cho hệ thống khả năng tự động học hỏi và cải thiện hiệu suất, độ chính xác dựa trên những kinh nghiệm từ dữ liệu đầu vào. Học máy tập trung vào việc phát triển các phần mềm, chương trình máy tính có thể truy cập vào dữ liệu và tận dụng nguồn dữ liệu đó để tự học.

Tuy nhiên, học máy vẫn đòi hỏi sự đánh giá của con người trong việc tìm hiểu dữ liệu cơ sở và lựa chọn các kĩ thuật phù hợp để phân tích dữ liệu. Đồng thời, trước khi sử dụng, cần yêu cầu rà soát dữ liệu phải chính xác và không có sai lệch. Các mô hình học máy yêu cầu lượng dữ liệu đủ lớn để "huấn luyện" và đánh giá mô hình. Trước đây, các thuật toán học máy thiếu quyền truy cập vào một lượng lớn dữ liệu cần thiết để mô hình hóa các mối quan hệ giữa các dữ liệu. Sự tăng trưởng trong dữ liệu lớn (big data) đã cung cấp các thuật toán học máy với đủ dữ liệu để cải thiện độ chính xác của mô hình và dự đoán.

1.2. Ứng dụng của học máy

Nhiều hoạt động hàng ngày của chúng ta được trợ giúp bởi các thuật toán Machine Learning, bao gồm:

- Trong y tế: xác định bệnh lý của người bệnh mới dựa trên dữ liệu lịch sử của các bệnh nhân có cùng bệnh lý có cùng các đặc điểm đã được chữa khỏi trước đây, hay xác định được các loại thuốc phù hợp.
- Trong lĩnh vực ngân hàng: xác định khả năng khách hàng chậm trả các khoản vay hoặc rủi ro tín dụng do nợ xấu dựa trên phân tích Credit score; xác định xem liệu các giao dịch có hành vi phạm tội, lừa đảo hay không.
- Trong giáo dục: phân loại các học sinh theo hoàn cảnh, học lực để sắp xếp những hỗ trợ cho những học sinh có hoàn cảnh sống khó khăn nhưng có học lực tốt.
- Trong thương mại điện tử: phân loại khách hàng theo sở thích cụ thể nhằm hỗ trợ personalized marketing hay xây dựng hệ thống khuyến nghị dựa trên dữ liệu từ các website, social media.

2. Thuật toán K-Nearest Neighbors trong học máy

2.1. Khái niệm về thuật toán

Thuật toán KNN (K-Nearest Neighbors) là một thuật toán học máy có giám sát, đơn giản và dễ triển khai. Thuật toán này có thể được sử dụng trong các bài toán phân loại và hồi quy. Thuật toán này dựa trên ý tưởng rằng các điểm dữ liệu tương tự nhau sẽ có xu hướng nằm gần nhau trong không gian, từ đó tìm K điểm gần với điểm dữ liệu cần kiểm tra nhất.

2.2. Ý tưởng của thuật toán:

Thuật toán KNN cho rằng những dữ liệu tương tự nhau sẽ tồn tại **gần nhau** trong không gian, từ đó công việc của chúng ta sẽ là tìm K điểm gần với dữ liệu cần kiểm tra nhất. Việc tìm khoảng cách giữa 2 điểm có

thể sử dụng nhiều công thức, mỗi loại sẽ có các ưu và nhược điểm riêng trong quá trình sử dụng. Thuật toán KNN có thể được miêu tả qua các bước sau:

- **Bước 1: Xác định tham số K:**
  Tham số K là số lượng hàng xóm gần nhất mà thuật toán sẽ sử dụng để đưa ra dự đoán.
- **Bước 2: Tính khoảng cách giữa điểm dữ liệu mới và các điểm dữ liệu trong tập huấn luyện:**
  Có nhiều cách khác nhau để tính toán khoảng cách giữa hai điểm dữ liệu. Các phương pháp tính khoảng cách phổ biến bao gồm:
  - ❖ **Khoảng cách Euclidean:** Là phương pháp phổ biến nhất và thường được sử dụng trong thuật toán KNN.

    Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

    - Ưu điểm:
      - Đơn giản, dễ hiểu và dễ triển khai và ứng dụng.
      - Có thể áp dụng cho nhiều loại dữ liệu.
    - Nhược điểm:
      - Có thể bị ảnh hưởng bởi các dữ liệu nhiễu.
  - ❖ **Khoảng cách Manhattan:** Được tính bằng tổng trị tuyệt đối của hiệu giữa các chiều của hai điểm dữ liệu.

    Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

    - Ưu điểm:

- Không bị ảnh hưởng bởi các dữ liệu nhiễu.

- Nhược điểm:
  - Có thể không phản ánh chính xác mối quan hệ giữa các điểm dữ liệu.
  - Có thể dẫn đến ước tính sai dữ liệu.

❖ **Khoảng cách Minkowski:** Là một phương pháp tổng quát hóa, cho phép điều chỉnh tham số P để điều chỉnh mức độ nhạy cảm của khoảng cách đối với các biến. Khi đặt P=1, thuật toán sử dụng công thức tính khoảng cách Manhattan; khi đặt P=2, thuật toán sử dụng công thức tính khoảng cách Euclidean; khi P>2, thuật toán sử dụng công thức tính khoảng cách Minkowski có sử dụng trọng số.

Minkowski $$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

- Ưu điểm:
  - Có thể điều chỉnh để phù hợp với các loại dữ liệu khác nhau.
  - Có thể phản ánh chính xác mối quan hệ giữa các điểm dữ liệu.
  - Tránh được sự ảnh hưởng của các dữ liệu nhiễu.
- Nhược điểm:
  - Phức tạp hơn các phương pháp khác.
  - Khó có thể chọn được giá trị P phù hợp.

- **Bước 3: Phân loại hoặc Hồi quy điểm dữ liệu mới dựa trên các điểm dữ liệu hàng xóm:**

  - Trong bài toán Phân loại, thuật toán KNN sẽ phân loại điểm dữ liệu mới dựa trên lớp của đa số các hàng xóm gần nhất.

  - Trong bài toán Hồi quy, thuật toán KNN sẽ phân loại điểm dữ liệu mới dựa trên giá trị trung bình của các hàng xóm gần nhất.

2.3. Ưu điểm của thuật toán KNN trong ứng dụng:

- Đơn giản và dễ hiểu.

- Không yêu cầu tiền xử lý dữ liệu phức tạp.

- Có thể áp dụng cho nhiều loại dữ liệu.

2.4. Nhược điểm của thuật toán KNN trong ứng dụng:

- Dễ bị ảnh hưởng bởi các dữ liệu nhiễu.

- Yêu cầu nhiều bộ nhớ để có thể lưu trữ được toàn bộ tập dữ liệu huấn luyện.

- Quá trình khởi chạy có thể chậm trong trường hợp tập dữ liệu quá lớn.

2.5. Những ứng dụng của thuật toán KNN:

Thuật toán K-Nearest Neighbors có thể được sử dụng trong nhiều lĩnh vực khác nhau, có thể kể đến như:

- Phân loại hình ảnh.

- Nhận dạng chữ viết.

- Phân loại văn bản.

- Phân loại đối tượng.

- Phân loại dữ liệu.

2.6. Dạng của thuật toán KNN:

Thuật toán K-Nearest Neighbor có thể được biểu diễn dưới dạng một hàm như sau:

$$\mathbf{y = f(x) = argmax(y_K)}$$

Trong đó:

- y là nhãn của điểm dữ liệu mới
- x là điểm dữ liệu mới
- K là số lượng điểm hàng xóm gần nhất
- y_K là nhãn của điểm hàng xóm K

2.7. Hàm mất mát của thuật toán KNN:

Thuật toán K-Nearest Neighbors (KNN) không sử dụng một hàm mất mát (loss function) cụ thể như các thuật toán học máy khác như hồi quy tuyến tính hoặc phân loại SVM. Thay vào đó, KNN dựa trên một cách tiếp cận dựa trên sự tương đồng (similarity) giữa các điểm dữ liệu.

Cụ thể, khi bạn sử dụng KNN để thực hiện phân loại (classification), quá trình hoạt động của nó như sau:

- Xác định số lượng láng giềng gần nhất (K) mà bạn muốn sử dụng để đưa ra quyết định cho điểm dữ liệu đang xét.
- Tìm K láng giềng gần nhất dựa trên độ đo sự tương đồng. Độ đo này thường là khoảng cách Euclidean hoặc các độ đo khác phụ thuộc vào bài toán cụ thể.
- Đếm số lượng láng giềng trong mỗi lớp (hoặc nhãn) của K láng giềng gần nhất.

- Gán nhãn cho điểm dữ liệu đang xét bằng nhãn có số lượng láng giềng nhiều nhất trong K láng giềng.

Không có hàm mất mát (loss function) chính thức trong KNN, bởi vì nó không tối ưu hóa các tham số hay mô hình như các thuật toán học máy khác. Thay vào đó, KNN đơn giản là so sánh sự tương đồng giữa điểm dữ liệu cần phân loại và các điểm dữ liệu trong tập huấn luyện để thực hiện phân loại.

2.8. Mức độ lỗi của thuật toán KNN:

Mức độ lỗi của thuật toán K-Nearest Neighbors (KNN) không được đo lường thông qua một hàm mất mát (loss function) như các thuật toán học máy khác. Thay vào đó, KNN thường sử dụng các phép đo đánh giá hiệu suất phân loại hoặc dự đoán.

Mức độ lỗi của thuật toán K-Nearest Neighbors (KNN) phụ thuộc vào nhiều yếu tố quan trọng sau đây:

- **Giá trị của K:** Giá trị K là số lượng láng giềng gần nhất mà bạn chọn để quyết định lớp của một điểm dữ liệu mới. Lựa chọn sai giá trị K có thể ảnh hưởng đến hiệu suất của thuật toán. Nếu K quá nhỏ, KNN có thể bị ảnh hưởng bởi nhiễu và làm tăng độ phức tạp của mô hình. Nếu K quá lớn, nó có thể làm mất đi thông tin quan trọng trong dữ liệu. Do đó, cần phải thử nghiệm và điều chỉnh giá trị K phù hợp với bài toán cụ thể.
- **Độ đo tương đồng (distance metric):** Lựa chọn độ đo tương đồng quyết định cách KNN tính toán khoảng cách giữa các điểm dữ liệu. Sử dụng độ đo không phù hợp có thể dẫn đến

kết quả sai lệch. Chọn một độ đo tương đồng phù hợp với tính chất của dữ liệu là quan trọng.

- **Xử lý dữ liệu nhiễu:** Dữ liệu nhiễu có thể ảnh hưởng đến hiệu suất của KNN. Các điểm dữ liệu nhiễu có thể tạo ra sự nhiễu trong việc tính khoảng cách và dẫn đến dự đoán sai. Cần xem xét các kỹ thuật xử lý dữ liệu nhiễu như loại bỏ điểm nhiễu hoặc sử dụng kỹ thuật smoothing để làm mịn dữ liệu.
- **Mất cân bằng lớp (class imbalance):** Khi số lượng điểm dữ liệu thuộc một lớp nhiều hơn so với lớp khác, KNN có thể trở nên chưa chính xác. Cần sử dụng các kỹ thuật xử lý mất cân bằng lớp như chọn lại mẫu (oversampling) hoặc giảm mẫu (undersampling) để cân bằng dữ liệu.
- **Tiền xử lý dữ liệu:** Việc tiền xử lý dữ liệu, chẳng hạn như chuẩn hóa dữ liệu hoặc giảm chiều dữ liệu, có thể ảnh hưởng đến hiệu suất của KNN.
- **Tính chất của dữ liệu:** Mức độ lỗi của KNN còn phụ thuộc vào tính chất của dữ liệu trong tập huấn luyện và tập kiểm tra. KNN hoạt động tốt trong các bài toán có tính chất không gian tương tự giữa các lớp.

CHƯƠNG II: PHÂN TÍCH YÊU CẦU ĐỀ TÀI

1. Giới thiệu về đề tài

1.1. Ý tưởng của đề tài

Sau quá trình học tập và nghiên cứu, nhóm chúng em nhận thấy rằng những kiến thức đã học có thể ứng dụng trong việc phân tích sự chuyển động và tương tác. Cùng với các kiến thức về sức khỏe, chúng em nhận thấy rằng việc chạm tay lên mặt có thể dẫn đến việc lây lan các mầm bệnh truyền nhiễm như các bệnh về da liễu, herpes,... Vì vậy, nhóm chúng em đã lên ý tưởng về một chương trình phân tích sự tương tác trên khuôn mặt ứng dụng những kiến thức về thuật toán KNN.

1.2. Phương pháp nghiên cứu

1.2.1. Thu thập dữ liệu

Nhóm chúng em đã lên ý tưởng về cách thu thập dữ liệu qua việc truy cập vào webcam của thiết bị sử dụng, thay vì sử dụng các dữ liệu có sẵn được truyền vào chương trình. Điều này có thể giảm tải được bộ nhớ mà chương trình phải lưu trữ bằng cách chuyển đổi hình ảnh thu được từ webcam thành một đối tượng được lưu trữ, bao gồm 2 thành phần là "hình ảnh" và "giá trị", sau đó lưu vào bộ nhớ của thư viện dưới dạng bộ nhớ cache. Việc làm này có thể dẫn tới việc khi ta tắt chương trình và khởi động lại, ta phải huấn luyện lại hệ thống từ đầu.

1.2.2. Tiền xử lý dữ liệu

Quá trình tiền xử lý dữ liệu được thực hiện với các yêu cầu sau:

- Tải dữ liệu video từ webcam

- Chuẩn hóa dữ liệu từ video bằng cách chuyển đổi dữ liệu về cùng một thang đo
- Trích xuất các đặc trưng từ dữ liệu video bằng cách sử dụng mô hình MobileNet
- Xây dựng mô hình KNN để phân loại và đánh giá các đặc trưng của dữ liệu

Quá trình này được xử lý bằng cách sử dụng các thư viện:

- **@tensorflow/tfjs:** Thư viện này cung cấp các hàm và phương thức để thực hiện các thao tác xử lý dữ liệu cơ bản, chẳng hạn như tải dữ liệu, chuẩn hóa dữ liệu và xây dựng mô hình.
- **@tensorflow-models/mobilenet:** Thư viện này cung cấp một mô hình học sâu đã được huấn luyện để phân loại hình ảnh. Mô hình này có thể được sử dụng để trích xuất các đặc trưng từ hình ảnh khuôn mặt.
- **@tensorflow-models/knn-classifier:** Thư viện này cung cấp một lớp phân loại k-nearest neighbors (KNN). Lớp phân loại này có thể được sử dụng để phân loại các dữ liệu dựa trên các đặc trưng của dữ liệu.
- Chương trình ứng dụng các thư viện này qua các câu lệnh import sau:

```
import * as knnClassifier from "@tensorflow-models/knn-classifier";
import * as mobilenetModule from "@tensorflow-models/mobilenet";
import * as tf from "@tensorflow/tfjs";
```

1.2.3. Xây dựng mô hình KNN

1.2.3.1. Chuẩn bị dữ liệu

Quá trình xây dựng mô hình KNN bắt đầu bằng việc chuẩn bị tập dữ liệu huấn luyện. Bộ dữ liệu được sử dụng để huấn luyện mô hình là các hình ảnh khuôn mặt của người dùng, lưu trữ trong bộ nhớ của thư viện và các hình ảnh này được chia thành hai lớp chính bao gồm:

- Lớp **"not_touch"**: Các hình ảnh trong đó người dùng không chạm tay lên mặt
- Lớp **"touch"**: Các hình ảnh trong đó người dùng đã chạm tay lên mặt

Những dữ liệu sẽ được thu lại thông qua hình ảnh mà webcam thiết bị thu nhận được, sau đó lưu trữ trong thư viện đã được import dưới dạng bộ nhớ cache của trình duyệt.

1.2.3.2. Huấn luyện mô hình

Mô hình KNN được xây dựng bằng cách sử dụng dữ liệu đã được chuẩn bị từ tập huấn luyện. Quá trình này được thực hiện thông qua hàm **train()** với quá trình chi tiết như sau:

1.2.3.2.1. Thêm dữ liệu vào bộ phân loại KNN

Các dữ liệu từ tập dữ liệu huấn luyện sẽ được thêm vào trong quá trình training thông qua việc gọi hàm **"training(label)"** trong quá trình huấn luyện. Quá trình này được thực hiện qua việc trích xuất các đặc trưng (embedding) từ mô hình MobileNet cho hình ảnh hiện tại từ webcam của người dùng. Sau đó thêm ví dụ này cùng với nhãn **(label)** tương ứng vào bộ phân loại KNN. Mỗi

ví dụ sẽ được thêm vào sau một khoảng thời gian chờ để đảm bảo quá trình đào tạo được thực hiện một cách chính xác nhất.

1.2.3.2.2. Lặp lại quá trình đào tạo

Quá trình huấn luyện sẽ được lặp lại liên tục trong vòng lặp với số lần cố định, có thể điều chỉnh bằng việc thay đổi giá trị **TRAINING_TIME** trong hệ thống. Trong mỗi lần lặp, một ví dụ mới được thêm vào bộ phân loại KNN và tiến trình huấn luyện sẽ được cập nhật qua thanh tiến trình nhằm hỗ trợ trong việc theo dõi và kiểm soát.

1.2.3.2.3. Xác định nhãn đích

Trước khi quá trình huấn luyện diễn ra, cần xác định xem chương trình đang huấn luyện lớp **TOUCH_LABEL** (chạm tay lên mặt) hay **NOT_TOUCH_LABEL** (không chạm tay lên mặt). Trong trường hợp chương trình xác nhận hình ảnh thu nhận thuộc lớp **TOUCH_LABEL,** sẽ có thông báo hiện lên (không ảnh hưởng đến mô hình KNN) và được in ra trong bảng console.

1.2.3.2.4. Kết thúc quá trình huấn luyện

Khi quá trình huấn luyện hoàn thành, một thông báo sẽ được đẩy lên bảng console, làm đầy thanh tiến trình và kết thúc quá trình huấn luyện.

1.2.4. Phân tích quá trình hoạt động

Sau khi quá trình xây dựng mô hình KNN đã được hoàn thành, chương trình giờ đây có khả năng dự đoán và phân loại nhãn cho các

dữ liệu mới. Quá trình này sẽ được thực hiện thông qua hàm **run()** với hoạt động chi tiết như sau:

1.2.4.1. Trích xuất dữ liệu từ video

Trong chương trình này, quá trình trích xuất dữ liệu từ video thông qua việc sử dụng mô hình Mobilenet để trích xuất đặc trưng (embedding) từ hình ảnh hiện tại từ video của máy ảnh người dùng.

1.2.4.2. Dự đoán nhãn cho dữ liệu

Hàm **"classifier.predictClass(embedding)"** được sử dụng để dự đoán nhãn của dữ liệu mới dựa trên các ví dụ đã được huấn luyện. Nếu mô hình dự đoán rằng tay đã chạm lên mặt (phù hợp với **TOUCH_LABEL**) và độ tự tin của quá trình dự đoán lớn hơn ngưỡng **"TOUCH_CONFIDENCE"** đã được thiết lập, giá trị của biến isTouch được thiết lập thành true và ngược lại với trường hợp **NOT_TOUCH_LABEL.**

1.2.4.3. Phát âm thanh và hiển thị thông báo

Nếu giá trị của biến isTouch được xác định là true, tương ứng với việc chương trình nhận diện rằng người dùng đang chạm tay lên mặt, mô hình sẽ phát âm thanh báo hiệu và hiển thị hình ảnh thông báo trên giao diện người dùng.

1.2.4.4. Lặp lại quá trình dự đoán

Quá trình dự đoán được lặp lại liên tục trong quá trình hoạt động. Điều này giúp ứng dụng tiếp tục theo dõi và phản ứng khi có sự thay đổi trong hình ảnh từ máy ảnh.

2. Xây dựng chương trình

2.1. Hình thức xây dựng

Quá trình xây dựng chương trình được thực hiện trên phần mềm Visual Studio Code với việc sử dụng ngôn ngữ lập trình JavaScript. Việc này sẽ có các ưu và khuyết điểm so với các ngôn ngữ khác như sau:

2.1.1. Ưu điểm:

- **Trình duyệt hỗ trợ tốt:** JavaScript là một ngôn ngữ được hỗ trợ mạnh mẽ bởi hầu hết các trình duyệt web hiện đại. Điều này cho phép quá trình triển khai mô hình học máy có thể thực hiện trực tiếp trên trình duyệt, giúp trải nghiệm người dùng dễ dàng hơn mà không cần cài đặt thêm phần mềm.
- **Phát triển nhanh chóng:** JavaScript là một ngôn ngữ dễ học và sử dụng, có cộng đồng lớn và nhiều tài liệu và thư viện hữu ích. Điều này giúp tăng tốc quá trình phát triển và thử nghiệm các mô hình học máy.
- **Khả năng tích hợp cao:** JavaScript có khả năng tích hợp tốt với các phần khác của ứng dụng web, chẳng hạn như giao diện người dùng và tương tác người dùng. Điều này giúp dễ dàng tích hợp học máy vào ứng dụng web tổng thể.
- **Đa dạng các thư viện phục vụ học máy:** JavaScript có các thư viện và khung làm việc cho học máy như TensorFlow.js, Brain.js, và Synaptic.js. Những thư viện này giúp triển khai các mô hình học máy một cách tiện lợi.

2.1.2. Nhược điểm:

- **Hiệu suất giới hạn:** JS thường không hiệu quả bằng các ngôn ngữ như Python hoặc C++ trong việc xử lý tính toán số học nặng và đối tượng học máy lớn. Điều này có thể dẫn đến thời gian đáng kể trong việc huấn luyện và dự đoán cho các mô hình phức tạp.
- **Hạn chế trong việc xử lý dữ liệu lớn:** JavaScript có khả năng xử lý dữ liệu lớn hạn chế hơn so với các ngôn ngữ phát triển đặc biệt cho học máy như Python với thư viện NumPy và Pandas.
- **Thư viện học máy hạn chế:** Mặc dù có một số thư viện học máy cho JS, nhưng chúng không phong phú và đa dạng như thư viện cho các ngôn ngữ khác như Python. Điều này có thể gây khó khăn trong việc triển khai các mô hình học máy phức tạp.
- **Hiện tượng "lock-in":** Một ứng dụng xây dựng bằng JavaScript có thể bị ràng buộc vào môi trường trình duyệt và ngôn ngữ này, điều này làm cho việc di chuyển sang môi trường khác khó khăn hơn.
- **An ninh và quyền riêng tư:** Trong trình duyệt web, việc xử lý dữ liệu nhạy cảm và học máy có thể tạo ra các vấn đề liên quan đến bảo mật và quyền riêng tư nếu không được thực hiện cẩn thận.

2.2. Các thư viện sử dụng

Chương trình được xây dựng và sử dụng các thư viện hỗ trợ sau:

```
import React, { useRef, useState } from "react";
import "./App.css";
import { Howl } from "howler";
import heySound from "./assets/hey_sondn.mp3";
import * as knnClassifier from "@tensorflow-models/knn-classifier";
import * as mobilenetModule from "@tensorflow-models/mobilenet";
import * as tf from "@tensorflow/tfjs";
import * as posenet from "@tensorflow-models/posenet";
```

- **“import { Howl } from "howler":** Howler là một thư viện âm thanh JavaScript cho phép bạn sử dụng và phát các tệp âm thanh trong chương trình.
- **“import * as knnClassifier from "@tensorflow-models/knn-classifier":** Tiếp theo, ta import toàn bộ module knnClassifier từ thư viện @tensorflow-models/knn-classifier. Thư viện này là một phần của TensorFlow.js và cung cấp một cài đặt của mô hình K-Nearest Neighbors (KNN) để phân loại dữ liệu.
- **“import * as mobilenetModule from "@tensorflow-models/mobilenet":** Sau đó, ta import toàn bộ module mobilenetModule từ thư viện @tensorflow-models/mobilenet. Đây là một mô hình mạng nơ-ron nhỏ gọn được sử dụng để trích xuất đặc trưng hình ảnh từ webcam của người dùng.
- **“import * as tf from "@tensorflow/tfjs":** Đoạn mã này import toàn bộ module từ thư viện @tensorflow/tfjs. TensorFlow.js là một thư viện học máy mã nguồn mở cho

JavaScript, cho phép bạn xây dựng và huấn luyện các mô hình học máy trực tiếp trong trình duyệt.

- **“import * as posenet from @tensorflow-models/posenet”:** Đoạn mã này import toàn bộ module từ thư viện @tensorflow-models/posenet. Posenet là thư viện hỗ trợ trong việc xác định vị trí các điểm ảnh trong mô hình thuật toán KNN trong học máy

2.3. Khai báo biến và gán giá trị:

Ta khai báo các biến và giá trị cho chương trình như sau:

```
const NOT_TOUCH_LABEL = "not_touch";
const TOUCH_LABEL = "touch";
const TRAINING_TIME = 200;
const TOUCH_CONFIDENCE = 0.8;
const ERROR_MESSAGE = "Đừng chạm tay lên mặt!";
let dataset = [];
```

- **const NOT_TOUCH_LABEL = "not_touch":** Đây là một hằng số được đặt tên là NOT_TOUCH_LABEL và nó lưu trữ chuỗi "not_touch". Được sử dụng để định danh lớp dữ liệu khi người dùng không chạm tay lên mặt.
- **const TOUCH_LABEL = "touch":** Tương tự, đây là một hằng số khác được đặt tên là TOUCH_LABEL và nó lưu trữ chuỗi "touch". Được sử dụng để định danh lớp dữ liệu khi người dùng chạm tay lên mặt.
- **const TRAINING_TIME = 200:** Hằng số TRAINING_TIME lưu trữ giá trị 200. Được sử dụng để xác định số lượng lần lặp trong quá trình huấn luyện mô hình KNN. Trong trường hợp này, mô hình sẽ được huấn luyện 200 lần.

- **const TOUCH_CONFIDENCE = 0.8:** Hằng số TOUCH_CONFIDENCE lưu trữ giá trị 0.8. Được sử dụng để xác định ngưỡng độ tin cậy khi dự đoán liệu có chạm tay lên mặt hay không. Nếu độ tin cậy (confidence) của dự đoán vượt quá 0.8, thì mô hình sẽ xem xét đó là chạm tay lên mặt.
- **const ERROR_MESSAGE = "Đừng chạm tay lên mặt!":** Hằng số ERROR_MESSAGE lưu trữ một chuỗi thông báo lỗi. Được sử dụng để hiển thị thông báo cho người dùng khi mô hình dự đoán rằng họ đang chạm tay lên mặt.
- **let dataset = []:** Biến dataset là một mảng rỗng được khai báo bên ngoài các hàm và thành phần khác trong ứng dụng. Dự kiến, biến này sẽ được sử dụng để lưu trữ dữ liệu hình ảnh và nhãn tương ứng của người dùng để huấn luyện mô hình KNN. Mỗi phần tử trong mảng dataset là một đối tượng chứa hình ảnh và nhãn.

2.4. Khởi tạo biến tham chiếu (refs):

```
const video = useRef();
const mobilenet = useRef();
const net = useRef();
const progress = useRef();
const classifier = knnClassifier.create();
const [isTouch, setIsTouch] = useState(false);
```

- **video, mobilenet, progress, net:** Đây là các biến tham chiếu sử dụng hook useRef để tham chiếu đến các phần tử trong DOM, cụ thể là một thẻ video (<video>) để hiển thị dữ liệu video từ webcam, một tham chiếu đến mô hình MobileNet, tham chiếu tới thư viện phân

tích điểm ảnh Posenet và một tham chiếu đến thanh tiến trình <progress>.

- **classifier:** Đây là biến sử dụng để tạo một mô hình phân loại K-Nearest Neighbors (KNN) bằng cách gọi hàm **knnClassifier.create()**. Mô hình này sẽ được sử dụng để huấn luyện và dự đoán liệu người dùng có đang chạm tay vào mặt hay không.
- **isTouch:** Đây là một biến state sử dụng hook useState để theo dõi trạng thái của ứng dụng, cụ thể là trạng thái khi người dùng chạm tay vào mặt hoặc không.

2.5. Xây dựng các hàm cho chương trình:

2.5.1. Hàm sleep

```
const sleep = (time) => new Promise((resolve) => setTimeout(resolve, time));
```

Đây là một hàm promise trả về một hứa hẹn được giải quyết sau một khoảng thời gian time milliseconds. Được sử dụng để tạo một đợi ngắn trong quá trình huấn luyện và sử dụng chương trình, đảm bảo quá trình huấn luyện và hoạt động cho mô hình được diễn ra với độ chính xác tối ưu nhất.

2.5.2. Hàm training

```
const training = (label) => {
    return new Promise(async (resolve) => {
        const embedding = mobilenet.current.infer(video.current, true);
        classifier.addExample(embedding, label);
        dataset = [...dataset, { image: video.current, label }];
        await sleep(100);
        resolve();
    });
};
```

Hàm **training** thực hiện tác vụ thêm dữ liệu huấn luyện cụ thể vào mô hình KNN và cập nhật dữ liệu. Nó sử dụng mô hình MobileNet để trích xuất đặc trưng từ hình ảnh hiện tại của video (video.current) và sau đó gọi

classifier.addExample(embedding, label) để thêm ví dụ huấn luyện vào mô hình. Ngoài ra, nó cũng cập nhật dataset bằng cách thêm thông tin về hình ảnh và nhãn tương ứng để sử dụng sau này để tính toán độ chính xác. Hệ thống code có thể được phân tích chi tiết như sau:

- **const training = (label) => { }:** Đây là một hàm arrow function với tham số label. Hàm này nhận một nhãn label để gắn cho dữ liệu huấn luyện.
- **return new Promise(async (resolve) => { }):** Hàm training trả về một promise. Promise này sẽ được giải quyết khi quá trình huấn luyện hoàn thành.
- **const embedding = mobilenet.current.infer(video.current, true):** Dòng này sử dụng mô hình MobileNet (mobilenet) để trích xuất các đặc trưng từ hình ảnh hiện tại của video (được lưu trong video.current). Hàm infer được sử dụng để tính toán các giá trị nhúng (embeddings) từ hình ảnh và trả về chúng dưới dạng một tensor.
- **classifier.addExample(embedding, label):** Dòng này thêm ví dụ vào mô hình KNN (classifier). Nó sử dụng dữ liệu embedding (các đặc trưng) tính toán từ hình ảnh để gắn với nhãn label. Điều này đang xây dựng một bộ dữ liệu huấn luyện cho mô hình KNN.
- **dataset = [...dataset, { image: video.current, label }]:** Dòng này cập nhật mảng dataset bằng cách thêm một đối tượng mới vào mảng này. Đối tượng này chứa hình ảnh (video.current) và nhãn label tương ứng. Mảng dataset được sử dụng để theo dõi toàn bộ các dữ liệu huấn luyện đã được thêm vào mô hình.

- **await sleep(100):** Dòng này sử dụng hàm sleep để tạo một đợi ngắn trong quá trình huấn luyện. Việc này giúp tạo ra một khoảng thời gian ngắn sau mỗi lần thêm dữ liệu huấn luyện để đảm bảo rằng quá trình huấn luyện không quá nhanh và đảm bảo độ chính xác cao hơn.
- **resolve():** Sau khi tất cả các thao tác đã hoàn thành, promise sẽ được giải quyết bằng cách gọi hàm resolve(). Điều này cho biết rằng quá trình huấn luyện đã hoàn thành và promise có thể được giải quyết.

2.5.3. Hàm train

```
const train = async (label) => {
    console.log(`[${label} đang check với gương mặt của bạn]`);
    for (let i = 0; i < TRAINING_TIME; i++) {
        console.log(`progress ${progress.current.value} %`);
        await training(label);
        progress.current.value = ((i + 1) * 100) / TRAINING_TIME;
        // setProgress(((i + 1) / TRAINING_TIME) * 100);
    }
    console.log("máy đã học xong");
};
```

Đây là hàm chính trong quá trình huấn luyện mô hình KNN. Hàm này có mục tiêu chính là thực hiện việc huấn luyện mô hình với dữ liệu được gắn nhãn. Hàm này sử dụng một vòng lặp để lặp qua **TRAINING_TIME** lần và trong mỗi lần lặp, nó gọi hàm training(label) để thêm dữ liệu huấn luyện và cập nhật tiến trình. Sau khi hoàn thành, nó in ra thông báo cho biết máy đã hoàn thành việc huấn luyện. Hệ thống code có thể được phân tích chi tiết như sau:

- **const train = async (label) => { }**: Đây là một hàm arrow function có tham số label. Hàm này sẽ thực hiện quá trình huấn luyện mô hình KNN với dữ liệu được gắn nhãn label.
- **console.log([${label} đang check với gương mặt của bạn]):** Dòng này đơn giản là in ra một thông báo vào console để hiển thị quá trình huấn luyện đang diễn ra với nhãn label tương ứng.
- **for (let i = 0; i < TRAINING_TIME; i++) {** }: Vòng lặp này chạy **TRAINING_TIME** lần (trong trường hợp này là 200 lần) để thực hiện quá trình huấn luyện.
- **console.log(progress ${progress.current.value} %):** Dòng này in ra tiến trình huấn luyện hiện tại trong đơn vị phần trăm. Giá trị tiến trình được lấy từ **progress.current.value**.
- **await training(label):** Dòng này gọi hàm **training** để thực hiện việc thêm dữ liệu huấn luyện cho mô hình KNN. Lưu ý rằng await được sử dụng để đợi cho đến khi hàm training hoàn thành trước khi tiếp tục vòng lặp.
- **progress.current.value = ((i + 1) * 100) / TRAINING_TIME:** Dòng này cập nhật giá trị của tiến trình **(progress.current.value)** bằng cách tính toán phần trăm hoàn thành của quá trình huấn luyện. Điều này cho phép hiển thị tiến trình trực quan cho người dùng.
- **console.log("máy đã học xong"):** Sau khi vòng lặp hoàn thành, dòng này in ra một thông báo cho biết rằng máy đã hoàn thành quá trình huấn luyện với nhãn tương ứng.

2.5.4. Hàm setupCamera

```
const setupCamera = async () => {
    return new Promise((resolve, reject) => {
        navigator.getUserMedia = navigator.getUserMedia || navigator.mozGetUserMedia ||
        navigator.mediaDevices.getUserMedia || navigator.webkitGetUserMedia || navigator.msGetUserMedia;
        if (navigator.getUserMedia) {
            navigator.getUserMedia(
                {
                    video: true,
                },
                (stream) => {
                    video.current.srcObject = stream;
                    video.current.addEventListener("loadeddata", resolve);
                },
                (error) => reject(error)
            );
        } else reject();
    });
};
```

Hàm **setupCamera** có nhiệm vụ thiết lập và kết nối camera của máy tính hoặc thiết bị đang sử dụng để thu thập hình ảnh video cho quá trình huấn luyện và dự đoán của ứng dụng. Khi quá trình này hoàn thành, nó trả về một Promise để thông báo việc hoàn thành hoặc thất bại của việc thiết lập camera. Hoạt động của hàm này có thể được phân tích như sau:

- Hàm này trả về một Promise, có nghĩa là nó sẽ trả về một giá trị sau khi quá trình thiết lập camera hoàn thành.
- Trong hàm này, nó sử dụng API **getUserMedia** của trình duyệt để yêu cầu truy cập vào camera của máy tính. Quá trình này có thể thực hiện thông qua các phương thức **navigator.getUserMedia, navigator.mozGetUserMedia, navigator.mediaDevices.getUserMedia, navigator.webkitGetUserMedia,** hoặc **navigator.msGetUserMedia** tùy thuộc vào trình duyệt.
- Nếu trình duyệt hỗ trợ **getUserMedia**, nó sẽ gọi hàm **navigator.getUserMedia** với một đối tượng cấu hình có

thuộc tính video được đặt thành **true**. Điều này yêu cầu truy cập vào video từ camera.

- Nếu quá trình yêu cầu truy cập camera thành công, hàm **navigator.getUserMedia** sẽ cung cấp một luồng video (stream). Luồng video này sẽ được gán cho phần tử video có tham chiếu là **video.current.**
- Hàm **addEventListener** được sử dụng để lắng nghe sự kiện "loadeddata" của phần tử video. Sự kiện này xảy ra khi dữ liệu video đã được tải và sẵn sàng cho việc hiển thị.
- Khi sự kiện "loadeddata" xảy ra, hàm resolve của Promise được gọi để báo hiệu rằng việc thiết lập camera đã hoàn thành thành công.
- Nếu có lỗi xảy ra trong quá trình yêu cầu truy cập camera, hàm reject của Promise sẽ được gọi để báo hiệu rằng việc thiết lập camera không thành công.

2.5.5. Hàm loadModel

```
const loadModel = async () => {
    try {
        tf.loadLayersModel();
        mobilenet.current = await mobilenetModule.load();
        net.current = await posenet.load();
        alert("thành công nhận diện");
        console.log("Cấm chạm tay lên mặt và bấm vào nút đầu tiên");
    } catch (error) {
        console.error(error);
    }
};
```

Hàm **loadModel** thực hiện việc tải mô hình máy học MobileNet để sử dụng trong chương trình và sau đó thông báo cho người dùng rằng việc tải đã hoàn thành thành công Hoạt động của hàm này có thể được phân tích như sau:

- **tf.loadLayersModel():** Đoạn mã này gọi phương thức loadLayersModel từ thư viện TensorFlow.js (được đặt tên là tf).
- **mobilenet.current = await mobilenetModule.load():** Đoạn mã này sử dụng thư viện mobilenetModule để tải mô hình máy học MobileNet. Nó sử dụng await để đợi cho đến khi quá trình tải mô hình hoàn thành trước khi tiếp tục. Khi quá trình tải xong, mô hình MobileNet được gán vào biến **mobilenet.current.**
- **net.current = await posenet.load()**: Đoạn mã gọi thư viện Posenet và lưu dữ liệu đã được thu thập và trong giá trị net.current.
- **alert("thành công nhận diện"):** Sau khi mô hình MobileNet đã được tải thành công, một cửa sổ thông báo xuất hiện với thông điệp "thành công nhận diện" để thông báo cho người dùng rằng ứng dụng đã sẵn sàng để sử dụng.

2.5.6. Sự kiện “document.body.onload”:

```
document.body.onload = async () => {
    console.log("init");
    setupCamera()
        .then(async () => {
            await loadModel();
        })
        .catch((error) => console.log(error));
};
```

Đoạn mã này gán một hàm callback vào sự kiện onload của thẻ body, tức là khi trang web được tải hoàn toàn, hàm callback sẽ được gọi. Đoạn mã này đảm bảo rằng khi trang web được tải hoàn toàn, camera sẽ được kết nối và mô hình máy học MobileNet sẽ được tải để chuẩn bị cho việc sử dụng trong ứng dụng

- **console.log("init"):** Đoạn mã này xuất thông điệp "init" ra console để thông báo rằng quá trình khởi tạo đang được thực hiện.
- **setupCamera().then(async () => { await loadModel();}):** Đoạn mã này gọi hàm **setupCamera()** để thiết lập camera của thiết bị người dùng và sau đó chờ cho đến khi camera đã sẵn sàng. Khi camera đã sẵn sàng, nó sẽ tiếp tục gọi hàm **loadModel()** để tải mô hình máy học MobileNet. Việc sử dụng async/await ở đây giúp đảm bảo rằng hàm **loadModel()** chỉ được gọi sau khi **setupCamera()** hoàn thành.
- **.catch((error) => console.log(error)):** Nếu có bất kỳ lỗi nào xảy ra trong quá trình thiết lập camera hoặc tải mô hình, lỗi sẽ được bắt và thông điệp lỗi sẽ được xuất ra console để ghi nhận lỗi.

2.5.7. Hàm run

```
const run = async () => {
    const embedding = mobilenet.current.infer(video.current, true);
    const result = await classifier.predictClass(embedding, NUM_NEIGHBOR);
    await drawImagePoint();
    console.log(result.confidences);
    const isTouch = result.label === TOUCH_LABEL;
    setIsTouch(isTouch);
    if (isTouch) {
        sound.play();
    }
    await sleep(1000);
    run();
};
```

Hàm run chịu trách nhiệm liên tục trích xuất đặc trưng từ hình ảnh camera, dự đoán nhãn sử dụng mô hình KNN, và cập nhật trạng thái của ứng dụng dựa trên kết quả dự đoán. Nó cũng khởi động âm thanh và thông báo khi tay của người dùng chạm lên mặt.

- **const embedding = mobilenet.current.infer(video.current, true):** Hàm này sử dụng mô hình MobileNet để trích xuất đặc trưng từ hình ảnh hiện tại của camera (video.current). Kết quả được lưu trữ trong biến embedding.
- **const result = await classifier.predictClass(embedding):** Hàm này sử dụng mô hình KNN đã được huấn luyện để dự đoán nhãn của hình ảnh trích xuất từ MobileNet. Kết quả dự đoán được lưu trữ trong biến result.

- **console.log(result.confidences):** Hàm này xuất ra console giá trị độ tự tin (confidence) của mô hình KNN trong việc dự đoán nhãn. Giá trị độ tự tin này được lưu trong thuộc tính confidences của biến result.
- **const isTouch = result.label === TOUCH_LABEL:** Hàm này so sánh nhãn dự đoán (result.label) với nhãn "touch" (TOUCH_LABEL). Nếu nhãn dự đoán trùng khớp với nhãn "touch", biến isTouch sẽ được gán giá trị true, ngược lại sẽ là false.
- **await drawImagePoint();:** Gọi hàm drawImagePoint để chương trình có thể xác định và vẽ các điểm ảnh.
- **setIsTouch(isTouch):** Hàm này sử dụng setIsTouch để cập nhật trạng thái "có chạm" hoặc "không chạm" của ứng dụng dựa trên giá trị isTouch.
- **if (isTouch) { sound.play(); }:** Nếu giá trị của biến isTouch là true, thì hàm này sẽ chơi âm thanh bằng cách gọi sound.play().
- **await sleep(1000):** Hàm này tạm dừng thực thi trong 2 giây (2000 miligiây) bằng cách sử dụng hàm sleep. Điều này giúp làm chậm vòng lặp và đảm bảo rằng dự đoán và xử lý sẽ được thực hiện định kỳ.
- **run():** Cuối cùng, hàm run tự gọi lại chính nó bằng cách sử dụng await run(). Điều này tạo ra một vòng lặp vô hạn trong đó các dự đoán và xử lý sẽ tiếp tục được thực hiện liên tục cho đến khi ứng dụng được đóng hoặc có lỗi xảy ra.

2.5.8. Hàm drawImagePoint

```
const drawImagePoint = async () => {
    const pose = await net.current.estimateSinglePose(video.current);
    const canvas = document.createElement("canvas");
    canvas.width = 360;
    canvas.height = 240;
    video.current.appendChild(canvas);
    const ctx = canvas.getContext("webgpu");

    ctx.drawImage(video.current, 0, 0, 360, 240);
    // Vẽ các điểm chính lên canvas
    pose.keypoints.forEach((keypoint) => {
        ctx.beginPath();
        ctx.arc(keypoint.position.x, keypoint.position.y, 10, 0, 2 * Math.PI);
        ctx.fillStyle = "red";
        ctx.fill();
    });
    console.log(pose.keypoints);
};
```

Hàm drawImagePoint xác định vị trí các điểm ảnh của mô hình KNN. Quá trình này giúp ta theo dõi được các vị trí K mà các thư viện đã thiết lập trong quá trình hoạt động của hệ thống. Hàm này được thực hiện bởi các câu lệnh với quá trình hoạt động như sau:

- **const pose = await net.current.estimateSinglePose(video.current);:** Đoạn này sử dụng mô hình của TensorFlow.js (được lưu trữ trong biến net.current) để ước tính một bức tranh (pose) từ video nguồn (được tham chiếu qua biến video.current). Pose ở đây là tọa độ

của các điểm quan trọng trên cơ thể hoặc các đối tượng khác được theo dõi trong video.

- Tiếp theo, chúng ta tạo một canvas mới để vẽ pose. Canvas này có kích thước là 360x240 và được thêm vào phần tử video (từ biến video.current). Canvas là nơi chúng ta sẽ vẽ pose và hiển thị nó lên màn hình.
- **const ctx = canvas.getContext("2d");:** Chúng ta lấy một ngữ cảnh 2D (context) từ canvas. Context này cho phép chúng ta vẽ các hình và điểm lên canvas.
- **ctx.drawImage(video.current, 0, 0, 360, 240);:** Đoạn này sao chép hình ảnh từ video (được tham chiếu qua video.current) và vẽ nó lên canvas. Hình ảnh sẽ được vẽ bắt đầu từ vị trí (0, 0) trên canvas và có kích thước 360x240.
- **console.log(pose.keypoints);:** Cuối cùng, chúng ta in ra các tọa độ của các keypoints trên console, để kiểm tra và theo dõi dữ liệu.

Output thông báo:

2.5.9. Hàm calculateAccuracy (trước khi tối ưu hóa)

```
const calculateAccuracy = async () => {
    if (dataset.length === 0) {
        //nếu chưa có dataset thì dừng hàm
        console.log("Chưa có dataset");
        return;
    }
    const k = Math.floor(Math.random() * 95) + 5;
    let numCorrect = 0;
    const numExample = dataset.length;
    for (let i = 0; i < numExample; i++) {
        const example = dataset[i];
        console.log(example);
        const embedding = mobilenet.current.infer(example.image, true);
        const result = await classifier.predictClass(embedding, k).catch((error) => console.log(error));
        console.log(result);
        if (result.label === example.label) {
            numCorrect++;
        }
    }
    console.log(numCorrect);
    const accuracy = (numCorrect / numExample) * 100;
    console.log(`Với k = ${k} , thuật toán có độ chính xác: ${accuracy.toFixed(2)}%`);
};
```

Hàm calculateAccuracy xử lý quá trình phân tích và tính toán độ chính xác của mô hình KNN. Quá trình này giúp ta tìm được số lượng K tối ưu nhất, chính xác nhất cho mô hình, số liệu K tối ưu nhất sẽ được phân tích tại phần sau của báo cáo. Hàm này được thực hiện bởi các câu lệnh với quá trình hoạt động như sau:

- Kiểm tra nếu dataset (danh sách các ví dụ huấn luyện) rỗng, tức là chưa có dữ liệu để tính độ chính xác, thì hàm sẽ dừng lại và xuất thông báo "Chưa có dataset".
- Tạo một số ngẫu nhiên k trong khoảng từ 5 đến 99. Giá trị k này sẽ được sử dụng để định số lượng láng giềng gần nhất mà thuật toán KNN sẽ sử dụng để dự đoán.
- Khởi tạo một biến **numCorrect** để đếm số ví dụ được dự đoán đúng.

- Lấy số lượng ví dụ trong dataset bằng cách sử dụng **dataset.length** và lưu vào biến **numExample.**
- Bắt đầu vòng lặp qua từng ví dụ trong dataset:
  a. Trích xuất đặc trưng từ hình ảnh của ví dụ bằng cách sử dụng mô hình MobileNet và lưu vào biến embedding.
  b. Sử dụng mô hình KNN để dự đoán nhãn của ví dụ bằng cách sử dụng **classifier.predictClass(embedding, k)** với số lượng láng giềng k đã được định.
  c. So sánh nhãn dự đoán (result.label) với nhãn thực tế của ví dụ (example.label). Nếu trùng khớp, tăng giá trị của biến **numCorrect** lên 1.
- Xuất ra console số lượng ví dụ được dự đoán đúng (**numCorrect**).
- Tính độ chính xác (**accuracy**) bằng cách chia số ví dụ được dự đoán đúng (**numCorrect**) cho tổng số ví dụ (**numExample**) và nhân với 100 để có giá trị phần trăm.
- Xuất ra console thông tin về độ chính xác với giá trị k ngẫu nhiên và độ chính xác tính được. Giá trị độ chính xác được làm tròn đến 2 chữ số thập phân.

2.5.10. Hàm calculateAccuracy (sau khi tối ưu hóa)

```
const calculateAccuracy = async () => {
    if (dataset.length === 0) {
        //nếu chưa có dataset thì dừng hàm
        console.log("Chưa có dataset");
        return;
    }
    let statistic = {
        numCorrect: 0,
        numTouch: 0,
        numCorrectTouch: 0,
    };
    const numExample = dataset.length;
    for (let i = 0; i < numExample; i++) {
        progress.current.value = ((i + 1) / numExample) * 100;
        const example = dataset[i];
        console.log(example);
        const embedding = mobilenet.current.infer(example.image, true);
        const result = await classifier.predictClass(embedding, NUM_NEIGHBOR);
        console.log(result);
        const isTrueCase = result.label === example.label;
        const isTouchCase = result.label === TOUCH_LABEL;
        if (isTrueCase) {
            statistic.numCorrect++;
        }
        if (isTouchCase) {
            if (isTrueCase) statistic.numCorrectTouch++;
            statistic.numTouch++;
        }
    }
    console.log(statistic.numCorrect);
    const accuracy = (statistic.numCorrect / numExample) * 100;
    const sensitivity = (statistic.numTouch / numExample) * 100;
    const specificityTouch = (statistic.numCorrectTouch / statistic.numTouch) * 100;
    const specificityNotTouch = ((statistic.numCorrect - statistic.numCorrectTouch) / (numExample - statistic.numTouch)) * 100;
    document.write(`
    Các thông số hiện tại:<br/>
    Accuracy : ${accuracy.toFixed(2)}%,<br/>
    Recall: ${sensitivity}% với lớp ${TOUCH_LABEL} , ${100 - sensitivity}% với lớp ${NOT_TOUCH_LABEL},<br/>
    Precision : ${specificityTouch}% với lớp ${TOUCH_LABEL}, ${specificityNotTouch}% với lớp ${NOT_TOUCH_LABEL},
    `);
};
```

Hàm calculateAccuracy xử lý quá trình phân tích và tính toán độ chính xác của mô hình KNN. Sau khi mô hình đã được tối ưu, hàm này sẽ in ra các thông số hiệu suất của mô hình dự đoán trong quá trình kiểm tra như Accuracy, Recall và Precision. Hàm này được thực hiện bởi các câu lệnh với quá trình hoạt động như sau:

- Kiểm tra nếu dataset (danh sách các ví dụ huấn luyện) rỗng, tức là chưa có dữ liệu để tính độ chính xác, thì hàm sẽ dừng lại và xuất thông báo "Chưa có dataset".

- Khởi tạo một đối tượng statistic để theo dõi các thống kê. numCorrect là số lượng ví dụ được dự đoán đúng, numTouch là số lượng ví dụ thuộc lớp **TOUCH_LABEL**, và numCorrectTouch là số lượng ví dụ thuộc lớp **TOUCH_LABEL** được dự đoán đúng. Khởi tạo một biến **numCorrect** để đếm số ví dụ được dự đoán đúng.
- Tiến hành duyệt qua từng ví dụ trong dataset để đánh giá mô hình dự đoán:
    - **progress.current.value = ((i + 1) / numExample) * 100;:** Cập nhật giá trị thanh tiến trình (progress bar) để theo dõi tiến trình tính toán. Nó sẽ hiển thị phần trăm hoàn thành.
    - Lấy ra một ví dụ từ dataset: **const example = dataset[i];**
    - Tính toán nhúng (embedding) của ví dụ bằng mô hình **Mobilenet: const embedding = mobilenet.current.infer(example.image, true);**
    - Sử dụng mô hình phân loại KNN để dự đoán lớp của ví dụ và số lượng hàng xóm (NUM_NEIGHBOR là số lượng hàng xóm sẽ được sử dụng): **const result = await classifier.predictClass(embedding, NUM_NEIGHBOR);**
    - Kiểm tra xem dự đoán có chính xác không và xác định xem nó thuộc lớp TOUCH_LABEL không: **const isTrueCase = result.label === example.label; và const isTouchCase = result.label === TOUCH_LABEL;**
    - Cập nhật statistic dựa trên kết quả dự đoán. Nếu dự đoán đúng (isTrueCase), tăng numCorrect lên. Nếu dự đoán thuộc lớp TOUCH_LABEL (isTouchCase), tăng numTouch và numCorrectTouch lên.

- Bắt đầu vòng lặp qua từng ví dụ trong dataset:
    - Trích xuất đặc trưng từ hình ảnh của ví dụ bằng cách sử dụng mô hình MobileNet và lưu vào biến embedding.
    - Sử dụng mô hình KNN để dự đoán nhãn của ví dụ bằng cách sử dụng **classifier.predictClass(embedding, k)** với số lượng láng giềng k đã được định.
    - So sánh nhãn dự đoán (result.label) với nhãn thực tế của ví dụ (example.label). Nếu trùng khớp, tăng giá trị của biến **numCorrect** lên 1.
- Tính toán các thông số đánh giá hiệu suất:
    - **Accuracy (Độ chính xác):** Tính theo tỷ lệ số lượng ví dụ được **dự đoán đúng** trên tổng số ví dụ trong dataset.
    - **Recall (Độ nhạy):** Tính theo tỷ lệ số lượng ví dụ được dự đoán thuộc lớp **TOUCH_LABEL**. Nó cũng tính tỷ lệ số lượng ví dụ thuộc lớp **NOT_TOUCH_LABEL** tương tự.
    - **Precision (Độ cụ thể) :** Tính theo tỷ lệ số lượng ví dụ thuộc lớp **TOUCH_LABEL** được dự đoán đúng trên tổng số ví dụ được dự đoán là thuộc lớp **TOUCH_LABEL.** Nó cũng tính tỷ lệ số lượng ví dụ thuộc lớp **NOT_TOUCH_LABEL** được dự đoán đúng.
- Cuối cùng, hàm sử dụng **document.write** để hiển thị kết quả đánh giá hiệu suất. Nó sẽ xuất ra các thông số như Accuracy, Recall, và Precision trên trang web.

Output kết quả:

Thuật toán có độ chính xác: 32.00%,
độ nhạy: 81.5% với lớp touch, 18.5% với lớp not_touch,
độ cụ thể: 38.95705521472393% với lớp touch, 1.3513513513513513% với lớp not_touch,

CHƯƠNG III: HOÀN THIỆN CHƯƠNG TRÌNH

1. Tối ưu hóa, thử nghiệm và đánh giá mô hình

Quá trình tối ưu hóa mô hình được phân chia theo các công việc như sau:

1.1. Phân chia dữ liệu:

Ta cần đánh giá cách phân chia dữ liệu huấn luyện và kiểm thử. Đảm bảo rằng mô hình được huấn luyện trên một tập dữ liệu đủ lớn và đa dạng để tránh overfitting.

Quá trình này đã được thực hiện tại phần khai báo hai nhãn **"NOT_TOUCH_LABEL"** và **"TOUCH_LABEL".** Việc này nhằm xác định cho mô hình được huấn luyện để phân biệt các giá trị chỉ vào hai nhãn này. Bên cạnh đó, mô hình sẽ lưu lại tất cả các dữ liệu huấn luyện thu nhận được trong quá trình hàm **training()** được khởi chạy với thời gian hoạt động **"TRAINING_TIME"**. Quá trình này có khả năng điều chỉnh thời gian thu nhận giá trị, giúp ta theo dõi, phân tích và xác định giá trị tối ưu nhất cho mô hình.

1.2. Tinh chỉnh tham số:

Quá trình này yêu cầu ta phải theo dõi và chỉnh sửa các giá trị **"TRAINING_TIME"** và **"TOUCH_CONFIDECE"** và **số lượng điểm hàng xóm K** (đã trình bày tại phần 2.3 của chương II) cho tối ưu nhất với chương trình.

Nhóm em đã thực hiện quá trình khởi chạy chương trình nhiều lần, qua đó theo dõi và kết luận giá trị tối ưu nhất, dựa vào chỉ số về **độ chính xác của mô hình** và **tốc độ xử lý.** Số liệu được tổng quát hóa như sau:

- Đối với **"TRAINING_TIME":** Sau quá trình kiểm nghiệm với số lần 40, set giá trị **"TRAINING_TIME"** trải từ 10 đến 400 với

bước nhảy giá trị 10, nhóm em nhận thấy rằng giá trị **"TRAINING_TIME"** tối ưu nhất là khi **"TRAINING_TIME"** có giá trị bằng 200 với độ chính xác 95.75% và tốc độ xử lý ở mức 214 giây. Có thể nhận thấy rằng với giá trị **"TRAINING_TIME"** thì độ chính xác của mô hình đạt mức cao nhất với 96.71%, tuy nhiên tốc độ xử lý lại đạt mức 298 giây. Với các số liệu như vậy, nhóm em quyết định đặt giá trị **"TRAINING_TIME"** ở mức 200, nhằm tối ưu hóa cả độ chính xác và tốc độ xử lý cho mô hình KNN.

- Đối với **"TOUCH_CONFIDECE":** Sau quá trình kiểm nghiệm với số lần 100, set giá trị **"TOUCH_CONFIDECE"** trải từ 0.01 (tương ứng với 1%) tới 1 (tương ứng với 100%) với bước nhảy giá trị 0.01, nhóm em nhận thấy rằng giá trị của **"TOUCH_CONFIDECE"** đạt mức tối ưu nhất là khi **"TOUCH_CONFIDECE"** có giá trị bằng 0.8, đạt mức 88% độ chính xác và tốc độ xử lý ở mức 223 giây. Với các số liệu như vậy, nhóm em quyết định đặt giá trị **"TOUCH_CONFIDECE"** ở mức 0.8, tương ứng với việc khi mô hình xác nhận dữ liệu thu được vượt mức 80% mới được ghi nhận và gắn nhãn phân loại nhằm tối ưu hóa cả độ chính xác và tốc độ xử lý cho mô hình KNN.
- Đối với **số lượng điểm hàng xóm K:** Sau quá trình kiểm nghiệm với số lần kiểm thử 500 lần, với **số lượng điểm hàng xóm K** được trải từ 1 tới 100 với bước nhảy 1, nhóm em thu nhận được những thông tin như sau:
    - Tỷ lệ chính xác cao nhất đạt mức 93% với K=91
    - Tỷ lệ chính xác thấp nhất chỉ đạt 10% với K=12

- Tỷ lệ chính xác cao nhất sau 5 lần kiểm thử với mỗi giá trị K đạt 74.4% với K=62
- Tỷ lệ chính xác thấp nhất sau 5 lần kiểm thử với mỗi giá trị K đạt 39.4% với K=4
- Giá trị K=1 không thỏa mãn cho mô hình khi không thể xác định khoảng cách giữa hai điểm trong không gian, vì vậy không phù hợp cho cả quá trình huấn luyện lẫn sử dụng mô hình.

- Với các số liệu như vậy, nhóm em có thể tìm ra và kết luận các giá trị tối ưu nhất cho mô hình thuật toán của mình

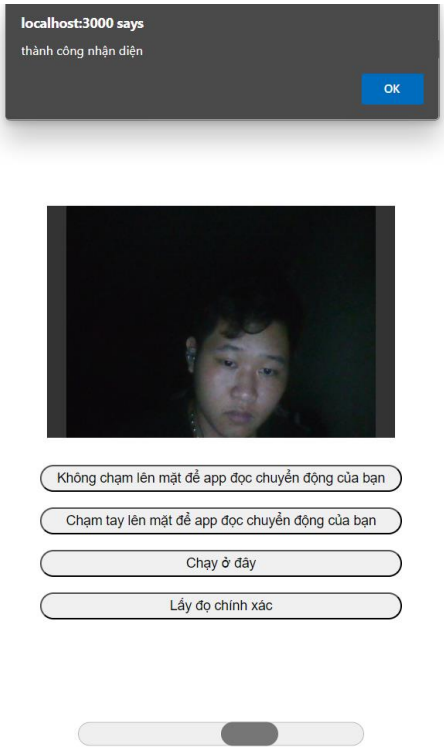
1.3. Đánh giá độ tin cậy của mô hình:

Sau khi kết luận được các giá trị tối ưu nhất cho mô hình với **"TRAINING_TIME"** là 200 lần, **"TOUCH_CONFIDECE"** là 80% và **số lượng điểm hàng xóm K** là 62 điểm, nhóm em khởi chạy chương trình và thu được kết quả với độ tin cậy 72.32%. Chi tiết về các số liệu kiểm thử đã phân tích để tối ưu hóa mô hình đã được ghi chép lại có thể theo dõi tại Sheet sau: File Test kiểm thử giá trị

2. Tối ưu hóa trải nghiệm người dùng

2.1. Tối ưu hóa giao diện người dùng (GUI):

Nhóm em mong muốn quá trình sử dụng chương trình được đơn giản hóa và dễ sử dụng nhất đối với mọi người dùng, vì vậy giao diện không cầu kỳ, chỉ tập trung vào các chức năng của chương trình.



2.2. Tối ưu âm thanh và thông báo:

- Tối ưu âm thanh: Đảm bảo rằng âm thanh được phát một cách trơn tru khi có thông báo.
- Tối ưu thông báo: Hiển thị thông báo dễ hiểu khi có lỗi hoặc khi máy nhận diện chuyển động.

3. Hoàn thiện mã nguồn hệ thống

Sau quá trình theo dõi, phân tích và tối ưu cho mô hình, mã nguồn của hệ thống được tổng kết lại như sau:

```
import React, { useRef, useState } from "react";
import "./App.css";
import { Howl } from "howler";
import heySound from "./assets/hey_sondn.mp3";
import * as knnClassifier from "@tensorflow-models/knn-classifier";
import * as mobilenetModule from "@tensorflow-models/mobilenet";
import * as tf from "@tensorflow/tfjs";
import * as posenet from "@tensorflow-models/posenet";

const sound = new Howl({
    src: [heySound],
```

```
});

// sound.play();

const NOT_TOUCH_LABEL = "not_touch";
const TOUCH_LABEL = "touch";
const TRAINING_TIME = 200;
const ERROR_MESSAGE = "Bỏ tay ra đi. Thứ tôi muốn thấy là nụ cười
của em";
const NUM_NEIGHBOR = 62;
let dataset = [];
function App() {
    const video = useRef();
    const mobilenet = useRef();
    const net = useRef();
    const progress = useRef();
    const classifier = knnClassifier.create();
    const [isTouch, setIsTouch] = useState(false);

    const sleep = (time) => new Promise((resolve) =>
setTimeout(resolve, time));
    const training = (label) => {
        return new Promise(async (resolve) => {
            const embedding = mobilenet.current.infer(video.current,
true);
            classifier.addExample(embedding, label);
            dataset = [...dataset, { image: video.current, label }];
            await sleep(100);
            resolve();
        });
    };
    const train = async (label) => {
        console.log(`[${label} đang check với gương mặt đẹp trai của
bạn]`);
        for (let i = 0; i < TRAINING_TIME; i++) {
            console.log(`progress ${progress.current.value} %`);
            await training(label);
            progress.current.value = ((i + 1) * 100) /
TRAINING_TIME;
            // setProgress(((i + 1) / TRAINING_TIME) * 100);
        }
        console.log("máy đã học xong");
    };
```

```
    const calculateAccuracy = async () => {
        if (dataset.length === 0) {
            //nếu chưa có dataset thì dừng hàm
            console.log("Chưa có dataset");
            return;
        }
        let statistic = {
            numCorrect: 0,
            numTouch: 0,
            numCorrectTouch: 0,
        };
        const numExample = dataset.length;
        for (let i = 0; i < numExample; i++) {
            progress.current.value = ((i + 1) / numExample) * 100;
            const example = dataset[i];
            console.log(example);
            const embedding = mobilenet.current.infer(example.image,
true);
            const result = await classifier.predictClass(embedding,
NUM_NEIGHBOR);
            console.log(result);
            const isTrueCase = result.label === example.label;
            const isTouchCase = result.label === TOUCH_LABEL;
            if (isTrueCase) {
                statistic.numCorrect++;
            }
            if (isTouchCase) {
                if (isTrueCase) statistic.numCorrectTouch++;
                statistic.numTouch++;
            }
        }
        console.log(statistic.numCorrect);
        const accuracy = (statistic.numCorrect / numExample) * 100;
        const sensitivity = (statistic.numTouch / numExample) * 100;
        const specificityTouch = (statistic.numCorrectTouch /
statistic.numTouch) * 100;
        const specificityNotTouch = ((statistic.numCorrect -
statistic.numCorrectTouch) / (numExample - statistic.numTouch)) *
100;
        document.write(`
        Các thông số hiện tại:<br/>
        Accuracy : ${accuracy.toFixed(2)}%,<br/>
```

```
        Recall: ${sensitivity}% với lớp ${TOUCH_LABEL} , ${100 -
sensitivity}% với lớp ${NOT_TOUCH_LABEL},<br/>
        Precision : ${specificityTouch}% với lớp ${TOUCH_LABEL},
${specificityNotTouch}% với lớp ${NOT_TOUCH_LABEL},
          `);
    };
    const drawImagePoint = async () => {
        const pose = await
net.current.estimateSinglePose(video.current);
        const canvas = document.createElement("canvas");
        canvas.width = 360;
        canvas.height = 240;
        video.current.appendChild(canvas);
        const ctx = canvas.getContext("2d");
        ctx.drawImage(video.current, 0, 0, 360, 240);

        // Vẽ các điểm chính lên canvas
        pose.keypoints.forEach((keypoint) => {
            ctx.beginPath();
            ctx.arc(keypoint.position.x, keypoint.position.y, 10, 0,
2 * Math.PI);
            ctx.fillStyle = "red";
            ctx.fill();
        });
        console.log(pose.keypoints);
    };

    const run = async () => {
        const embedding = mobilenet.current.infer(video.current,
true);
        const result = await classifier.predictClass(embedding,
NUM_NEIGHBOR);
        await drawImagePoint();
        console.log(result.confidences);
        const isTouch = result.label === TOUCH_LABEL;
        setIsTouch(isTouch);
        if (isTouch) {
            sound.play();
        }
        await sleep(1000);
        run();
    };
```

```
    const setupCamera = async () => {
        return new Promise((resolve, reject) => {
            navigator.getUserMedia = navigator.getUserMedia ||
navigator.mozGetUserMedia || navigator.mediaDevices.getUserMedia ||
navigator.webkitGetUserMedia || navigator.msGetUserMedia;
            if (navigator.getUserMedia) {
                navigator.getUserMedia(
                    {
                        video: true,
                    },
                    (stream) => {
                        video.current.srcObject = stream;
                        video.current.addEventListener("loadeddata",
resolve);
                    },
                    (error) => reject(error)
                );
            } else reject();
        });
    };
    const loadModel = async () => {
        try {
            tf.loadLayersModel();
            mobilenet.current = await mobilenetModule.load();
            net.current = await posenet.load();
            alert("thành công nhận diện");
            console.log("Cấm chạm tay lên mặt và bấm vào nút đầu
tiên");
        } catch (error) {
            console.error(error);
        }
    };
    document.body.onload = async () => {
        console.log("init");
        setupCamera();
        loadModel();
    };
    return (
        <div className={`main ${isTouch && "touched"}`}>
            <video
                className="video"
                autoPlay
                ref={video}
```

```
                id="video"
            />
            <div className="control">
                <button
                    className="btn"
                    onClick={async () => {
                        await train(NOT_TOUCH_LABEL);
                    }}
                >
                    Không chạm lên mặt để app đọc chuyển động của
bạn
                </button>
                <button
                    className="btn"
                    id="js-train-not-touch"
                    onClick={async () => {
                        await train(TOUCH_LABEL);
                    }}
                >
                    Chạm tay lên mặt để app đọc chuyển động của bạn
                </button>
                <button
                    className="btn"
                    id="js-run"
                    onClick={() => {
                        run().catch((error) => {
                            alert("Máy chưa được học, hãy để cho máy
học rồi mới chạy");
                            console.log(error);
                        });
                    }}
                >
                    Chạy ở đây
                </button>
                <button
                    onClick={() => calculateAccuracy()}
                    className="btn"
                >
                    Lấy các thông số
                    <br />
                    DEVELOPER ONLY
                </button>
            </div>
```

```
            {isTouch && (
                <div className="modal">
                    <img
                        src="./assets/warning.png"
                        alt={ERROR_MESSAGE}
                        id="error"
                        style={{
                            width: "20%",
                        }}
                    />
                </div>
            )}
            <progress
                max="100"
                style={{
                    height: 50,
                    width: "20%",
                    position: "absolute",
                    bottom: 0,
                }}
                ref={progress}
            ></progress>
        </div>
    );
}

export default App;
```

4. Cải thiện và mở rộng chức năng

Nhóm chúng em đề ra các phương án phát triển và mở rộng hơn cho mô hình như sau

- **Tích hợp thêm chức năng nhận diện:** Nếu phù hợp, xem xét việc tích hợp nhận diện các hành động khác như cử chỉ hoặc biểu cảm khuôn mặt.
- **Tích hợp chức năng bảo mật:** Xem xét việc tích hợp chức năng bảo mật, ví dụ như xác thực hai yếu tố.

- **Phát triển giao diện người dùng GUI:** Xây dựng và thiết kế một giao diện thân thiện, bắt mắt và tối ưu nhất cho hệ thống.
- **Tích hợp thêm các thuật toán khác cho mô hình:** Việc sử dụng nhiều thuật toán xen kẽ với nhau có thể giúp cho quá trình hoạt động của mô hình mang lại một kết quả chính xác hơn.

Bên cạnh đó, mô hình vẫn còn cần phải cải thiện hiệu suất và tốc độ xử lý, độ chính xác và tin cậy của mô hình trong quá trình ứng dụng thực tế.

Những kế hoạch và dự định này sẽ được nhóm chúng em phân tích, thảo luận và xây dựng trong thời gian sắp tới.

Kết luận chung

Trong báo cáo này, cả nhóm đã tìm hiểu về quá trình xây dựng một ứng dụng học máy sử dụng JavaScript và thư viện TensorFlow.js. Chương trình của nhóm ta sử dụng mô hình K-Nearest Neighbors (KNN) để phân loại việc chạm tay vào khuôn mặt của người dùng.

Cả nhóm đã bắt đầu bằng việc xây dựng dữ liệu huấn luyện, trong đó nhóm đã thu thập các hình ảnh khuôn mặt của người dùng khi họ không chạm vào khuôn mặt (not_touch) và khi họ chạm vào khuôn mặt (touch). Điều này giúp mô hình học được cách phân biệt giữa hai trạng thái này.

Sau đó, nhóm đã sử dụng mô hình Mobilenet để trích xuất các đặc trưng từ các hình ảnh khuôn mặt. Các đặc trưng này được sử dụng làm dữ liệu đầu vào cho mô hình KNN.

Quá trình huấn luyện mô hình KNN đã được thực hiện bằng cách theo dõi và phân tích số liệu. Nhóm đã lặp lại quá trình huấn luyện nhiều lần để đảm bảo rằng mô hình đã học được cách phân loại đúng.

Sau khi mô hình đã được huấn luyện, nhóm đã sử dụng nó để dự đoán việc chạm tay vào khuôn mặt của người dùng thông qua webcam. Mô hình dự đoán dựa trên khoảng cách giữa dữ liệu mới và dữ liệu huấn luyện đã biết.

Chương trình cũng có khả năng tính toán độ chính xác của mô hình dựa trên một giá trị K (số lượng hàng xóm gần nhất) ngẫu nhiên. Điều này giúp cả nhóm đánh giá khả năng phân loại của mô hình. Cuối cùng, nhóm đã hoàn thiện chương trình và tạo giao diện đơn giản cho người dùng.

Chương trình có khả năng cảnh báo khi phát hiện chạm tay vào khuôn mặt và thể hiện độ chính xác của mô hình.

Trong báo cáo này, cả nhóm đã thấy sự mạnh mẽ của JavaScript và TensorFlow.js trong việc xây dựng ứng dụng học máy và ứng dụng phân loại thời gian thực. Nhóm đã thực hiện một ứng dụng thú vị có thể có nhiều ứng dụng thực tế, từ việc kiểm soát quyền truy cập vào các khuôn mặt đến việc tạo ra các ứng dụng tương tác sử dụng học máy.

Quá trình học tập và ứng dụng các kiến thức được học từ bộ môn Machine Learning qua việc thảo luận, lên ý tưởng, thiết kế, thực hiện và tối ưu hóa mô hình học máy đã giúp cho nhóm em có những kiến thức mới về khả năng của trí tuệ nhân tạo và những tác vụ đặc biệt mà nó có thể thực hiện.

Lời cuối cùng, nhóm em xin chân thành cảm ơn sự dạy dỗ của thầy Trần Anh Đạt trong bộ môn Học máy, cảm ơn toàn thể các thành viên đã luôn nỗ lực trong quá trình xây dựng hệ thống để đạt được những kết quả tốt nhất. Dù cho hệ thống vẫn còn nhiều khuyết điểm và sai sót, nhóm em xin ghi nhận và sẽ cải thiện trong những lần phát triển tiếp theo của dự án.