**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**CƠ SỞ TẠI TPHCM**

**KHOA CÔNG NGHỆ THÔNG TIN II**

**MÔN THỊ GIÁC MÁY TÍNH**



**BÁO CÁO CUỐI KỲ**

**NHẬN DẠNG TRẠNG THÁI NGỦ GẬT DÙNG PYTORCH**

GVHD: Hồ Nhựt Minh

Sinh viên thực hiện: Hoàng Ngọc Hưng

MSSV: N19DCPT023

Lớp: D19CQPU01-N

*Năm học 2022 – 2023*

LỜI NÓI ĐẦU

Thế giới hôm nay đang chứng kiến những đổi thay có tính chất “khuynh đảo” trong mọi hoạt động phát triển kinh tế - xã hội nhờ những thành tựu của công nghệ thông tin (CNTT). CNTT đã góp phần quan trọng cho việc tạo ra những nhân tố năng động mới, cho quá trình hình thành nền kinh tế tri thức và xã hội thông tin. Chính vì thế, việc ứng dụng CNTT trong các hoạt động là vô cùng cần thiết.

Việc ứng dụng CNTT để ứng dụng phát hiện ngủ gật trong các lớp học online để nhắc nhở học viên hay cảnh báo các tài xế khi lái xe giúp giảm tỉ lệ tai nạn.

Với khoảng thời gian không nhiều, vừa nghiên cứu tìm hiểu khai thác ngôn ngữ mới, vừa thực hiện chương trình. Và nhờ sự quan tâm giúp đỡ tận tình của thầy ***Hồ Nhựt Minh,*** em đã hoàn thành đề tài của mình.

Tuy nhiên trong quá trình làm không thể tránh khỏi nhầm lẫn, còn có nhiều sai sót nên em rất mong nhận được những ý kiến đóng góp của thầy cùng toàn thể các bạn trong lớp để đề tài của em được hoàn thiện. Và cũng là góp ý cho em hoàn thành tốt hơn các đề tài tiếp sau.

*Em xin chân thành cảm ơn!*

**LỜI CẢM ƠN**

Em xin chân thành cảm ơn khoa CNTT trường Học viện công nghệ Bưu Chính Viễn Thông đã tạo điều kiện tốt cho em thực hiện tốt đề tài.

Đặc biệt, em xin bày tỏ lòng biết ơn sâu sắc tới thầy ***Hồ Nhựt Minh*** người đã tận tình hướng dẫn em trong suốt quá trình thực hiện đề tài.

Mặc dù đã rất cố gắng nhưng đề tài em thực hiện không tránh khỏi còn nhiều thiếu sót. Em rất mong nhận được sự góp ý, phê bình của quý thầy cô, anh chị và các bạn.

Một lần nữa em xin chân thành cảm ơn!

Sinh viên thực hiện:

Hoàng Ngọc Hưng

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

Giảng viên hướng dẫn

Hồ Nhựt Minh

**MỤC LỤC**

[**CHƯƠNG 1: TỔNG QUAN**](#_heading=h.m4nxeefvdf2f) **7**

[1. Giới thiệu đề tài](#_heading=h.3dy6vkm) 7

[2. Mục tiêu của đề tài](#_heading=h.1t3h5sf) 7

[3. Phạm vi nghiên cứu](#_heading=h.jb8oih7z6tvg) 7

[4. Phương pháp thực hiện](#_heading=h.4d34og8) 7

[5. Kết quả mong muốn đạt được](#_heading=h.2s8eyo1) 8

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**](#_heading=h.17dp8vu) **9**

[1. Pytorch](#_heading=h.26in1rg) 9

[2. Convolutional neural network](#_heading=h.lnxbz9) 10

[3. Loss function](#_heading=h.m5l7oeexeozy) 15

[4. Optimizer](#_heading=h.vnqdkcpdmkwn) 15

[5. Fine-turning](#_heading=h.auqtqbvft3nq) 15

[6. Các bước triển khai 1 model](#_heading=h.mfxuodwz01rq) 15

[**CHƯƠNG 3: THIẾT KẾ HỆ THỐNG**](#_heading=h.17dp8vu) **24**

[1. Yêu cầu hệ thống](#_heading=h.pve8hw5mr936) 24

[2. Mô hình hệ thống](#_heading=h.64iu7a8yipo6) 24

[**CHƯƠNG 4: KẾT LUẬN VÀ KẾT QUẢ CHI TIẾT**](#_heading=h.17dp8vu) **31**

[1. Kết quả huấn luyện và kiểm tra](#_heading=h.urvms0plh8en) 31

[2. So sánh với các model khác](#_heading=h.urvms0plh8en) 35

[3. Ứng dụng](#_heading=h.sndwppi9xv92) 40

[**Tài liệu tham khảo**](#_heading=h.vtobysg6s7h1) **43**

# CHƯƠNG 1: TỔNG QUAN

### Giới thiệu đề tài

Ngủ gật là 1 vấn đề thường xuyên xảy ra khi học online. Với số lượng học viên nhiều, giáo viên không thể kiểm soát được hết. Ngoài ra nhiều nghiên cứu khác nhau đã chỉ ra rằng khoảng 20% ​​các vụ tai nạn đường bộ có liên quan đến mệt mỏi, lên đến 50% trên một số con đường nhất định. Nhận dạng ngủ gật là một công nghệ giúp cảnh báo đến các học viên và giáo viên khi có người ngủ gật hay cảnh báo đến các tài xế khi lái xe làm ngăn ngừa các tai nạn xảy ra do người lái xe buồn ngủ.

### Mục tiêu của đề tài

Đề tài “Nhận dạng trạng thái ngủ gật bằng pytorch” được thực hiện nhằm mục đích:

* Xây dựng mô hình mạng CNN ứng dụng trong việc nhận diện trạng thái ngủ gật trên máy tính.
* Ứng dụng mô hình CNN để nhận diện được trạng thái ngủ gật và cảnh báo cho người dùng.

### Phạm vi nghiên cứu

Trong đồ án này, người thực hiện chỉ xem xét thuật toán nhận dạng trạng thái ngủ gật qua một vài kiến trúc khác nhau của mô hình CNN và đánh giá mô hình dựa trên biểu đồ độ chính xác và loss function. Sau đó chọn ra mô hình phù hợp trong thư viện PyTorch để triển khai hệ thống nhận diện khuôn mặt.

### Phương pháp thực hiện

Trong đồ án này, người thực hiện chỉ xem xét thuật toán nhận dạng cảm xúc khuôn mặt qua một vài kiến trúc khác nhau của mô hình CNN và đánh giá mô hình dựa trên biểu đồ độ chính xác và loss function.

Đề tài này sử dụng một vài phương pháp nghiên cứu nhưng chủ yếu là phương pháp phân tích và tổng kết kinh nghiệm. Cụ thể là từ những công trình nghiên cứu liên quan đến đề tài và sự hỗ trợ từ thư viện Máy học PyTorch để đề xuất một cách tiếp cận mới trong giải quyết vấn đề đặt ra.

### Kết quả mong muốn đạt được

Mô hình đạt độ chính xác trên 95% và thời gian xử lý cho dữ liệu đầu vào thấp hơn 1/24 giây

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

### Pytorch

Pytorch chính là một framework hỗ trợ Deep Learning được phát triển bởi Facebook. (Bên cạnh Amazon, Google hay Apple, Facebook được biết đến là đơn vị công nghệ đầu tư rất nhiều nguồn lực cho việc phát triển trí tuệ nhân tạo).

Phát triển Pytorch với mã nguồn mở Facebook đã tạo được một cộng đồng chia sẻ rất lớn. Với nguồn tài nguyên rộng lớn trên cộng đồng này các vấn đề mà bạn đang mắc phải có thể đã có ai đó giải quyết và được chia sẻ lên cộng đồng.

Việc nhiều người sử dụng Pytorch cũng như Tensorflow hay Keras để giải quyết các bài toán về Deep Learning, đã làm chúng trở thành những framework phổ biến nhất hiện nay. Đặc biệt phải kể đến lĩnh vực nghiên cứu, các nhà phát triển hay tác giả hầu như ưu tiên sử dụng Pytorch giúp triển khai các bài toán của mình bởi Pytorch dễ dàng giúp chúng bạn debug và Visuallize cùng với đó là cơ chế Dynamic Graphs cho phép giảm thời gian huấn luyện mô hình.

So với Tensorflow hay Keras, số lượng người dùng sử dụng Pytorch cũng không hề kém cạnh bởi những lợi ích Pytorch đem lại vì:

* Mã nguồn mở: như đã chia sẻ ở trên, nhờ sử dụng mã nguồn mở đã tạo nên một cộng đồng rất lớn với nguồn tài nguyên “chất lượng” và “số lượng”.
* Khả năng xử lý đồ họa: như Numpy đồng thời có kiểm soát CPU & GPU rõ ràng.
* Tập hợp nhiều Pythonic trong tự nhiên.
* Dễ dàng xử lý khi gặp bug.
* Có TouchScript được xem là một tập hợp con của Python. Tập hợp này giúp triển khai các ứng dụng vào quy mô sản xuất từ đó mở rộng quy mô. Đồng thời khi nói đến việc xây dựng các nguyên mẫu với tốc độ nhanh, sử dụng Pytorch được ưu tiên hơn so với Tensorflow vì nó nhẹ hơn.
* Các hàm, cú pháp cơ bản trong Pytorch giúp xử lí các bài toán về AI nhanh chóng.

Cách cài đặt

Môi trường windows:

Yêu cầu máy tính người dùng phải cài đặt trước python.

B1: mở cmd hoặc powershell nhập câu lệnh:

pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu116

Nếu máy tính người dùng có anaconda thì sử dụng câu lệnh:

conda install pytorch torchvision torchaudio pytorch-cuda=11.6 -c pytorch -c nvidia

B2: Import thư viện và các module sử dụng

from torch.utils.data import Dataset

import torch

import torch.nn as nn

from torchvision import transforms,models

from torch.utils.data import DataLoader

Google colab:

Trên google colab đã được cài đặt sẵn pytorch nên chỉ cần Import thư viện và các module mà mình sử dụng.

### Convolutional neural network

Convolutional Neural Network (CNN hoặc ConvNet) được tạm dịch là: Mạng nơ ron tích chập. Đây được xem là một trong những mô hình của Deep Learning – tập hợp các thuật toán để có mô hình dữ liệu trừu tượng hóa ở mức cao bằng cách sử dụng nhiều lớp xử lý cấu trúc phức tạp. Hiểu đơn giản, CNN là một lớp của mạng nơ-ron sâu , được áp dụng phổ biến nhất để phân tích hình ảnh trực quan.

Mạng CNN được thiết kế với mục đích xử lý dữ liệu thông qua nhiều lớp mảng. Ngoài ra, CNN có thể giúp bạn tạo ra được hệ thống thông minh, phản ứng với độ chính xác khá cao.

So với những mạng nơ ron thông thường, mạng CNN nhận đầu vào là một mảng hai chiều và hoạt động trực tiếp trên hình ảnh thay vì tập trung trích xuất tính năng mà bạn thường thấy ở các mạng nơ ron khác.

**Feature**

Feature được dịch theo nghĩa tiếng Việt là đặc điểm. Khi sử dụng thuật toán CNN so sánh hình ảnh theo từng mảnh, mỗi mảnh đó được gọi là Feature.

Mỗi Feature được xem như một hình ảnh mini hay gọi là những mảng hai chiều nhỏ. Các Feature được khớp với những khía cạnh chung trong bức ảnh đó. Nghĩa là Feature sẽ tương ứng với khía cạnh nào đó của hình ảnh và chúng sẽ khớp lại với nhau.

**Convolutional**

Convolutional được hiểu là tích chập. Xét về cơ bản, khi xem một hình ảnh mới, thuật toán CNN sẽ không nhận biết được nó ở vị trí nào, các Feature sẽ khớp với nhau ở đâu?

Chính vì vậy, Convolutional sẽ thử chúng với tất cả các vị trí khác nhau và tạo thành một bộ lọc gọi là Filter. Quá trình này được thực hiện thông qua phần toán nơ ron tích chập.

**Convolutional layer**

Trong ba lớp của Convolutional Neural Network, Convolutional Layer được xem là lớp có vai trò quan trọng nhất. Bởi vì Convolutional Layer sẽ đại diện CNN thực hiện mọi phép toán.

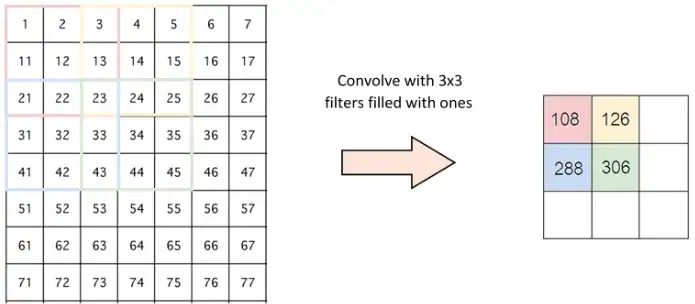
Khi nhắc đến lớp Convolutional Layer, chúng ta cần làm rõ một số khái niệm đó là: Filter Map, Stride, Padding, Feature Map.

**Filter Map**

Nếu như ANN kết nối với từng Pixel của hình ảnh đầu vào thì CNN được sử dụng những Filter để áp vào các vùng của hình ảnh. Những Filter Map này có thể xem là một ma trận 3 chiều, bao gồm những con số và các con số chính là Parameter.

**Stride**

Stride là số lượng pixel dịch chuyển trên ma trận đầu vào. Khi stride là 1 thì chúng ta di chuyển các bộ lọc đến 1 pixel cùng một lúc. Khi stride là 2 thì di chuyển các bộ lọc đến 2 pixel cùng một lúc, v.v. Hình dưới đây cho thấy phép tích chập sẽ hoạt động với stride là 2.



**Padding**

Đôi khi bộ lọc không phù hợp hoàn hảo với hình ảnh đầu vào. Chúng tôi có hai lựa chọn:

* Đệm ảnh bằng số không (0)
* Bỏ hình ảnh mà nơi bộ lọc không vừa. Chỉ giữ lại một phần hợp lệ của hình ảnh.

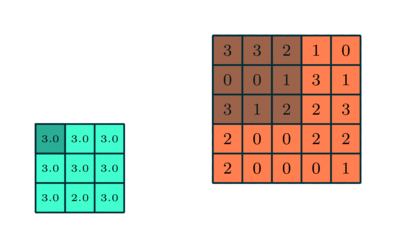
**Feature Map**

Đây là kết quả hiển thị sau mỗi lần Filter Map quét qua Input. Cứ mỗi lần quét như vậy, bạn sẽ thấy sự xuất hiện của quá trình tính toán được xảy ra.

**Pooling Layer**

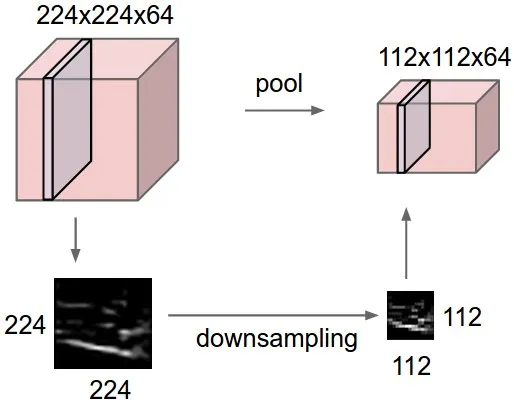
Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.

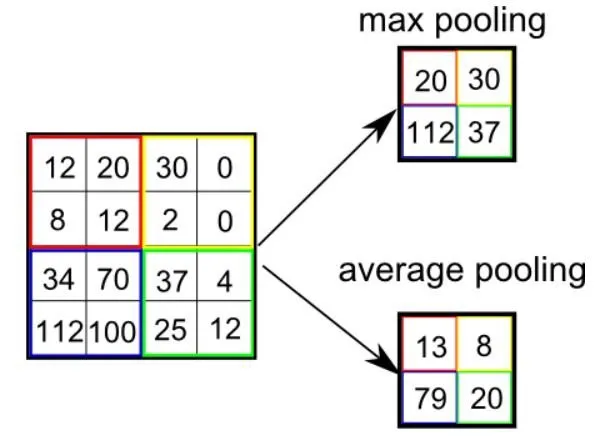


max pooling layer với size=(3,3), stride=1, padding=0

Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên .

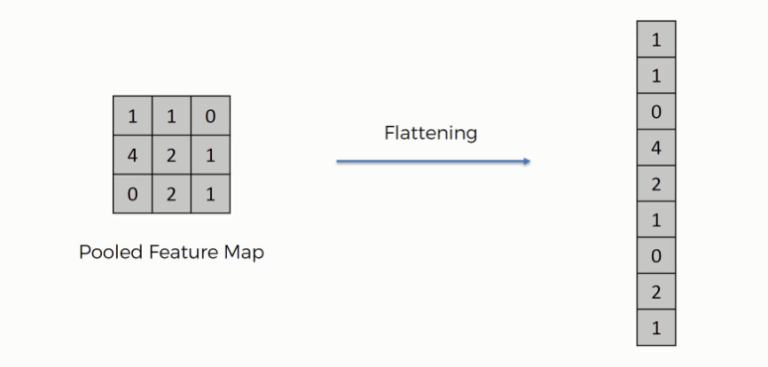


Có 2 loại pooling layer phổ biến là: max pooling và average pooling.



**Fully connected layer**

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,…) thì tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W)



Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

### Loss function

Huấn luyện mạng nơ ron nhân tạo giống với cách con người học tập. Chúng ta đưa mô hình dữ liệu, nó đưa ra dự đoán và chúng ta phản hồi xem dự đoán đó có chính xác hay không. Dựa trên phản hồi, mô hình có thể sửa những lỗi sai trước đó. Quá trình này lặp đi lặp lại cho đến khi mô hình đạt đến một độ chính xác nhất định.

Việc chỉ ra rằng mô hình đã đoán sai là vô cùng quan trọng để nó có thể học ra tri thức từ dữ liệu. Và đó là nguyên nhân loss function được thiết kế. Loss function sẽ chỉ ra mô hình đoán sai bao nhiêu so với giá trị thực tế.

### Optimizer

Optimizer là một thuật toán để xác định vị trí mà loss value đạt giá trị bé nhất.

### Fine-turning

Fine-tuning là lấy 1 pre-trained model, tận dụng 1 phần hoặc toàn bộ các layer, thêm/sửa/xóa 1 vài layer/nhánh để tạo ra 1 model mới. Thường các layer đầu của model được freeze (đóng băng) lại - tức weight các layer này sẽ không bị thay đổi giá trị trong quá trình train. Lý do bởi các layer này đã có khả năng trích xuất thông tin mức trừu tượng thấp , khả năng này được học từ quá trình training trước đó. Ta freeze lại để tận dụng được khả năng này và giúp việc train diễn ra nhanh hơn (model chỉ phải update weight ở các layer cao). Có rất nhiều các Object detect model được xây dựng dựa trên các Classifier model. VD densenet,googlenet,vgg,mobilenet…

**Các lớp trong 1 model**

| Layer | Mô tả |
| --- | --- |
| Convolution layer | Thực hiện các tính toán, dùng các kernel để lấy các features có trong ảnh. |
| Nonlinear layer (Relu) | Chuẩn hóa các output đầu ra, đưa các giá trị vượt ngưỡng về các giá trị phù hợp |
| Pooling layer | Thường được sử dụng ngay sau lớp convolution, layer này giảm kích thước của ma trận xuống 1 nửa nhưng vẫn giữ được chiều sâu (không bị mất thông tin ) |
| Batch Normalization | Là layer để chuẩn hóa dữ liệu giữa 2 layer trong mạng neural |
| Drop Out | Ẩn đi một số neural có trong mạng, làm giảm tình trạng overfit của model |
| Global Average Pooling | Là lớp tổng hợp được tạo ra để thay thế lớp fully connected trong mạng CNN cổ điển. |
| Flatten | Định hình lại tensor để có hình dạng bằng với số lượng phần tử có trong tensor. |

### Các bước triển khai 1 model

**B1: Tiền xử lý, hậu xử lý, in/out của mạng**

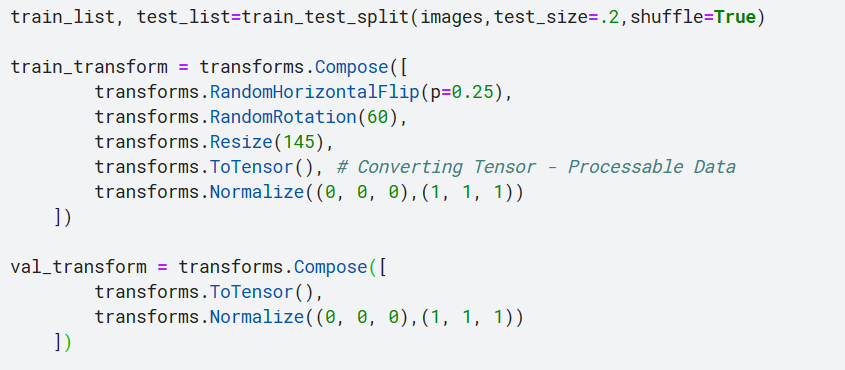
Đầu tiên chúng ta sẽ load các file ảnh trong dataset vào biến images dưới dạng list các đường dẫn.

****

Chia tập dữ liệu thành 2 tập train (80%) và tập test (20%)

Tập train chúng ta sẽ xử lý transform và random lật và xoay hình để giảm overfit, tăng thêm sự đa dạng do dữ liệu train. Sau đó resize, chuyển sang dạng tensor và nomalize.

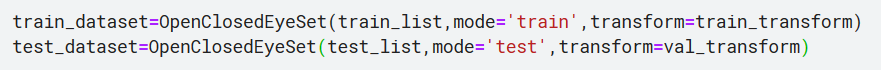
Tập validation chỉ chuyển sang tensor và nomalize.

****

**B2: Tạo dataset**

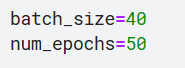
Tạo 2 tập train\_dataset và test\_dataset và dán nhãn cho từng ảnh

****

****

**B3: Tạo dataloader**

Batch size là số lượng mẫu dữ liệu trong một lần huấn luyện. Trong bài toán này batch size =40, nghĩa là 1 lần lặp ta sẽ cho ngẫu nhiên 40 bức hình nhắm mắt hoặc mở mắt chạy trong mạng. Tiếp theo cho tiếp 40 hình ngẫu nhiên, không lặp với các hình trước đó, vào mạng, đến khi nào không còn hình nào có thể cho vào nữa -> hoàn thành 1 epoch.

****

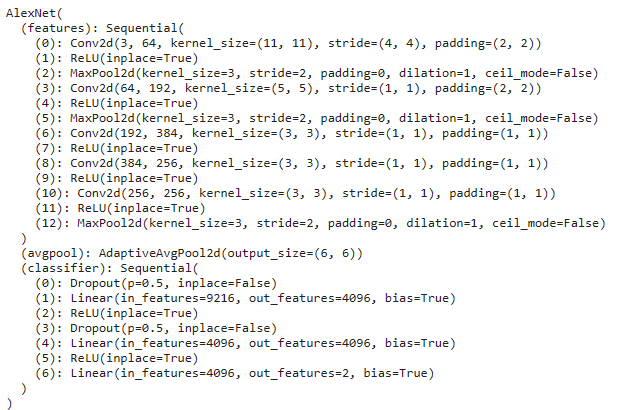
****

**B4: Xây dựng network và định nghĩa forward() trong network**

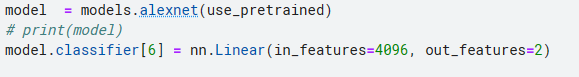
Trong bài toán này chúng ta sẽ không đi sâu vào việc thiết kế 1 model mà chỉ tận dụng các model pre-train có sẵn và chỉnh sửa lại chúng để phù hợp với bài toán đã đặt ra.

Phương pháp sử dụng là fine-tuning.

Trong ví dụ dưới đây, chúng ta sử dụng model pre-train Alexnet









Đầu tiên load model về, sau đó chỉnh sửa out\_features trong layer classifier lớp linear cuối cùng. Vì bài toán nhận dạng trạng thái ngủ gật chỉ có 2 features là nhắm mắt và mở mắt nên chúng ta sẽ chỉnh out\_features =2

**B5: Định nghĩa hàm loss**

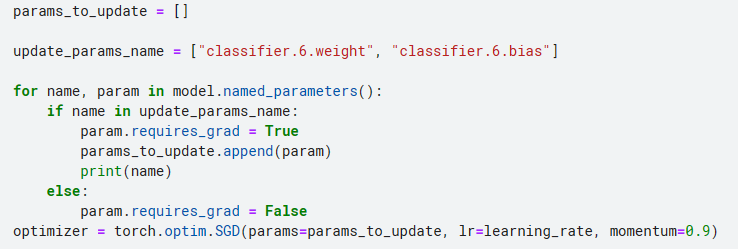
sử dụng Cross entropy loss

****

**B6: Thiết lập thuật toán tối ưu**

Bài toán này sử dụng thuật toán tối ưu Stochastic Gradient Descent

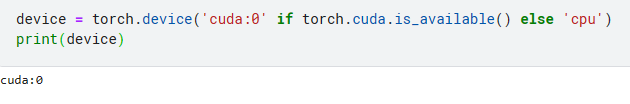
Ưu điểm của thuật toán này: thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà Gradient Descent không làm được.

****

param\_to\_update để điều chỉnh lại chỉ update parameter của layer cuối cùng, tắt update parameter của các layer khác.

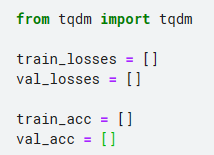
**B7: Thực hiện học và kiểm thử network**

Đầu tiên chúng ta chuyển model sang GPU (nếu có) để tốc độ train nhanh hơn

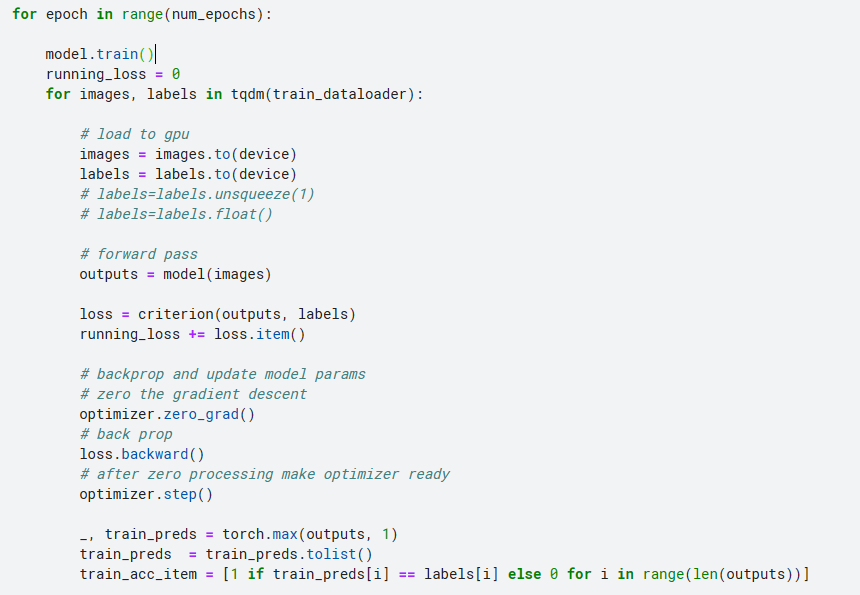
****

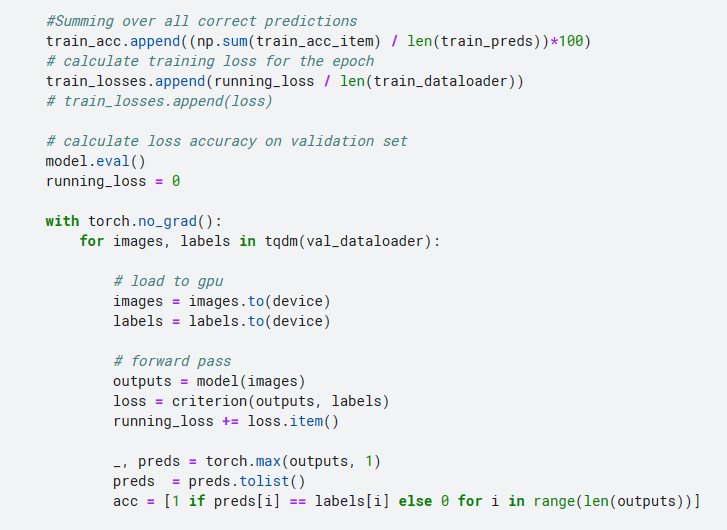
****

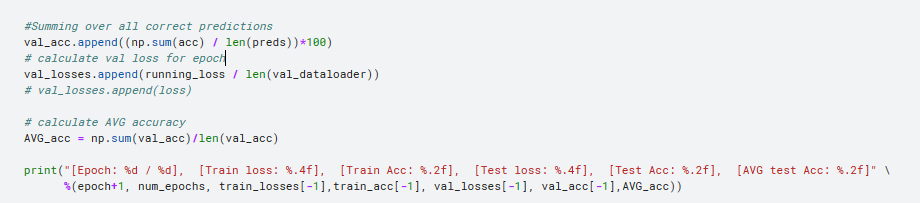
Sau đó khởi tạo các giá trị train và test của loss và accuracy



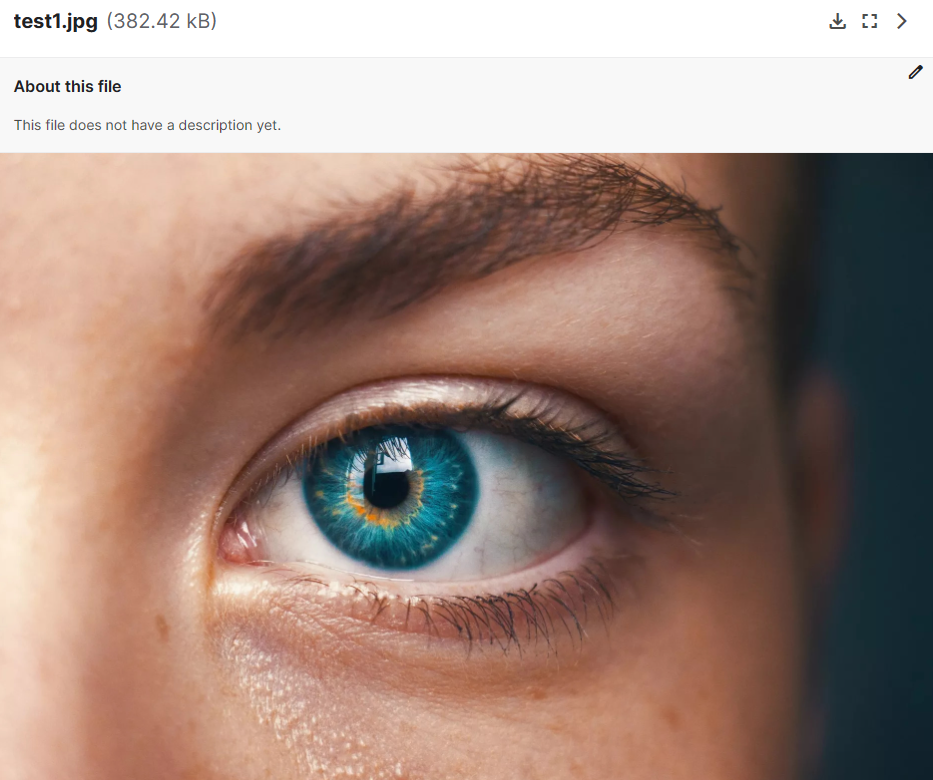
Tiến hành học và kiểm thử







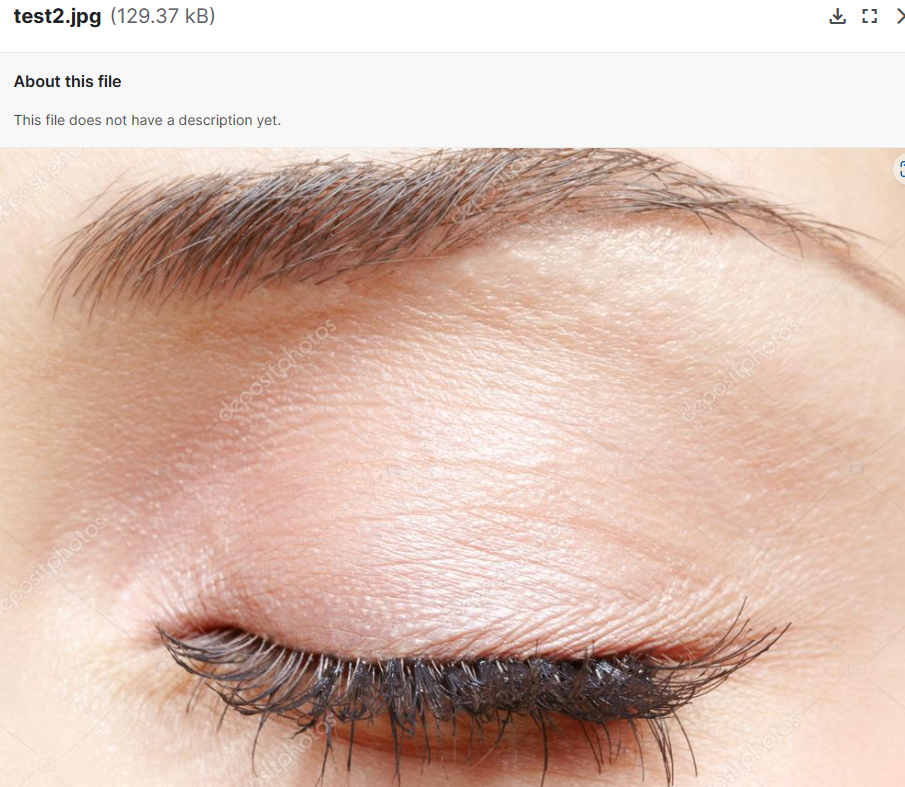
**B8: Dự đoán với data test**

****

****

****

Với hình ảnh test1.jpg, kết quả cho ra 0 tương ứng với label open





Tương tự với test2.jpg, kết quả trả về là 1 tương ứng với label close.

# CHƯƠNG 3: THIẾT KẾ HỆ THỐNG

### Yêu cầu hệ thống

Hệ thống nhận dạng trạng thái ngủ gật qua trạng thái mắt sử dụng mạng

nơ-ron tích chập cần thoả mãn những yêu cầu nhất định sau:

* Hệ thống nhận dạng trạng thái ngủ gật theo 3 phương thức: ảnh có sẵn,

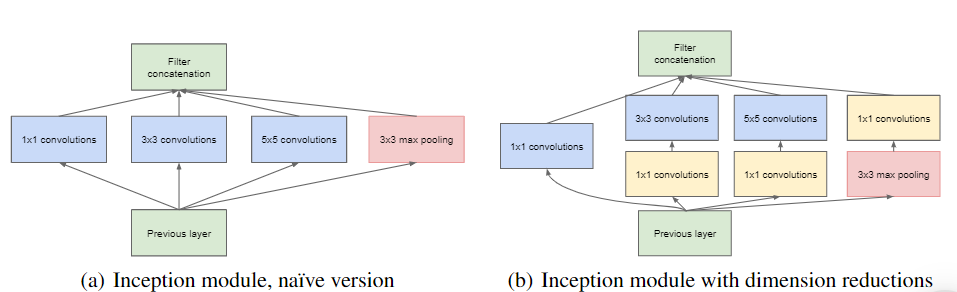
video có sẵn hoặc camera trong cự ly khoảng 1m trở lại.

* Hệ thống nhận dạng trạng thái ngủ gật tốt nhất trong điều kiện đủ ánh sáng, độ tương phản thích hợp, không bị tác động bởi các yếu tố gây nhiễu như vật cản, camera quá mờ
* Hệ thống sẽ nhận dạng được 2 trạng thái: mắt nhắm (ngủ) và mở mắt (bình thường).

### Mô hình hệ thống

1. **Kiến trúc CNN được sử dụng: GoogleNet**
   1. **Khối Inception**

Khối tích chập cơ bản trong mô hình GoogLeNet được gọi là Inception

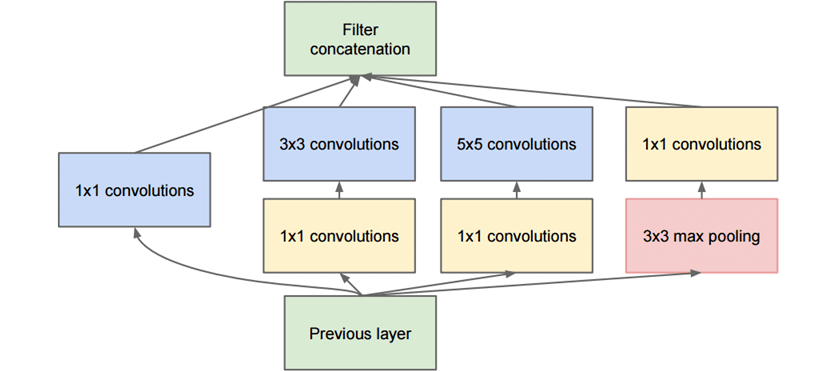
****

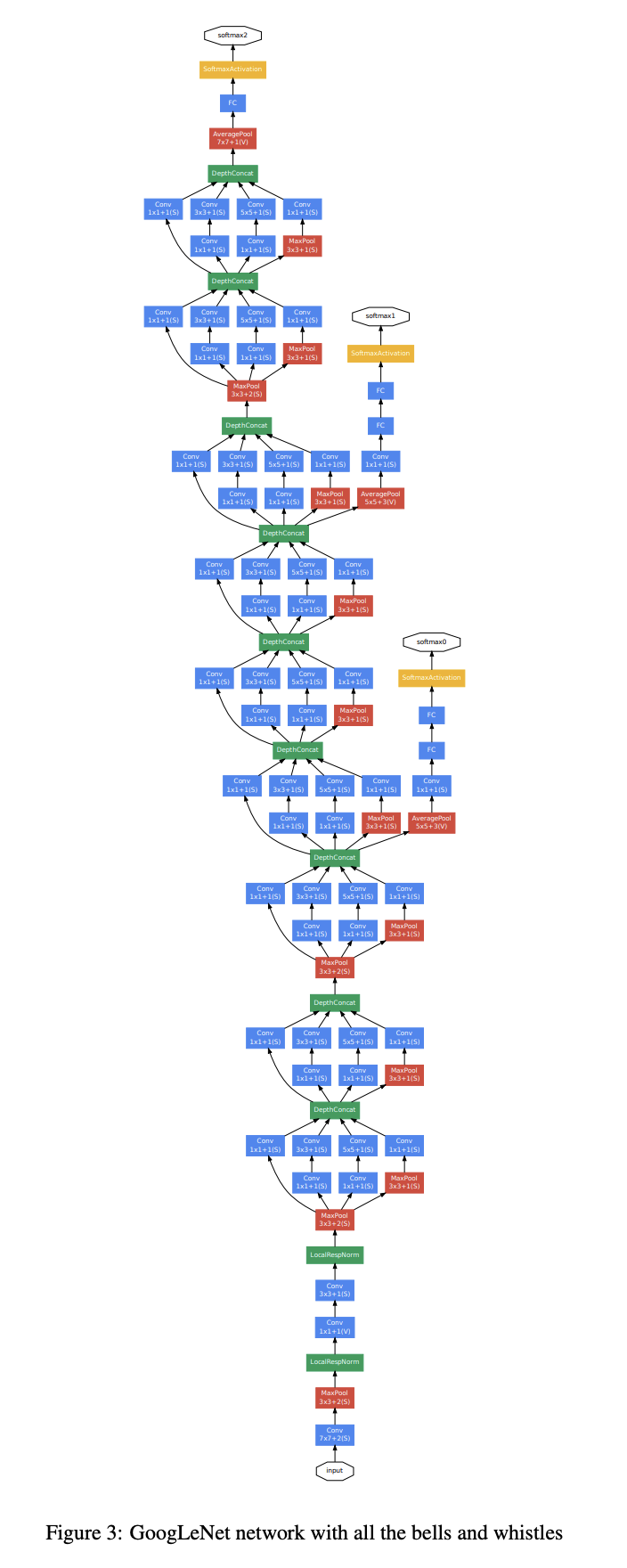
Như mô tả ở hình trên, khối inception bao gồm bốn nhánh song song với nhau. Ba nhánh đầu sử dụng các tầng tích chập với kích thước cửa sổ trượt lần lượt là 1×1 , 3×3 , và 5×5 để trích xuất thông tin từ các vùng không gian có kích thước khác nhau. Hai nhánh giữa thực hiện phép tích chập 1×1 trên dữ liệu đầu vào để giảm số kênh đầu vào, từ đó giảm độ phức tạp của mô hình. Nhánh thứ tư sử dụng một tầng gộp cực đại kích thước 3×3 , theo sau là một tầng tích chập 1×1 để thay đổi số lượng kênh. Cả bốn nhánh sử dụng phần đệm phù hợp để đầu vào và đầu ra của khối có cùng chiều cao và chiều rộng. Cuối cùng, các đầu ra của mỗi nhánh sẽ được nối lại theo chiều kênh để tạo thành đầu ra của cả khối. Các tham số thường được tinh chỉnh của khối Inception là số lượng kênh đầu ra mỗi tầng.

* 1. **Mô hình GoogLeNet**

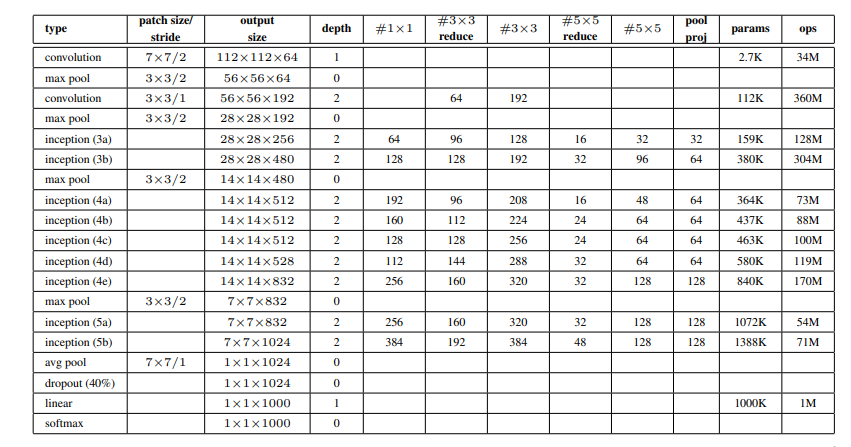
Mô hình GoogLeNet sử dụng tổng cộng 9 khối inception và tầng gộp trung bình toàn cục xếp chồng lên nhau. Phép gộp cực đại giữa các khối inception có tác dụng làm giảm kích thước chiều. Phần đầu tiên của GoogleNet giống AlexNet và LeNet, có các khối xếp chồng lên nhau kế thừa từ thiết kế của VGG và phép gộp trung bình toàn cục giúp tránh phải sử dụng nhiều tầng kết nối đầy đủ liên tiếp ở cuối.

Khối inception là một khối các layer kết nối với nhau như hình bên dưới



****

Chi tiết các layer GoogleNet



Đầu vào của mạng GoogleNet là hình ảnh có kích thước 224x224x3

Qua lớp convolution đầu tiên, in\_channels =3 tương ứng với 3 kênh màu của ảnh đầu vào RGB, out\_channels = 64 là số lượng kênh được tạo bởi convolution, kernel\_size=(7, 7), stride=(2, 2) thu được ma trận kích thước 112x112x64

Sau đó qua lớp Max pooling kernel\_size=3, stride=2 thu được ma trận kích thước 56x56x64

Lớp convolution tiếp theo,in\_channels =64, out\_channels = 64, kernel\_size=(3, 3), stride=(1, 1) thu được ma trận kích thước 56x56x64.

Lớp convolution tiếp theo,in\_channels =64, out\_channels = 192, kernel\_size=(3, 3), stride=(1, 1) thu được ma trận kích thước 56x56x192.

Sau đó qua lớp Max pooling kernel\_size=3, stride=2 thu được ma trận kích thước 28x28x192

Tiếp theo đưa qua khối Inception như trong mô hình, thu được ma trận kích thước 28x28x256

Tiếp tục đưa qua 1 khối Inception, thu được ma trận 28x28x480

….

Lớp avg pool giảm chiều cao và chiều rộng đầu vào xuống 1x1.

Qua lớp Dropout với hệ số 0.2 để giảm bớt 20% số node không cần thiết, giúp

tránh overfitting, thu được ma trận 1x1x1024 (Kỹ thuật dropout hoạt động bằng cách giảm ngẫu nhiên số lượng neural thần kinh kết nối với nhau trong một mạng lưới thần kinh. Ở mỗi bước đào tạo, mỗi neural thần kinh đều có khả năng bị loại bỏ, hay đúng hơn là bỏ qua sự đóng góp được đối chiếu từ các neural thần kinh được kết nối.)

Sau đó, đưa qua lớp Linear bao gồm 1000 đơn vị ẩn, tương ứng với 1000 lớp có trong tập dữ liệu Imagenet.

Lớp cuối cùng là lớp softmax; Lớp này sử dụng hàm SoftMax, một activation function được sử dụng để lấy phân bố xác suất của một tập hợp các số trong một vector đầu vào.

1. **Sơ đồ khối hệ thống**

Hình ảnh đầu vào được lấy từ camera hoặc từ hình ảnh (hoặc video) được

lưu sẵn trên thiết bị.

Qua quá trình tiền xử lý các video sẽ được cắt ra thành những khung ảnh tĩnh. Sau đó, các hình ảnh sẽ được chuyển thành ảnh xám và cắt con mắt ra nhờ 68 điểm facial landmark trên khuôn mặt.

Qua quá trình này, 2 mắt được phát hiện sẽ được chuyển thành ma trận 3 chiều gồm chiều dài, chiều rộng và chiều sâu, cùng với đó là chuyển đổi độ phân giải cho phù hợp với mô hình mạng CNN và chuyển đổi ảnh xám thành ảnh nhị phân. Sau đó, model sẽ trích xuất các đặc trưng của mắt và đưa các đặc trưng đó qua mạng CNN để phân loại.



Sơ đồ khối hệ thống

Dưới đây là ví dụ về 68 facial landmark



Output sẽ trả về tọa độ các điểm trên khuôn mặt, chúng ta chỉ cần trích xuất chúng. Ví dụ muốn lấy tọa độ mắt trái, ta chỉ cần lấy tọa độ của 4 điểm 37,39,40,42

1. **Tập dữ liệu**

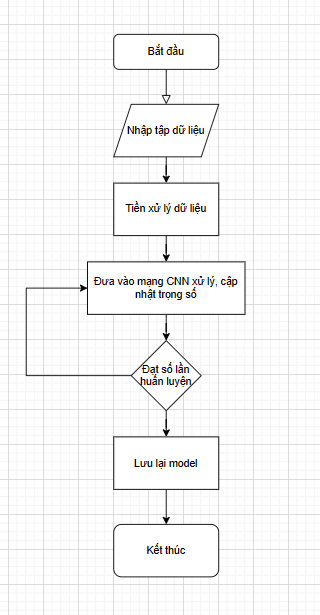
Tập dữ liệu được sử dụng bao gồm 726 hình ảnh mắt nhắm và 726 hình ảnh mắt mở.

Tập dữ liệu được lấy từ Kaggle của tác giả DHEERAJ PERUMANDLA

1. **Quá trình huấn luyện và đánh giá kết quả**

Đầu tiên ta nhập dữ liệu đầu vào. Tăng cường dữ liệu đầu vào và tăng cường dữ liệu bằng các phương pháp xoay, zoom ảnh, lật, v.v.

Dùng mạng CNN với kiến trúc mạng được cập nhật từ file để xử lý, cập nhật các trọng số theo từng epoch và lưu lại các thông số loss, accuracy theo mỗi epoch



# CHƯƠNG 4: KẾT LUẬN VÀ KẾT QUẢ CHI TIẾT

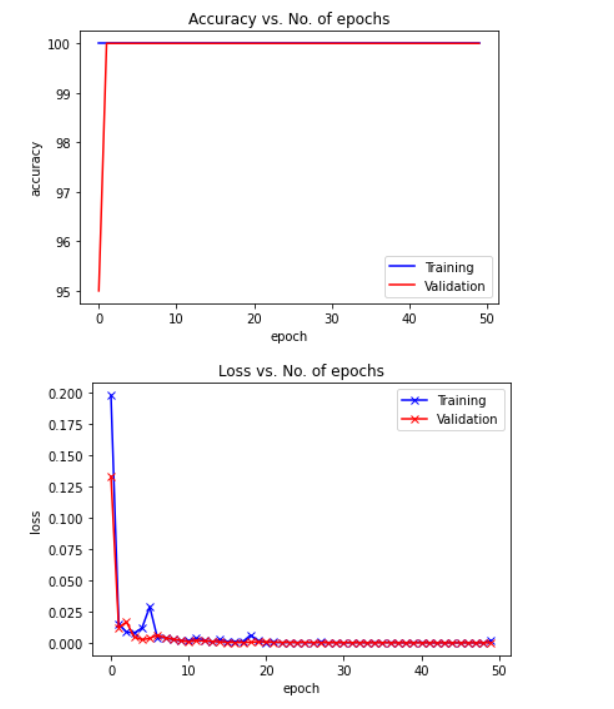
### Kết quả huấn luyện và kiểm tra

**Kích thước model: 21.57MB**

**Thời gian huấn luyện: 520 giây (GPU Tesla P100)**

Qua các lần thử nghiệm với nhiều loại kiến trúc khác nhau, người thực hiện đã chọn và sử dụng một thuật toán có tên GoogleNet với 50 epoch, 1 batch bao gồm 40 hình ảnh. Kết quả cho ra độ chính xác ~100%, cao hơn so tham khảo là 94%.

Người thực hiện cũng đo thời gian xử lý ảnh của model. Một tấm hình kích thước 145x145 px khi qua model xử lý trung bình mất 26.5 mili giây (CPU Intel core I7 10750H)



Drop out = 0.2

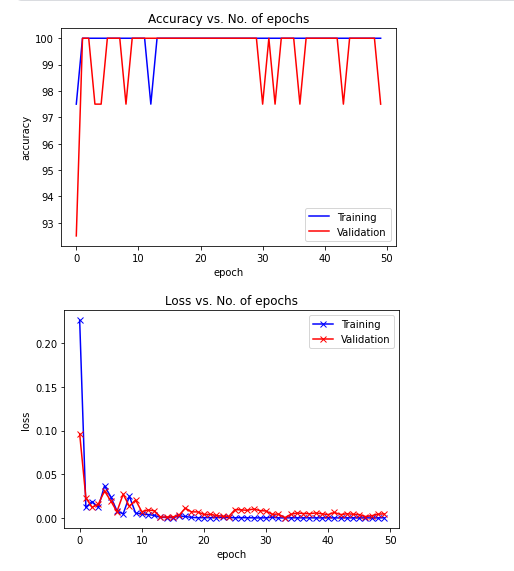
Mô hình hội tụ rất nhanh, qua 10 epoch mô hình đã cho độ chính xác gần như tuyệt đối

Optimizer được sử dụng khi train là Stochastic Gradient Descent (SGD).

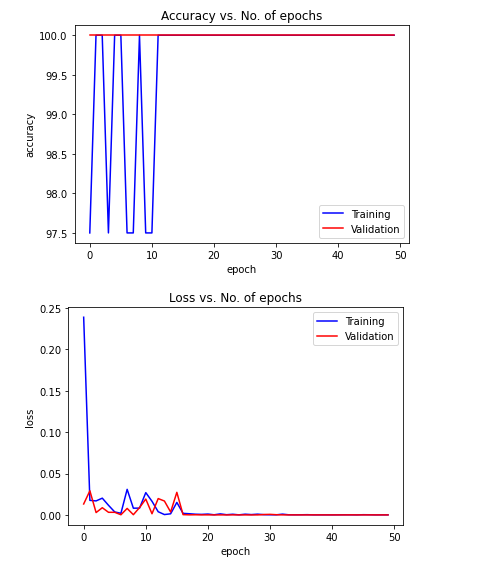
Stochastic là 1 biến thể của Gradient Descent . Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần. Nhìn vào 1 mặt , SGD sẽ làm giảm đi tốc độ của 1 epoch. Tuy nhiên nhìn theo 1 hướng khác,SGD sẽ hội tụ rất nhanh chỉ sau vài epoch. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.

Loss function được sử dụng là Cross Entropy Loss. Cross entropy Loss được sử dụng để học phân bố xác suất của dữ liệu. Trong khi các hàm khác phạt các giá trị sai, Cross-entropy Loss phạt mô hình dựa trên cả tính đúng và độ chắc chắn (tự tin) của dự đoán. Hàm này phạt cả các dự đoán sai lẫn các dự đoán đúng nhưng độ chắc chắn thấp.

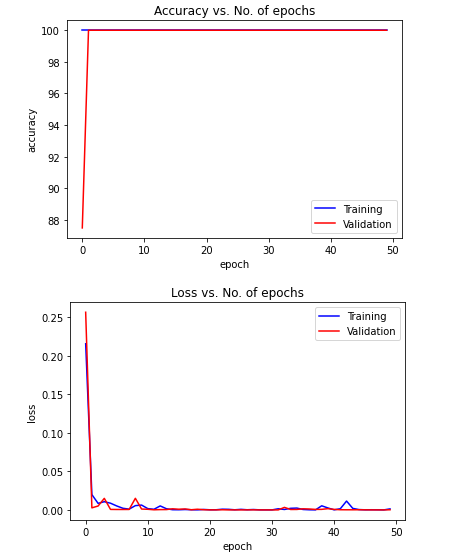
Kết quả thử nghiệm khi chỉnh các thông số drop out khác nhau



Drop out = 0.5



Drop out = 0.8



Drop out = 0

**Kết luận:** model cho kết quả rất tốt, không bị overfit hay under fit nên khi chỉnh drop out kết quả cho ra không thay đổi nhiều

### So sánh với các model khác

Các thông số được thiết lập giống với GoogleNet

Learning rate: 0.01

Loss Function được sử dụng: Cross Entropy Loss

Thuật toán tối ưu được sử dụng: Stochastic gradient descent (SGD)

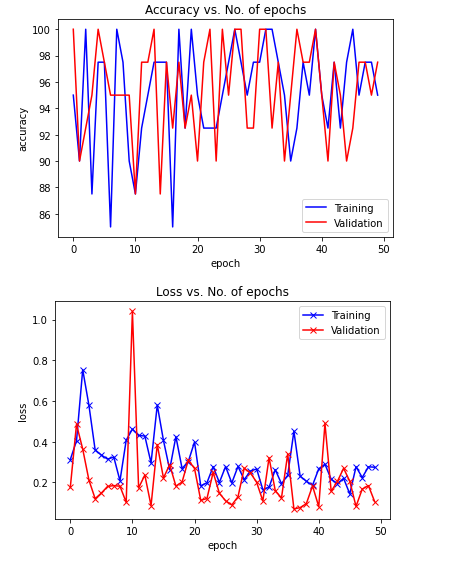
Model được train bằng GPU Tesla P100 trên Kaggle.com

Thời gian xử lý ảnh được đo trên máy tính người thực hiện với CPU Intel Core i7 10700H

* 1. **AlexNet**

**Kích thước model: 217.49MB**

**Thời gian trung bình xử lý 1 ảnh: 16.8 mili giây**

**Thời gian huấn luyện: 431 giây **

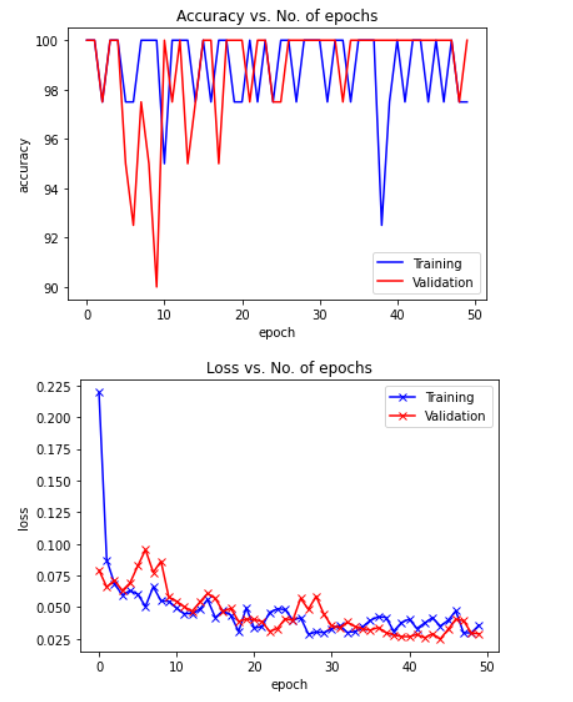
Model cho độ chính xác khoảng 98%, tốc độ nhận dạng rất nhanh nhưng model khá nặng (gần gấp 10 lần so với GoogleNet)

* 1. **MobileNet V3 Large**

**Kích thước model: 16.23MB**

**Thời gian trung bình xử lý 1 ảnh: 19.4 mili giây**

**Thời gian huấn luyện: 438 giây**

****

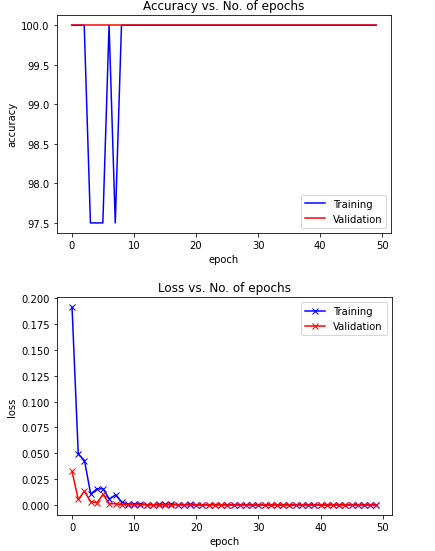
Model cho độ chính xác ~98%, tốc độ xử lý cũng rất nhanh và model rất nhẹ.

* 1. **Densenet201**

**Kích thước model: 70.34MB**

**Thời gian trung bình xử lý 1 ảnh: 82.2 mili giây**

**Thời gian huấn luyện: 724 giây**

****

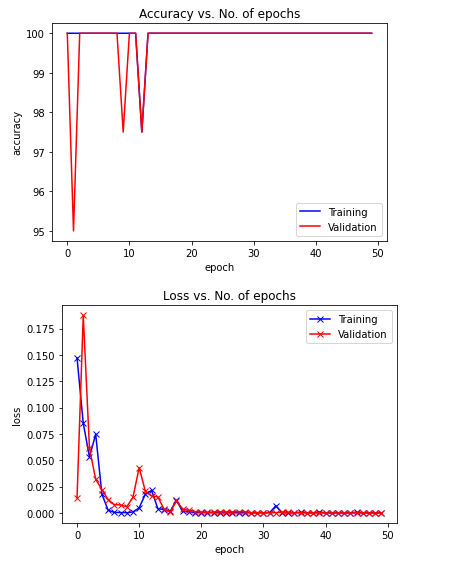
Model cho độ chính xác cao ~100%. Nhưng tốc độ nhận dạng khá chậm

* 1. **ResNet**

**Kích thước model: 476.95MB**

**Thời gian trung bình xử lý 1 ảnh: 26.9 mili giây**

**Thời gian huấn luyện: 1078 giây**

****

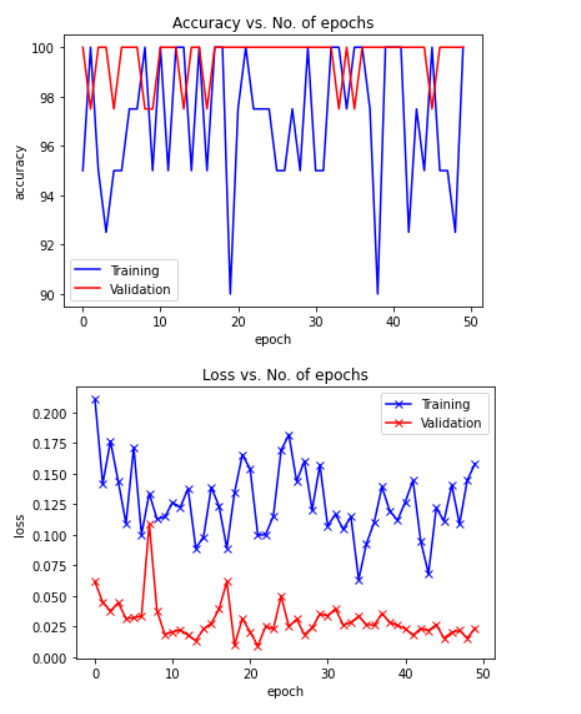
Model cho độ chính xác cao ~99% nhưng lại rất nặng, thời gian huấn luyện gấp đôi so với các model khác

* 1. **VGG16**

**Kích thước model: 512.2MB**

**Thời gian trung bình xử lý 1 ảnh: 94.9 mili giây**

**Thời gian huấn luyện: 511 giây**

****

Model cho độ chính xác 97%, tốc độ nhận dạng của model chậm và model khá nặng

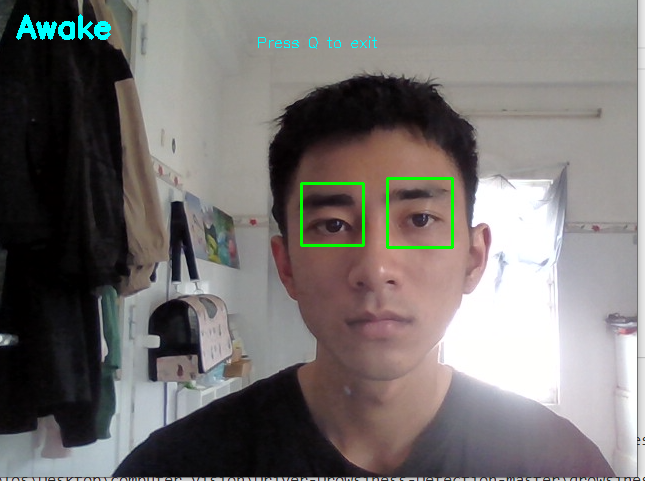
### Ứng dụng

Nghiên cứu nào có giá trị cũng thể hiện trên ứng dụng nhất định trong thực tế. Vì vậy, người thực hiện đã đưa mô hình nhận dạng khuôn mặt này vào một mô hình ứng dụng trên máy tính với sự hỗ trợ thiết kế giao diện app của bộ công cụ GUI Tkinter

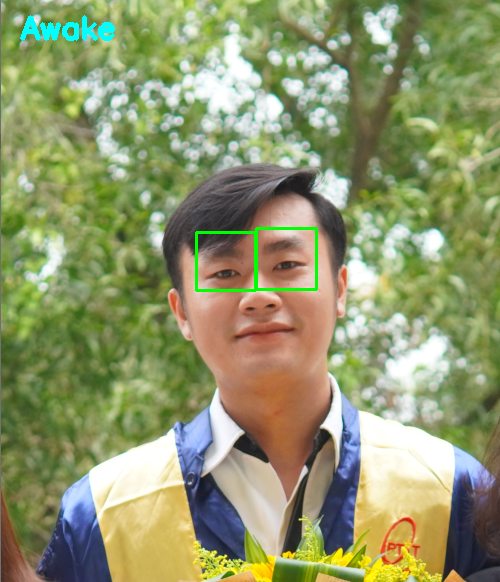


Giao diện ứng dụng nhận dạng trạng thái ngủ gật

Cách dùng ứng dụng khá đơn giản. Khi người dùng ấn vào nhận dạng dùng camera, ứng dụng sẽ mở webcam lên và tiến hành nhận dạng. Nếu nhắm mắt lâu hơn 6 giây ứng dụng sẽ phát âm thanh cảnh báo đến người sử dụng.



Khi người dùng chọn nhận dạng hình ảnh, ứng dụng sẽ cho ta chọn file và hiện trạng thái ở góc màn hình



# Tài liệu tham khảo

1. Eneszzeybek - Driver Drowsiness Detection

<https://github.com/eneszzeybek/Driver-Drowsiness-Detection>

Tham khảo cách triển khai và train model bằng pytorch

1. Nguyen Huu Trinh

https://www.youtube.com/watch?v=9bc9Wc8BYrc&list=PLVnyU2MngPda-pqImB\_n5iNPdhnBngMOn&ab\_channel=ManabiTV

Tham khảo cách triển khai và train model bằng pytorch

1. https://pytorch.org/

Tham khảo các model và cách triển khai.

1. Kaggle.com - Drowsiness dataset

https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset

1. Trung Thành Nguyễn - Fine tuning pre-trained model trong pytorch và áp dụng vào Visual Saliency Prediction
2. Trituenhantao.io - Hàm loss trong Pytorch
3. Trần Trung Trực - Các phương pháp tránh Overfitting
4. Trần Trung Trực - Optimizer- Hiểu sâu về các thuật toán tối ưu ( GD,SGD,Adam,..)
5. Nttuan - Deep Learning cơ bản
6. Nguyễn Đình Quý - <https://ndquy.github.io/>
7. Pawangfg - Understanding GoogLeNet Model – CNN Architecture
8. Chung Pham Van - [Deep Learning] Tìm hiểu về mạng tích chập (CNN)