**HƯỚNG DẪN SỬ DỤNG**

GITHUB: [Dat0309/drowsy-detection: Đề tài kết thúc học phần môn Trí tuệ nhân tạo lớp CTK43 (github.com)](https://github.com/Dat0309/drowsy-detection)

**Hướng dẫn sử dụng:**

Đầu tiên, chúng ta mở ứng dụng.

Sau đó chọn nút Show.

Graphical user interface, text, application

Description automatically generated

Sau khi nhấn nút Show. Màn hình giám sát khuôn mặt sẽ hiện lên màn hình chính của ứng dụng.

Graphical user interface, application

Description automatically generated

Lúc phát hiện buồn ngủ, ứng dụng sẽ đổ chuông đánh thức người dùng dậy.

Graphical user interface, application

Description automatically generated

**Mã nguồn:**

training\_model.py

import imp

from keras import callbacks

from keras.models import Sequential

from keras.callbacks import ModelCheckpoint

from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

import os

# Lấy đường dần hình ảnh trong tập dữ liệu

def plot\_img(dir, top = 10):

    all\_item\_in\_dir = os.listdir(dir)

    for file in all\_item\_in\_dir:

        item\_file = [os.path.join(dir, file)][:5]

data\_path = 'dataset/train'

directories = ['/Closed', '/Open']

BATCH\_SIZE = 128

CLASSES = 4

EPOCHS = 20

for j in directories:

    plot\_img(data\_path + j)

# ImageDataGenerator tăng cường dữ liệu training model (Lật, cắt ngẫu nhiên, zoom, thay đổi tỷ lệ)

train\_data = ImageDataGenerator(

    horizontal\_flip=True,

    rescale=1./255,

    zoom\_range=0.2,

    validation\_split=0.1

    )

test\_data = ImageDataGenerator(rescale=1./255)

train\_data\_path = 'dataset/train'

test\_data\_path = 'dataset/test'

train\_set = train\_data.flow\_from\_directory(

    train\_data\_path,

    target\_size=(24, 24),

    batch\_size=BATCH\_SIZE,

    color\_mode='grayscale',

    class\_mode='categorical'

    )

test\_set = test\_data.flow\_from\_directory(

    test\_data\_path,

    target\_size=(24, 24),

    batch\_size=BATCH\_SIZE,

    color\_mode='grayscale',

    class\_mode='categorical'

    )

# Layer Model

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', input\_shape = (24, 24, 1), activation='relu'))

model.add(MaxPooling2D(pool\_size=(1, 1)))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(1, 1)))

model.add(Conv2D(64,(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(1, 1)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

print(model.summary())

# compile model

model.compile(

    loss='categorical\_crossentropy',

    optimizer= 'adam',

    metrics=['accuracy']

    )

model\_path = 'eye\_detection.h5'

checkpoint = ModelCheckpoint(

    model\_path,

    monitor='val\_accuracy',

    verbose=1,

    save\_best\_only=True,

    mode='max')

callbacks = [checkpoint]

training\_steps = train\_set.n//train\_set.batch\_size

val\_steps = test\_set.n//test\_set.batch\_size

history = model.fit\_generator(

    train\_set, epochs=EPOCHS,

    steps\_per\_epoch=training\_steps,

    validation\_data=test\_set,

    validation\_steps=val\_steps,

    callbacks=callbacks

    )

plt.figure(figsize=(20, 10))

plt.subplot(1, 2, 1)

plt.suptitle('Optimizera : Adam', fontsize = 10)

plt.ylabel('Loss', fontsize = 16)

plt.plot(history.history['loss'], label = 'Training loss')

plt.plot(history.history['val\_loss'], label = 'Validation Loss')

plt.legend(loc = 'upper right')

plt.subplot(1, 2, 2)

plt.ylabel('Accuracy', fontsize=16)

plt.plot(history.history['accuracy'], label=' Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Acc')

plt.legend(loc='lower right')

plt.savefig("plot.png")

plt.show()

output\_window.py

import re

from numpy.lib.npyio import save

from main\_window import Ui\_MainWindow

from PyQt5 import QtCore, QtGui, QtWidgets

import sys, icons

from PySide2 import \*

from keras.models import load\_model

import numpy as np

import cv2 as cv

import os

from pygame import mixer

class MainWindow(QtWidgets.QMainWindow):

    def \_\_init\_\_(self):

        QtWidgets.QMainWindow.\_\_init\_\_(self)

        self.ui = Ui\_MainWindow()

        self.ui.setupUi(self)

        self.setWindowIcon(QtGui.QIcon(":/icon/icons/mask.svg"))

        self.setWindowTitle("PHÁT HIỆN BUỒN NGỦ")

        QtWidgets.QSizeGrip(self.ui.size\_grip)

        self.ui.mainBody.setGraphicsEffect(QtWidgets.QGraphicsDropShadowEffect(blurRadius=25, xOffset=0, yOffset=0))

        self.ui.menuBtn.setGraphicsEffect(QtWidgets.QGraphicsDropShadowEffect(blurRadius=25, xOffset=5, yOffset=5))

        self.ui.centralwidget.setGraphicsEffect(QtWidgets.QGraphicsDropShadowEffect(blurRadius=25, xOffset=0, yOffset=0))

        self.ui.slideMenu.setGraphicsEffect(QtWidgets.QGraphicsDropShadowEffect(blurRadius=25, xOffset=0, yOffset=0))

        self.show()

        self.ui.widget\_face\_rec.hide()

        self.ui.widget\_main.show()

        self.logic = 0

        self.value = 1

        self.score = 0

        '''

        Event Function

        '''

        self.ui.show\_main.clicked.connect(self.show\_video)

        self.ui.cap\_main.clicked.connect(self.cap\_video)

        self.ui.menuBtn.clicked.connect(self.showMenubar)

        self.ui.hideBtn.clicked.connect(self.showMinimized)

        self.ui.maxBtn.clicked.connect(self.restore\_or\_maximized\_window)

        self.ui.xBtn.clicked.connect(self.close)

        self.ui.exitBtn.clicked.connect(self.close)

        self.ui.showBtn.clicked.connect(self.showButton)

        # self.ui.homeBtn.clicked.connect(self.homeButton)

        # self.ui.capBtn.clicked.connect(self.fileOpen)

        # self.ui.subMenu1.clicked.connect(self.face\_rec)

        # Move windows with mouse

        def moveWindow(e):

            if self.isMaximized() == False:

                if e.buttons() == QtCore.Qt.LeftButton:

                    self.move(self.pos() + e.globalPos() - self.clickPosition)

                    self.clickPosition = e.globalPos()

                    e.accept()

        self.ui.headerFrame.mouseMoveEvent = moveWindow

    def mousePressEvent(self, event):

        self.clickPosition = event.globalPos()

    def restore\_or\_maximized\_window(self):

        if self.isMaximized():

            self.showNormal()

            self.ui.maxBtn.setIcon(QtGui.QIcon(u":/icon/icons/maximize.svg"))

        else:

            self.showMaximized()

            self.ui.maxBtn.setIcon(QtGui.QIcon(u":/icon/icons/minimize.svg"))

    def showMenubar(self):

        width = self.ui.slideMenu.width()

        if width == 0:

            newWidth = 248

            self.ui.menuBtn.setIcon(QtGui.QIcon(u":/icon/icons/chevron-left.svg"))

        else:

            newWidth = 0

            self.ui.menuBtn.setIcon(QtGui.QIcon(u":/icon/icons/Search-write.svg"))

        self.animation = QtCore.QPropertyAnimation(self.ui.slideMenu, b"maximumWidth")

        self.animation.setDuration(250)

        self.animation.setStartValue(width)

        self.animation.setEndValue(newWidth)

        self.animation.setEasingCurve(QtCore.QEasingCurve.InOutQuart)

        self.animation.start()

    '''

    define all function.

    '''

    def showButton(self):

        # self.ui.widget.hide()

        self.ui.widget\_face\_rec.hide()

        self.ui.widget\_main.show()

    def homeButton(self):

        self.ui.widget\_main.hide()

        self.ui.widget.show()

    def displayPictureIn(self, img, window = 1):

        qformat = QtGui.QImage.Format\_Indexed8

        if len(img.shape) == 3:

            if(img.shape[2]) == 4:

                qformat = QtGui.QImage.Format\_RGBA888

            else:

                qformat = QtGui.QImage.Format\_RGB888

        img = QtGui.QImage(img, img.shape[1], img.shape[0], qformat)

        img = img.rgbSwapped()

        self.ui.lb\_imgIn.setPixmap(QtGui.QPixmap.fromImage(img))

        self.ui.lb\_imgIn.setAlignment(QtCore.Qt.AlignHCenter | QtCore.Qt.AlignVCenter)

    def displayPictureOut(self, img, window = 1):

        qformat = QtGui.QImage.Format\_Indexed8

        if len(img.shape) == 3:

            if(img.shape[2]) == 4:

                qformat = QtGui.QImage.Format\_RGBA888

            else:

                qformat = QtGui.QImage.Format\_RGB888

        img = QtGui.QImage(img, img.shape[1], img.shape[0], qformat)

        img = img.rgbSwapped()

        self.ui.lb\_imgOut.setPixmap(QtGui.QPixmap.fromImage(img))

        self.ui.lb\_imgOut.setAlignment(QtCore.Qt.AlignHCenter | QtCore.Qt.AlignVCenter)

    '''

        Phát hiện buồn ngủ qua hình ảnh

    '''

    def drowsy\_detect(self, picture):

        frame = cv.imread(picture)

    '''

        # Chức năng lưu ảnh khuôn mặt

    '''

    def savePic(self, img, box, width=224, height=224):

        self.value += 1

        x, y, w, h = box

        imgCrop = img[y:h, x:w]

        imgCrop = cv.resize(imgCrop, (width, height))

        cv.imwrite(self.new\_path + '%s.png'%(self.value), imgCrop)

    '''

        # Chức năng phát Video/ Video stream trên màn hình ứng dụng

    '''

    def show\_video(self):

        # File Âm thanh cảnh báo

        mixer.init()

        sound = mixer.Sound('alarm.wav')

        # Đọc bộ nhận dạng haar cascade của mắt trái, mắt phải và phía trước mặt

        face = cv.CascadeClassifier('haar cascade files\haarcascade\_frontalface\_alt.xml')

        leye = cv.CascadeClassifier('haar cascade files\haarcascade\_lefteye\_2splits.xml')

        reye = cv.CascadeClassifier('haar cascade files\haarcascade\_righteye\_2splits.xml')

        label = ['NHẮM MẮT', 'MỞ MẮT']

        model = load\_model('eye\_detection.h5')

        camCapture = cv.VideoCapture(0)

        font = cv.FONT\_HERSHEY\_COMPLEX

        COUNT = 0

        SCORE = 0

        THICK = 2

        RIGHTPRED = [99]

        LEFTPRED = [99]

        while(True):

            ret, frame = camCapture.read()

            height, width = frame.shape[:2]

            gray = cv.cvtColor(frame, cv.COLOR\_BGR2GRAY)

            '''

                Để phát hiện khuôn mặt trong hình ảnh, trước tiên chúng ta cần phải chuyển đổi hình ảnh thành thang màu xám (gray)

                vì thang màu xám không phức tạp như RGB, OpenCV có thể dễ dàng thao tác trên hình ảnh màu xám.

                Sử dụng phân loại Haar cascade để phát hiện các đặc điểm trên khuôn mặt như mắt trái, mắt phải.

                DetectMultilScale() trả về  một mảng phát hiện với toạ độ x,y và chiều cao w, rộng h của khuôn mặt.

                Ta sẽ dùng các toạ độ để vẽ hình chữ nhật xác định khuôn mặt.

            '''

            faces = face.detectMultiScale(gray, minNeighbors = 5, scaleFactor= 1.1, minSize=(25, 25))

            left\_eye = leye.detectMultiScale(gray)

            right\_eye = reye.detectMultiScale(gray)

            # vẽ hình chữ nhật ở vị trí khuôn mặt

            for(x, y, w, h) in faces:

                cv.rectangle(frame, (x, y), (x +w, y +h), (100, 100, 100), 1)

            for (x, y, w, h) in right\_eye:

                r\_eye = frame[y:y+h, x:x+w]

                COUNT = COUNT + 1

                r\_eye = cv.cvtColor(r\_eye, cv.COLOR\_BGR2GRAY)

                r\_eye = cv.resize(r\_eye, (24, 24))

                r\_eye = r\_eye/255

                r\_eye = r\_eye.reshape(24, 24, -1)

                r\_eye = np.expand\_dims(r\_eye, axis=0)

                RPredict = model.predict(r\_eye)

                RIGHTPRED = np.argmax(RPredict, axis=1)

                if(RIGHTPRED[0] == 1):

                    label='MỞ MẮT'

                if(RIGHTPRED[0] == 0):

                    label='NHẮM MẮT'

                break

            for (x, y, w, h) in left\_eye:

                left\_eye = frame[y:y+h, x:x+w]

                COUNT = COUNT + 1

                left\_eye = cv.cvtColor(left\_eye, cv.COLOR\_BGR2GRAY)

                left\_eye = cv.resize(left\_eye, (24, 24))

                left\_eye = left\_eye/255

                left\_eye = left\_eye.reshape(24, 24, -1)

                left\_eye = np.expand\_dims(left\_eye, axis=0)

                LPredict = model.predict(left\_eye)

                LEFTPRED = np.argmax(LPredict, axis=1)

                if(LEFTPRED[0] == 1):

                    label='MỞ MẮT'

                if(LEFTPRED[0] == 0):

                    label = 'NHẮM MẮT'

                break

            if(RIGHTPRED[0] == 0 and LEFTPRED[0] == 0):

                SCORE += 1

                cv.putText(frame, 'CLOSED', (10, height - 20), font, 1,(255,0,0), 1, cv.LINE\_AA)

            else:

                SCORE-=1

                cv.putText(frame, 'OPEN',(10, height -20), font, 1,(0,255,0), 1, cv.LINE\_AA)

            if(SCORE < 0):

                SCORE = 0

            cv.putText(frame, 'Score:' + str(SCORE), (110, height - 20), font, 1,cv.LINE\_AA )

            if(SCORE > 9):

                try:

                    sound.play()

                    cv.imwrite('image.jpg', frame)

                except:

                    pass

                if(THICK < 10):

                    THICK+=2

                else:

                    THICK-=2

                    if(THICK<2):

                        THICK=2

                cv.rectangle(frame, (0,0), (width, height), (0, 0, 255), THICK)

            if ret == True:

                self.displayImage(frame, 1)

                if (self.logic==2):

                    self.logic = 1

                    break

                    # self.value = self.value + 1

                    # cv.imwrite('%s.png'%(self.value), frame)

                if cv.waitKey(1) & 0xFF == ord('q'):

                    break

            else:

                print("return not found")

        camCapture.release()

        cv.destroyAllWindows()

    '''

    Hiển thị Video lên Pixmap

    '''

    def displayImage(self, img, window = 1):

        qformat = QtGui.QImage.Format\_Indexed8

        if len(img.shape) == 3:

            if(img.shape[2]) == 4:

                qformat = QtGui.QImage.Format\_RGBA888

            else:

                qformat = QtGui.QImage.Format\_RGB888

        img = QtGui.QImage(img, img.shape[1], img.shape[0], qformat)

        img = img.rgbSwapped()

        #self.ui.label\_4.setScaledContents(True)

        self.ui.label\_4.setPixmap(QtGui.QPixmap.fromImage(img))

        self.ui.label\_4.setAlignment(QtCore.Qt.AlignHCenter | QtCore.Qt.AlignVCenter)

    def cap\_video(self):

        self.logic = 2

if \_\_name\_\_ == "\_\_main\_\_":

    app = QtWidgets.QApplication(sys.argv)

    window = MainWindow()

    sys.exit(app.exec\_())