A Project/Dissertation Review Report

on

DRIVER DROWSINESS DETECTION SYSTEM USING CONVOLUTIONAL NEURAL NETWORK

Submitted in partial fulfillment of the

requirement for the award of the degree of

BTECH IN COMPUTER SCIENCEENGINEERING

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of

Dr. JOHN A
Associate Professor

Submitted By

Name Of Student	Enrollment number
UTKARSH GOYAL	18021140054
ABHISHEK CHAUDHARY	18021140111

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"DRIVER DROWSINESS DETECTION SYSTEM"** in partial fulfillment of the requirements for the award of the **B.TECH**-submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of month, Year to Month and Year, under the supervision of **Dr. John A (Associate Professor)**, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

UTKARSH GOYAL 18SCSE1140006

Abhishek Chaudhary 18SCSE1140065

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. John A

(Associate Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of UTKARSH GOYAL |18SCSE1140006,ABHISHEK CHAUDHARY |18SCSE1140065has been held on ________________ and his/her work is recommended for the award of BACHELOR OF COMPUTER SCIENCE AND TECHNOLOGY.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: November, 2013

Place: Greater Noida

ABSTRACT

Driver fatigue is one of the major causes of accidents in the world. Detecting the drowsiness of the driver is one of the surest ways of measuring driver fatigue. In this project we aim to develop a prototype drowsiness detection system

This document is the research conducted and the project made in the field of computer engineering to develop a system for driver drowsiness detection to prevent accidents from happening because of driver fatigue and sleepiness. The report proposed the results and solutions on the limited implementation of the various techniques that are introduced in the project.

Whereas the implementation of the project give the real world idea of how the system works and what changes can be done in order to improve the utility of the overall system. Furthermore, the paper states the overview of the observations made by the authors in order to help further optimization in the mentioned field to achieve the utility at a better efficiency for a safer road.

This system works by monitoring the eyes of the driver and sounding an alarm when he/she is drowsy. The system so designed is a non-intrusive real-time monitoring system. The priority is on improving the safety of the driver without being obtrusive. In this project the eye blink of the driver is detected. If the drivers eyes remain closed for more than a certain period of time, the driver is said to be drowsy and an alarm is sounded.

The programming for this is done in OpenCV using the Haarcascade library for the detection of facial features. The aim of this project is to develop a prototype drowsiness detection system. The focus will be placed on designing a system that will accurately monitor the open or closed state of the driver's eyes in real- time.

TABLE OF CONTENT

1. INTRODUCTION
2. PROPOSED SOLUTION
    2.1 WHY OPENCV
    2.2 COMPUTER VISION
    2.3 CNN Module
    2.4 Facial Landmark
    2.5 Dlib
3. WORK PLAN LAYOUT
4. CONCLUSION

INTRODUCTION

Driver fatigue is a significant factor in a large number of vehicle accidents. Recent statistics estimate that annually 1,200 deaths and 76,000 injuries can be attributed to fatigue related crashes. Because of the hazard that drowsiness presents on the road, methods need to be developed for counteracting its affects.

The aim of this project is to develop a prototype drowsiness detection system. The focus will be placed on designing a system that will accurately monitor the open or closed state of the driver's eyes in real-time.
By monitoring the eyes, it is believed that the symptoms of driver fatigue can be detected early enough to avoid a car accident. Detection of fatigue involves the observation of eye movements and blink patterns in a sequence of images of a face.
This is where OpenCV came in. OpenCV is an open source computer vision library. It is designed for computational efficiency and with a strong focus on real time applications. It helps to build sophisticated vision applications quickly and easily. OpenCV satisfied the low processing power and high speed requirements of
our application.

We have used the Haartraining applications in OpenCV to detect the face and eyes. This creates a classifier given a set of positive and negative samples. The steps were as follows:-

- Gather a data set of face and eye. These should be stored in one or more directories indexed by a text file. A lot of high quality data is required for the classifier to work well.
- The utility application createsamples() is used to build a vector output file. Using this file we can repeat the training procedure. It extracts the positive samples from images before normalizing and resizing to specified width and height.
- The Viola Jones cascade decides whether or not the object in an image is similar to the training set. Any image that doesn't contain the object of interest can be turned into negative sample. So in order to learn any object it is required to take a sample of negative background image.
- Training of the image is done using boosting. In training we learn the group of classifiers one at a time. Each classifier in the group is a weak classifier. These weak classifiers are typically composed of a single variable decision tree called stumps. In training the decision stump learns its classification decisions from its data and also learns a weight for its vote from its accuracy on the data. Between training each classifier one by one, the data points are reweighted so that more attention is paid to the data points where errors were made. This process continues until the total error over the dataset arising from the combined weighted vote of the decision trees falls below a certain threshold

LITERATURE SURVEY

Driver Drowsiness Detection System and Techniques

According to the experts it has been observed that when the drivers do not take break they tend to run a high risk of becoming drowsy. Study shows that accidents occur due to sleepy drivers in need of a rest, which means that road accidents occurs more due to drowsiness rather than drink-driving. Attention assist can warn of inattentiveness and drowsiness in an extended speed range and notify drivers of their current state of fatigue and the driving time since the last break, offers adjustable sensitivity and, if a warning is emitted, indicates nearby service areas in the COMAND navigation system.

Implementation of the Driver Drowsiness Detection System

This paper is about making cars more intelligent and interactive which may notify or resist user under unacceptable conditions, they may provide critical information of real time situations to rescue or police or owner himself. Driver fatigue resulting from sleep disorders is an important factor in the increasing number of accidents on today's roads. In this paper, we describe a real-time safety prototype that controls the vehicle speed under driver fatigue. To advance a system to detect fatigue symptoms in drivers and control the speed of vehicle to avoid accidents is the purpose of such a mode. In this paper, we propose a driver drowsiness detection system in which sensor like eye blink sensor are used for detecting drowsiness of driver. If the driver is found to have sleep, buzzer will start buzzing and then turns the vehicle ignition off.

Detecting Driver Drowsiness Based on Sensors

Researchers have attempted to determine driver drowsiness using the following measures: vehicle-based measures; behavioural measures and physiological measures. A detailed review on these measures will provide insight on the present systems, issues associated with them and the enhancements that need to be done to make a robust system. This paper reviews the three measures as to the sensors used and discuss the advantages and limitations of each. The various ways through which drowsiness has been experimentally manipulated is also discussed. It is concluded that by designing a hybrid drowsiness detection system that combines non-intrusive physiological measures with other measures one would accurately determine the drowsiness level of a driver. A number of road accidents might then be avoided if an alert is sent to a driver that is deemed drowsy.

Why OpenCV

Specific
OpenCV was designed for image processing. Every function and data structure has been designed with an Image Processing application in mind. Meanwhile, Matlab, is quite generic. You can get almost everything in the world by means of toolboxes. It may be financial toolboxes or specialized DNA toolboxes.

Speedy
Matlab is just way too slow. Matlab itself was built upon Java. Also Java was built upon C. So when we run a Matlab program, our computer gets busy trying to interpret and compile all that complicated Matlab code.
Then it is turned into Java, and finally executes the code.

If we use C/C++, we don't waste all that time. We directly provide machine language code to the computer, and it gets executed. So ultimately we get more image processing, and not more interpreting.
After doing some real time image processing with both Matlab and OpenCV, we usually got very low speeds, a maximum of about 4-5 frames being processed per second with Matlab. With OpenCV however, we get actual real time processing at around 30 frames being processed per second.

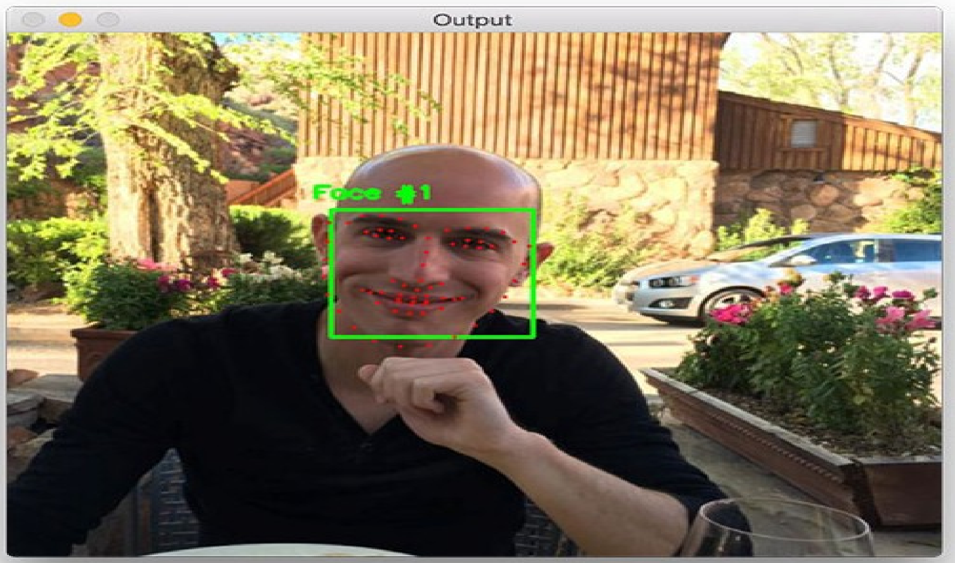
Sure we pay the price for speed – a more cryptic language to deal with, but it's definitely worth it. We can do a lot more, like perform some really complex mathematics on images using C and still get away with good enough speeds for your application.

Efficient
Matlab uses just way too much system resources. With OpenCV, we can get away with as little as 10mb RAM for a real-time application. Although with today's computers, the RAM factor isn't a big thing to be worried about. However, our drowsiness detection system is to be used inside a car in a way that is non-intrusive and small; so a low processing requirement is vital.

Convolutional Neural Networks (CNN) Model

- The model we used is built with Keras using Convolutional Neural Networks (CNN). A convolutional neural network is a special type of deep neural network which performs extremely well for image classification purposes. A CNN basically consists of an input layer, an output layer and a hidden layer which can have multiple numbers of layers. A convolution operation is performed on these layers using a filter that performs 2D matrix multiplication on the layer and filter.The CNN model architecture consists of the following layers:Convolutional layer; 32 nodes, kernel size 3Convolutional layer; 32 nodes, kernel size 3Convolutional layer; 64 nodes, kernel size 3Fully connected layer; 128 nodesThe final layer is also a fully connected layer with 2 nodes. In all the layers, a Relu activation function is used except the output layer in which we used Softmax.
- A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.
- The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.
- CNN is a type of neural network model which allows us to extract higher representations for the image content. Unlike the classical image recognition where you define the image features yourself, CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification.

FACIAL LANDMARKS



Facial landmarks are used to localize and represent salient regions of the face, such as:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

Facial landmarks have been successfully applied to face alignment, head pose estimation, face swapping, blink detection and much more.

In today's blog post we'll be focusing on the basics of facial landmarks, including:

1. Exactly what facial landmarks are and how they work.
2. How to detect and extract facial landmarks from an image using dlib, OpenCV, and Python.

Detecting facial landmarks is a *subset* of the *shape prediction* problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In the context of facial landmarks, our goal is detect important facial structures on the face using shape prediction methods.

Detecting facial landmarks is therefore a two step process:

- Step #1: Localize the face in the image.
- Step #2: Detect the key facial structures on the face ROI.

Understanding dlib’s facial landmark detector

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of ***68 (x, y)-coordinates*** that map to facial structures on the face.

The indexes of the 68 coordinates can be visualized on the image below:

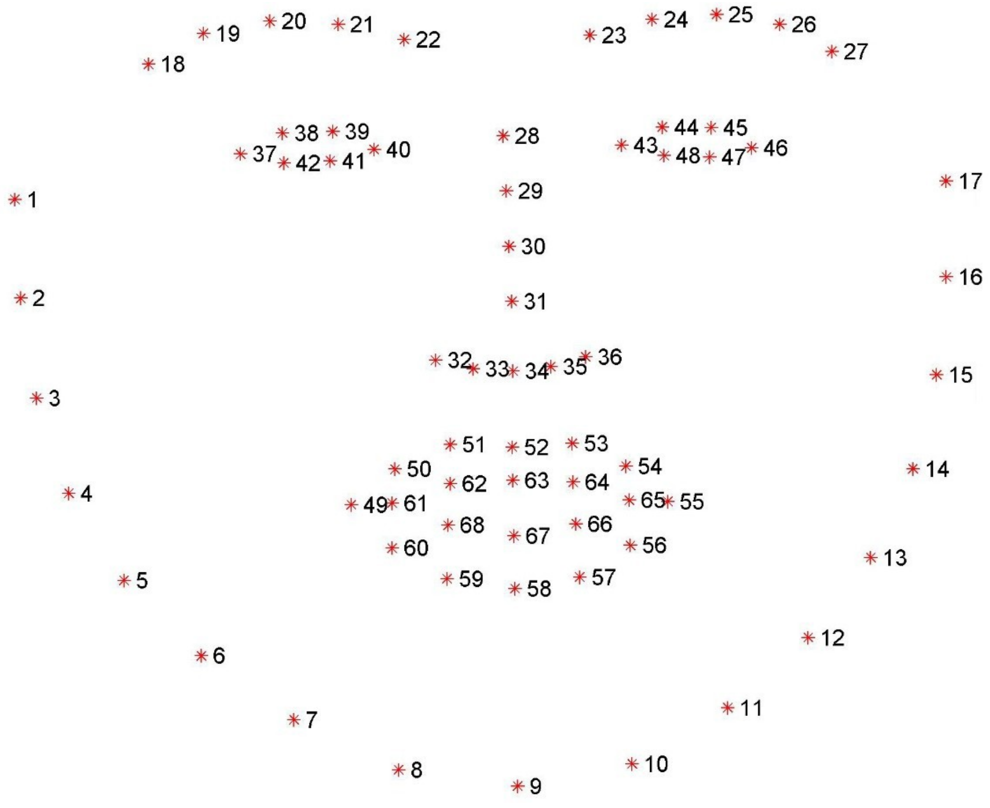


Figure2: Visualizing the 68 facial landmark coordinates from the iBUG 300-W dataset (higher resolution).

These annotations are part of the 68 point **iBUG 300-W dataset** which the dlib facial landmark predictor was trained on.

It’s important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the **HELEN dataset**.

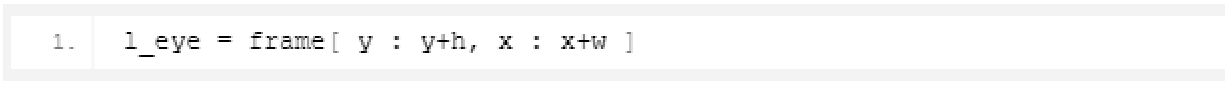
Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

Complete work plan layout

Step 1 – Take Image as Input from a Camera
With a webcam, we will take images as input. So to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV, **cv2.VideoCapture(0)** to access the camera and set the capture object (cap). **cap.read()** will read each frame and we store the image in a frame variable.

Step 2 – Detect Face in the Image and Create a Region of Interest (ROI)
To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes gray images in the input. We don't need color information to detect the objects. We will be using haar cascade classifier to detect faces. This line is used to set our classifier **face = cv2.CascadeClassifier(' path to our haar cascade xml file')**. Then we perform the detection using **faces = face.detectMultiScale(gray)**. It returns an array of detections with x,y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

Step 3 – Detect the eyes from ROI and feed it to the classifier
The same procedure to detect faces is used to detect eyes. First, we set the cascade classifier for eyes in **leye** and **reye** respectively then detect the eyes using **left_eye = leye.detectMultiScale(gray)**. Now we need to extract only the eyes data from the full image. This can be achieved by extracting the boundary box of the eye and then we can pull out the eye image from the frame with this code.



```
l_eye = frame[ y : y+h, x : x+w ]
```

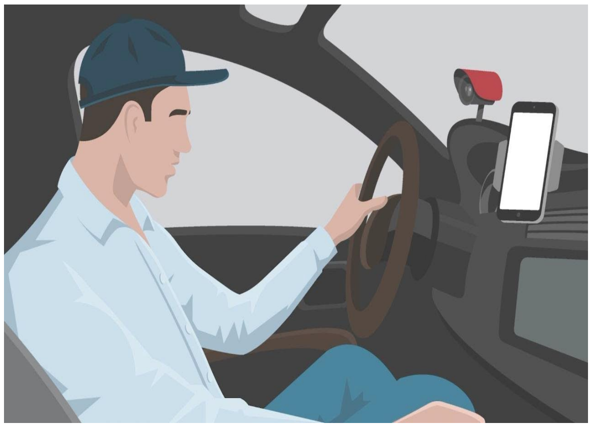
l_eye only contains the image data of the eye. This will be fed into our CNN classifier which will predict if eyes are open or closed. Similarly, we will be extracting the right eye into **r_eye**.

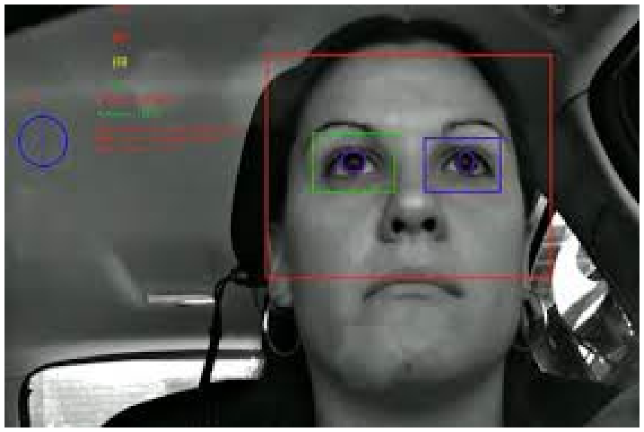
Step 4 – Classifier will Categorize whether Eyes are Open or Clos

We are using CNN classifier for predicting the eye status. To feed our image into the model, we need to perform certain operations because the model needs the correct dimensions to start with. First, we convert the color image into grayscale using **r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY)**. Then, we resize the image to 24*24 pixels as our model was trained on 24*24 pixel images **cv2.resize(r_eye, (24,24))**. We normalize our data for better convergence **r_eye = r_eye/255** (All values will be between 0-1). Expand the dimensions to feed into our classifier. We loaded our model using **model = load_model('models/cnnCat2.h5')** . Now we predict each eye with our model **lpred = model.predict_classes(l_eye)**. If the value of lpred[0] = 1, it states that eyes are open, if value of lpred[0] = 0 then, it states that eyes are closed.

Step 5 – Calculate Score to Check whether Person is Drowsy

The score is basically a value we will use to determine how long the person has closed his eyes. So if both eyes are closed, we will keep on increasing score and when eyes are open, we decrease the score. We are drawing the result on the screen using cv2.putText() function which will display real time status of the person.

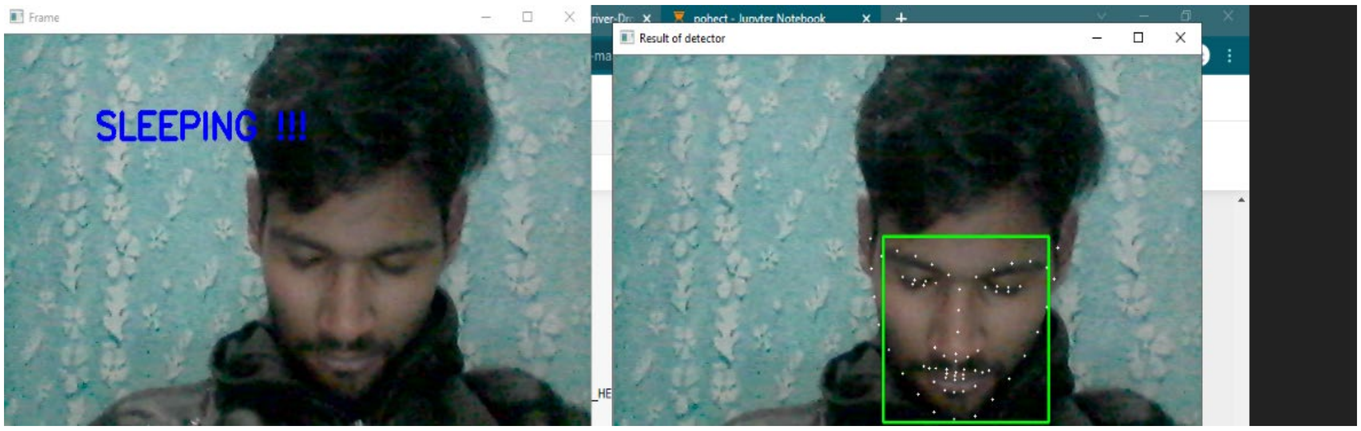


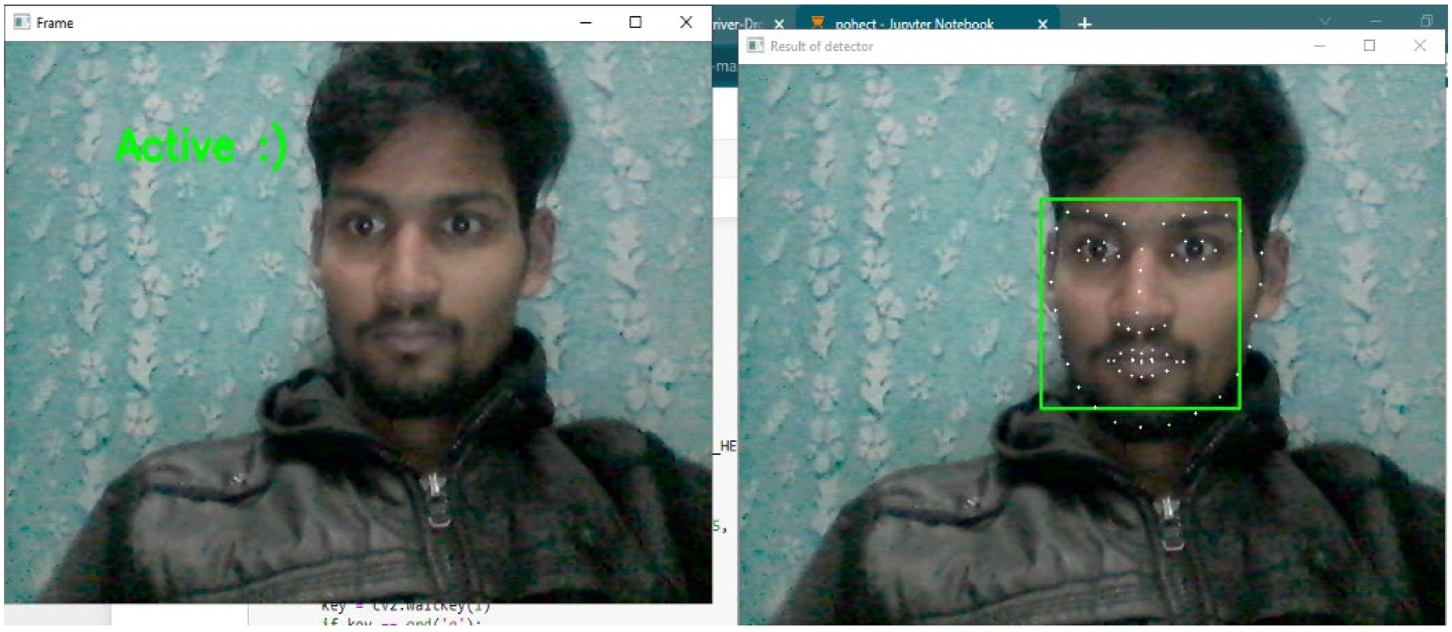


SOURCE CODE

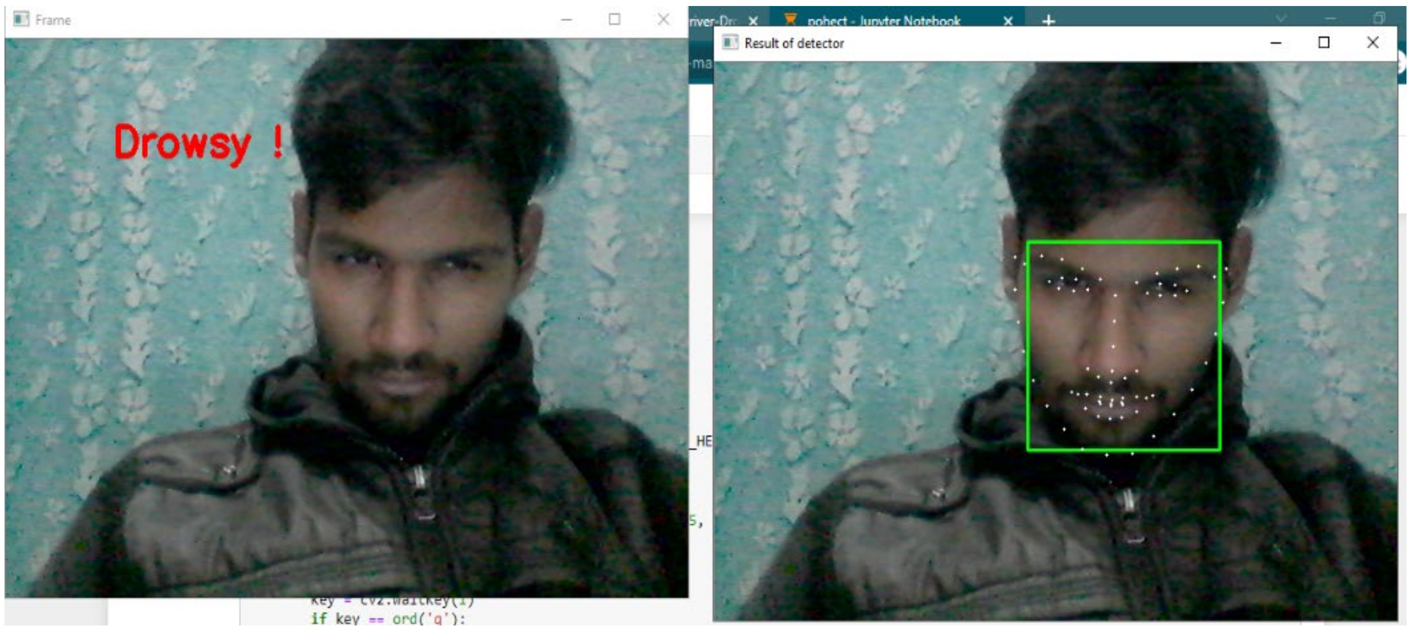
```
 #Importing OpenCV Library for basic image processing functions
 import cv2
# Numpy for array related functions
 import numpy as np
# Dlib for deep learning based Modules and face landmark detection
 import dlib
#face_utils for basic operations of conversion
 from imutils import face_utils
  #Initializing the camera and taking the instance
 cap = cv2.VideoCapture(0)

#Initializing the face detector and landmark detector
 detector = dlib.get_frontal_face_detector()
 predictor = dlib.shape_predictor("C:\\Users\\utkarsh\\Desktop\\NEW Project\\Driver-
 Drowsiness-Detection-master\\shape_predictor_68_face_landmarks.dat")
 #status marking for current state2
 sleep = 0
 drowsy = 0
 active = 0
 status=""
 color=(0,0,0)
 def compute(ptA,ptB):
        dist = np.linalg.norm(ptA - ptB)
        return dist
 def blinked(a,b,c,d,e,f):
        up = compute(b,d) + compute(c,e)
        down = compute(a,f)
        ratio = up/(2.0*down)

        #Checking if it is blinked
        if(ratio>0.25):
                return 2
        elif(ratio>0.21 and ratio<=0.25):
                return 1
        else:
                return 0
while True:
  _, frame = cap.read()
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

  faces = detector(gray)
  #detected face in faces array
  for face in faces:
      x1 = face.left()
      y1 = face.top()
      x2 = face.right()
      y2 = face.bottom()

      face_frame = frame.copy()
      cv2.rectangle(face_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

      landmarks = predictor(gray, face)
      landmarks = face_utils.shape_to_np(landmarks)

      #The numbers are actually the landmarks which will show eye
      left_blink = blinked(landmarks[36],landmarks[37],
        landmarks[38], landmarks[41], landmarks[40], landmarks[39])
      right_blink = blinked(landmarks[42],landmarks[43],
        landmarks[44], landmarks[47], landmarks[46], landmarks[45])

      #Now judge what to do for the eye blinks
      if(left_blink==0 or right_blink==0):
        sleep+=1
        drowsy=0
        active=0
        if(sleep>6):
                status="SLEEPING !!!"
                color = (255,0,0)

      elif(left_blink==1 or right_blink==1):
        sleep=0
        active=0
      drowsy+=1
      if(drowsy>6):
              status="Drowsy !"
              color = (0,0,255)

    else:
      drowsy=0
      sleep=0
      active+=1
      if(active>6):
              status="Active :)"
              color = (0,255,0)

    cv2.putText(frame, status, (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1.2, color,3)

    for n in range(0, 68):
      (x,y) = landmarks[n]
      cv2.circle(face_frame, (x, y), 1, (255, 255, 255), -1)

    cv2.imshow("Frame", frame)
    cv2.imshow("Result of detector", face_frame)
    key = cv2.waitKey(1)
    if key == ord('q'):
       break;
cap.release()
cv2.destroyAllWindows()
```

OUTPUT

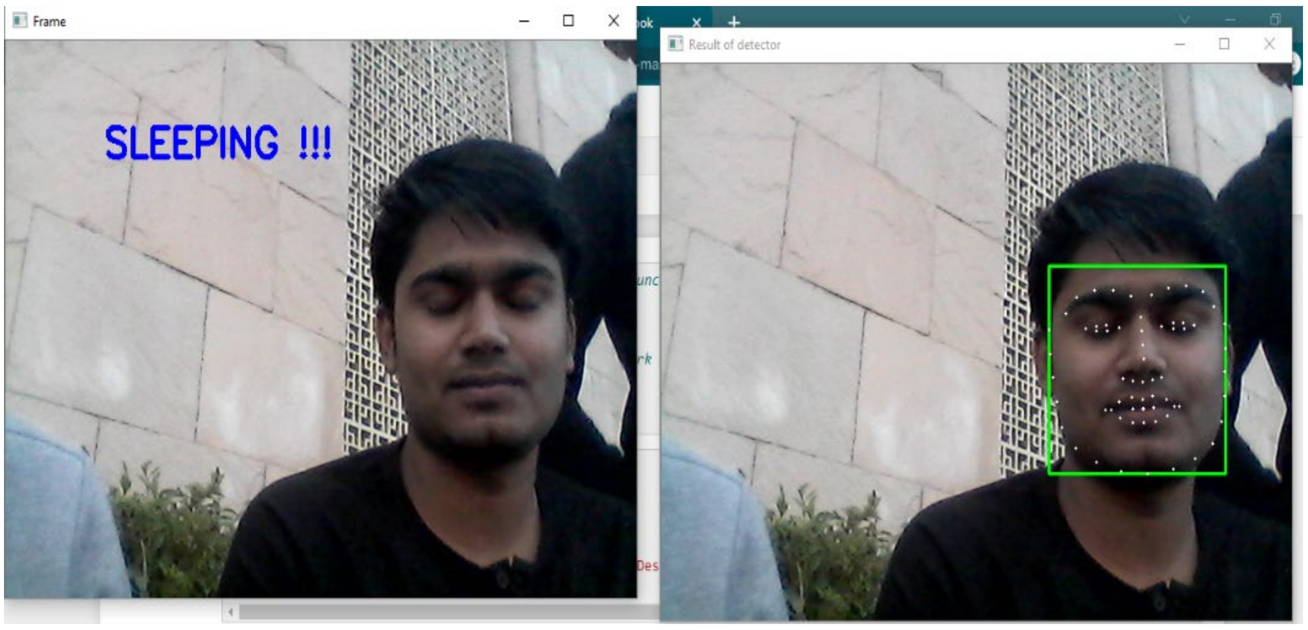




```
key = cv2.waitKey(1)
if key == ord('q'):
```



```
key = cv2.waitKey(1)
if key == ord('q'):
```

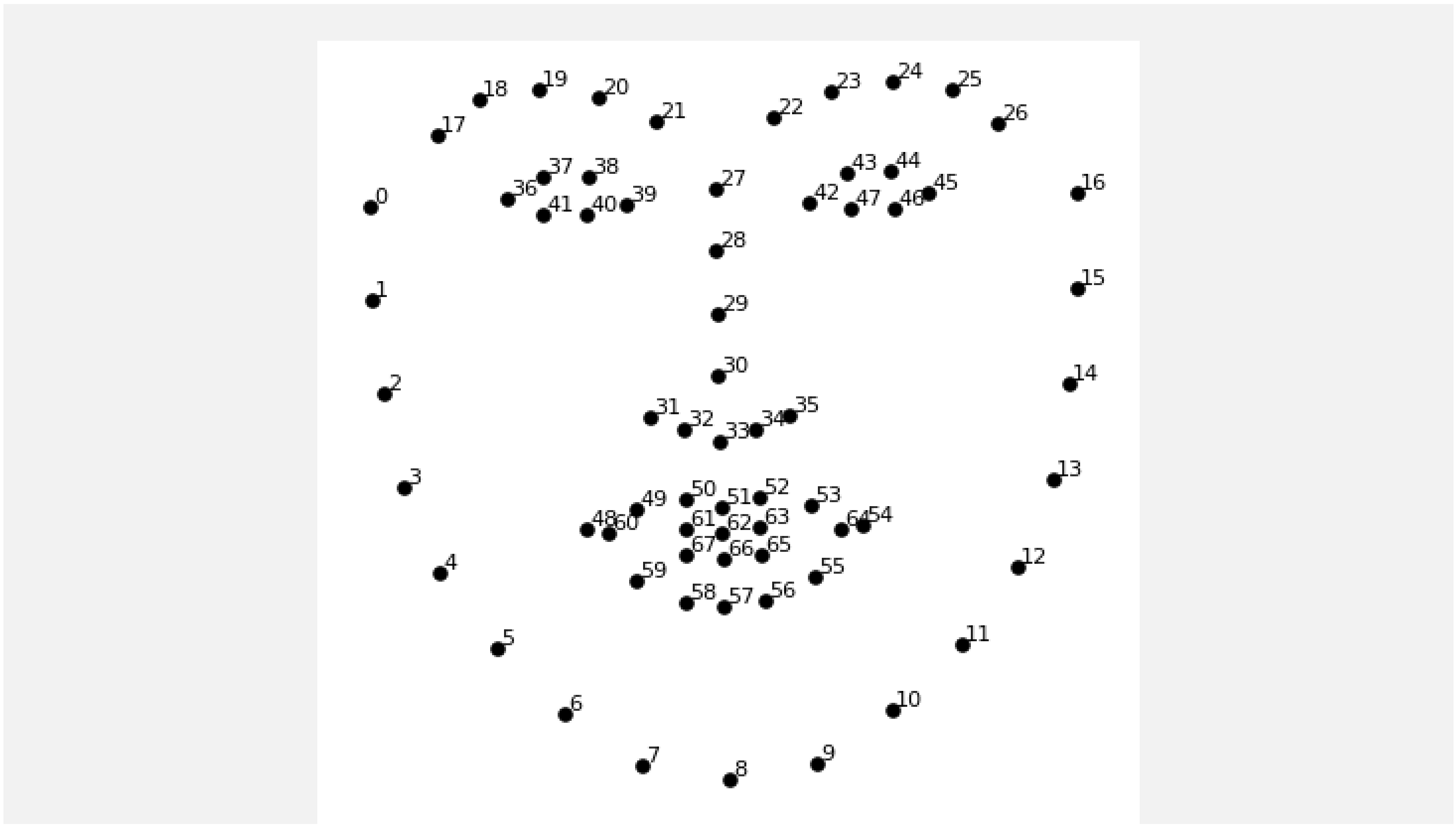


Face landmark detection mechanism

As you can see from above, we initialize the face landmark detector by using the pretrained model. The model is based on ensemble regression trees because the model will predict continuous numbers. You can read the details about the model.

That model is trained on the iBUG-300 W dataset, where it contains images and their corresponding 68 face landmark points. In general, those landmark points belong to the nose, the eyes, the mouth, and the edge of a face. You can download the dataset.

Here is the visualization of the face landmark locations below:

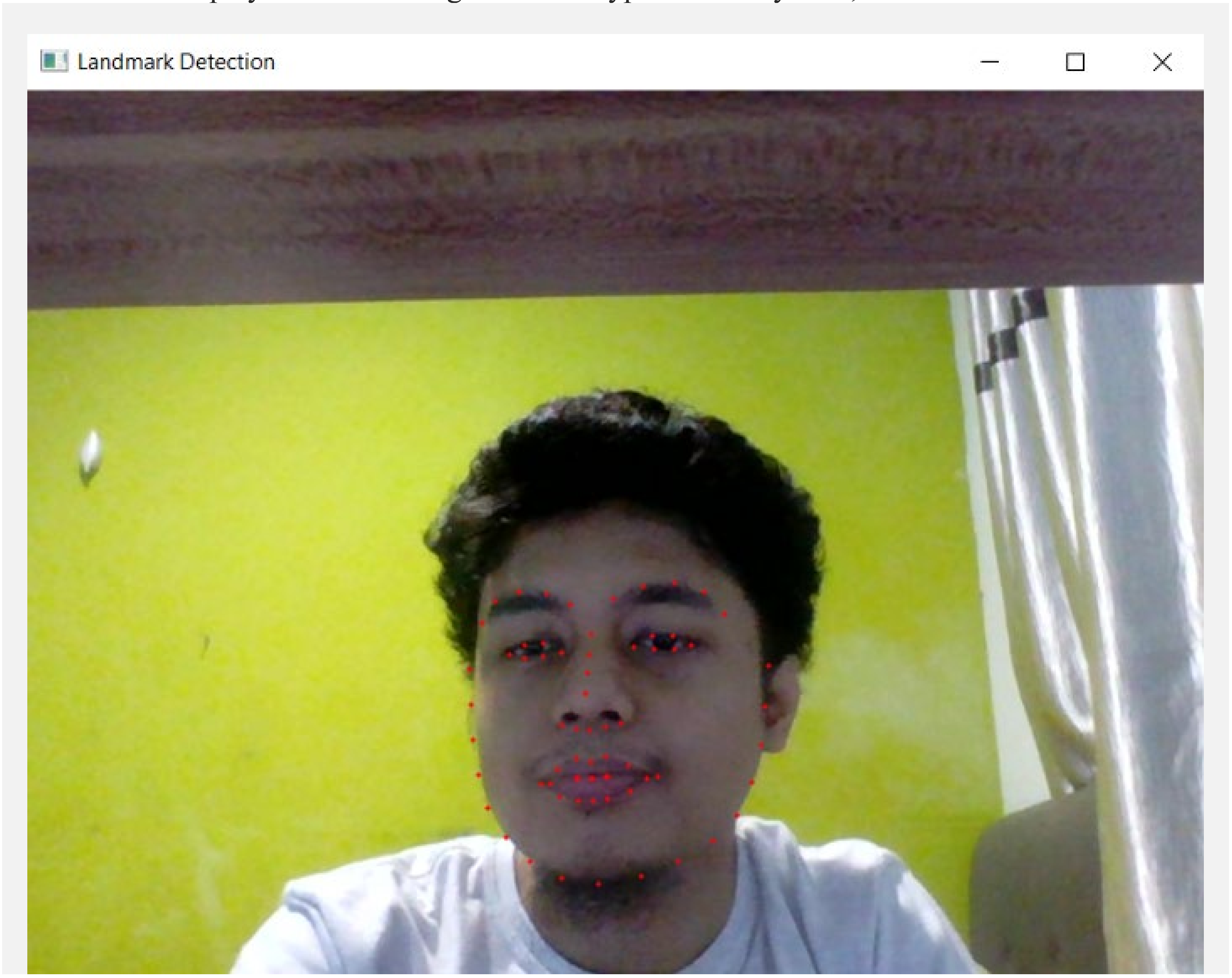


The image is created by Brandon Amos from CMU that creates OpenFace

Implement the face landmark detection

Now you know how the face landmark detection algorithm works. Now let’s implement the algorithm. For implementing that, you can see the code below along with explanations on each line of code:

By combining all the code as one, now let’s try the code! If the code doesn’t have any errors, the webcam will display the result along with the keypoints. In my case, here is the result:



The image is captured by the author.

Face Landmark Detection with Mediapipe

Mediapipe is a tool for implementing ML-based computer vision solutions. The tool is created by Google.

This tool contains varieties computer vision solutions, such as face detection, pose estimation, object detection, and many more.

The advantage of this library is that you can apply the solutions on many platforms, such as web, mobile, PC, and many more.

I've already explained in the previous section to you how to implement face landmark detection using dlib. Now let's implement the face landmark detection using Mediapipe.

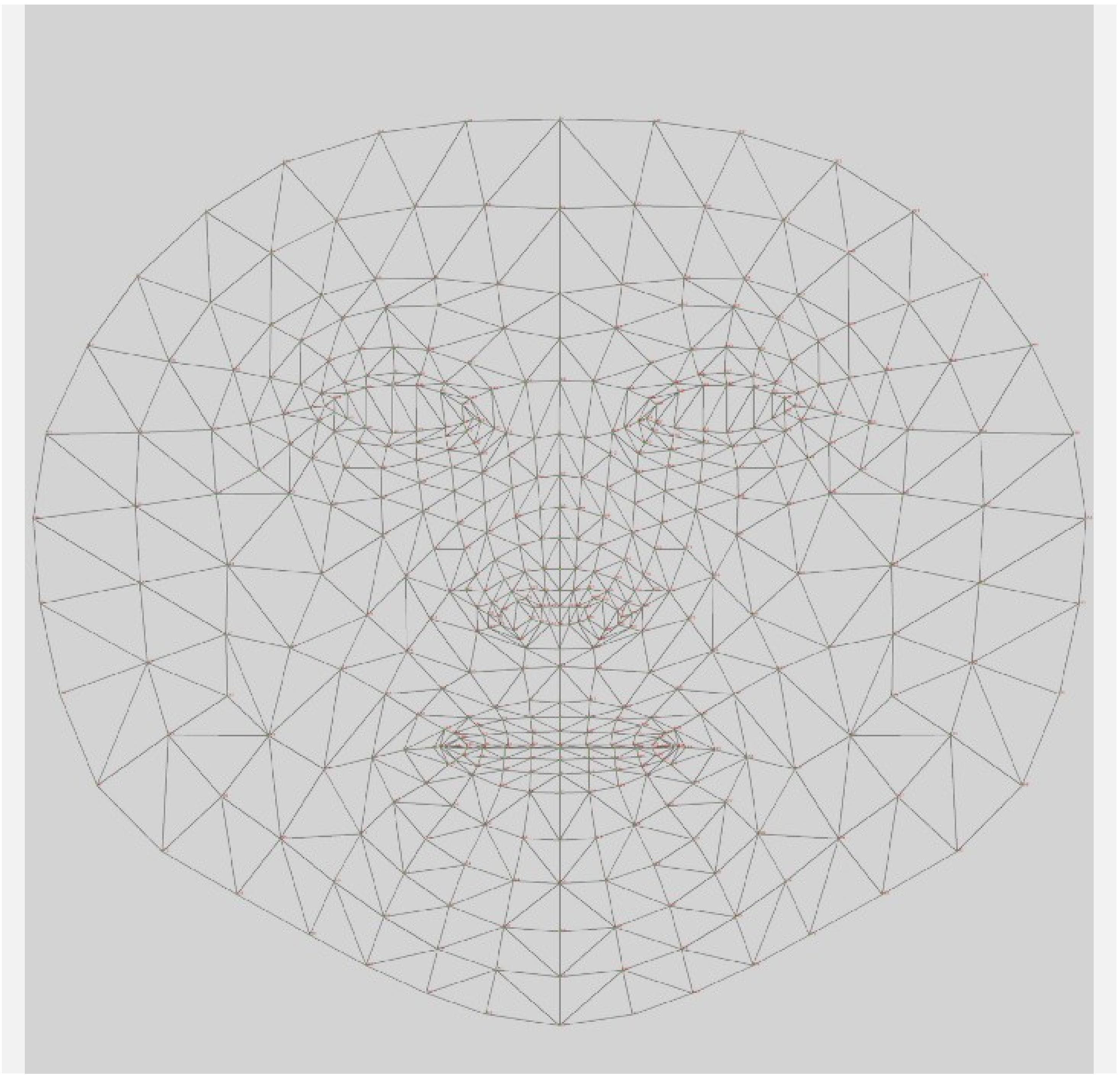
The mechanism

The library uses the BlazeFace model for detecting face landmarks. BlazeFace is a deep learning model that is already optimized for low spec devices like smartphones. Therefore, we can use the model in real-time.

BlazeFace contains two main steps. First, the model detects one or more faces on an image. Second, the image detects around 468 face keypoints by using regression.

Different from the dlib library, this model detects 3D coordinates. Those x and y coordinates are normalized from the image scale. The z coordinate is retrieved by taking the relative calculation between the screen and the model x coordinates. You can read more details.

Here is the flattened mesh from a face with their corresponding indexes:



Dlib

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

To follow or participate in the development of dlib subscribe to dlib on github. Also be sure to read the how to contribute page if you intend to submit code to the project.

To quickly get started using dlib, follow these instructions to build dlib.

Major Features

- **Documentation**
    - Unlike a lot of open source projects, this one provides complete and precise documentation for every class and function. There are also debugging modes that check the documented preconditions for functions. When this is enabled it will catch the vast majority of bugs caused by calling functions incorrectly or using objects in an incorrect manner.
    - Lots of example programs are provided
    - *I consider the documentation to be the most important part of the library*. So if you find anything that isn't documented, isn't clear, or has out of date documentation, tell me and I will fix it.
- **High Quality Portable Code**
    - Good unit test coverage. The ratio of unit test lines of code to library lines of code is about 1 to 4.
    - The library is tested regularly on MS Windows, Linux, and Mac OS X systems. However, it should work on any POSIX system and has been used on Solaris, HPUX, and the BSDs.
    - No other packages are required to use the library. Only APIs that are provided by an out of the box OS are needed.
    - There is no installation or configure step needed before you can use the library. See the How to compile page for details.
    - All operating system specific code is isolated inside the OS abstraction layers which are kept as small as possible. The rest of the library is either layered on top of the OS abstraction layers or is pure ISO standard C++.
- **Machine Learning Algorithms**
    - Deep Learning
    - Conventional SMO based Support Vector Machines for classification and regression
    - Reduced-rank methods for large-scale classification and regression

    - Relevance vector machines for classification and regression
    - General purpose multiclass classification tools
    - A Multiclass SVM
    - A tool for solving the optimization problem associated with structural support vector machines.
    - Structural SVM tools for sequence labeling
    - Structural SVM tools for solving assignment problems
    - Structural SVM tools for object detection in images as well as more powerful (but slower) deep learning tools for object detection.
    - Structural SVM tools for labeling nodes in graphs
    - A large-scale SVM-Rank implementation
    - An online kernel RLS regression algorithm
    - An online SVM classification algorithm
    - Semidefinite Metric Learning
    - An online kernelized centroid estimator/novelty detector and offline support vector one-class classification
    - Clustering algorithms: linear or kernel k-means, Chinese Whispers, and Newman clustering.
    - Radial Basis Function Networks
    - Multi layer perceptrons
- **Numerical Algorithms**
    - A fast matrix object implemented using the expression templates technique and capable of using BLAS and LAPACK libraries when available.
    - Numerous linear algebra and mathematical operations are defined for the matrix object such as the singular value decomposition, transpose, trig functions, etc.
    - General purpose unconstrained non-linear optimization algorithms using the conjugate gradient, BFGS, and L-BFGS techniques
    - Levenberg-Marquardt for solving non-linear least squares problems
    - Box-constrained derivative-free optimization via the BOBYQA algorithm
    - An implementation of the Optimized Cutting Plane Algorithm
    - Several quadratic program solvers
    - Combinatorial optimization tools for solving optimal assignment and min cut/max flow problems as well as the CKY algorithm for finding the most probable parse tree
    - A big integer object
    - A random number object
- **Graphical Model Inference Algorithms**
    - Join tree algorithm for exact inference in a Bayesian network.
    - Gibbs sampler markov chain monte carlo algorithm for approximate inference in a Bayesian network.
    - Routines for performing MAP inference in chain-structured, Potts, or general factor graphs.
- **Image Processing**
    - Routines for reading and writing common image formats.
    - Automatic color space conversion between various pixel types

- Common image operations such as edge finding and morphological operations
- Implementations of the SURF, HOG, and FHOG feature extraction algorithms.
- Tools for detecting objects in images including frontal face detection and object pose estimation.
- High quality face recognition

- **Threading**
  - The library provides a portable and simple threading API
  - A message passing pipe for inter-thread and inter-process communication
  - A timer object capable of generating events that are regularly spaced in time
  - Threaded objects
  - Threaded functions
  - Parallel for loops
  - A thread_pool with support for futures
- **Networking**
  - The library provides a portable and simple TCP sockets API
  - An object to help you make TCP based servers
  - iostream and streambuf objects that enables TCP sockets to interoperate with the C++ iostreams library
  - A simple HTTP server object you can use to embed a web server into your applications
  - A message passing pipe for inter-thread and inter-process communication
  - A tool used to implement algorithms using the Bulk Synchronous Parallel (BSP) computing model
- **Graphical User Interfaces**
  - The library provides a portable and simple core GUI API
  - Implemented on top of the core GUI API are numerous widgets
  - Unlike many other GUI toolkits, the entire dlib GUI toolkit is threadsafe
- **Data Compression and Integrity Algorithms**
  - A CRC 32 object
  - MD5 functions
  - Various abstracted objects representing parts of data compression algorithms. Many forms of the PPM algorithm are included.
- **Testing**
  - A thread safe logger object styled after the popular Java logger log4j
  - A modular unit testing framework
  - Various assert macros useful for testing preconditions
- **General Utilities**
  - A type-safe object to convert between big and little endian byte orderings
  - A command line parser with the ability to parse and validate command lines with various types of arguments and options
  - An XML parser
  - An object that can perform base64 conversions
  - Many container classes
  - Serialization support
  - Many memory manager objects that implement different memory pooling strategies

OPEN CV

OpenCV is a cross-platform library using which we can develop real-time **computer vision applications**. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Let’s start the chapter by defining the term "Computer Vision".

Computer Vision

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields −

- **Image Processing** − It focuses on image manipulation.
- **Pattern Recognition** − It explains various techniques to classify patterns.
- **Photogrammetry** − It is concerned with obtaining accurate measurements from images.

Computer Vision Vs Image Processing

Image processing deals with image-to-image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Applications of Computer Vision

Here we have listed down some of major domains where Computer Vision is heavily used.

Robotics Application

- Localization − Determine robot location automatically
- Navigation
- Obstacles avoidance
- Assembly (peg-in-hole, welding, painting)
- Manipulation (e.g. PUMA robot manipulator)
- Human Robot Interaction (HRI) − Intelligent robotics to interact with and serve people

Medicine Application

- Classification and detection (e.g. lesion or cells classification and tumor detection)
- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery

Industrial Automation Application

- Industrial inspection (defect detection)
- Assembly
- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)

Security Application

- Biometrics (iris, finger print, face recognition)
- Surveillance − Detecting certain suspicious activities or behaviors

Transportation Application

- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

Features of OpenCV Library

Using OpenCV library, you can −

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Perform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images.
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

This tutorial explains the concepts of OpenCV with examples using Java bindings.

OpenCV Library Modules

Following are the main library modules of the OpenCV library.

Core Functionality

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array **Mat**, which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.core**.

Image Processing

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.imgproc**.

Video

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.video**.

Video I/O

This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.videoio**.

calib3d

This module includes algorithms regarding basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence and elements of 3D reconstruction. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.calib3d**.

features2d

This module includes the concepts of feature detection and description. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.features2d**.

Objdetect

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.objdetect**.

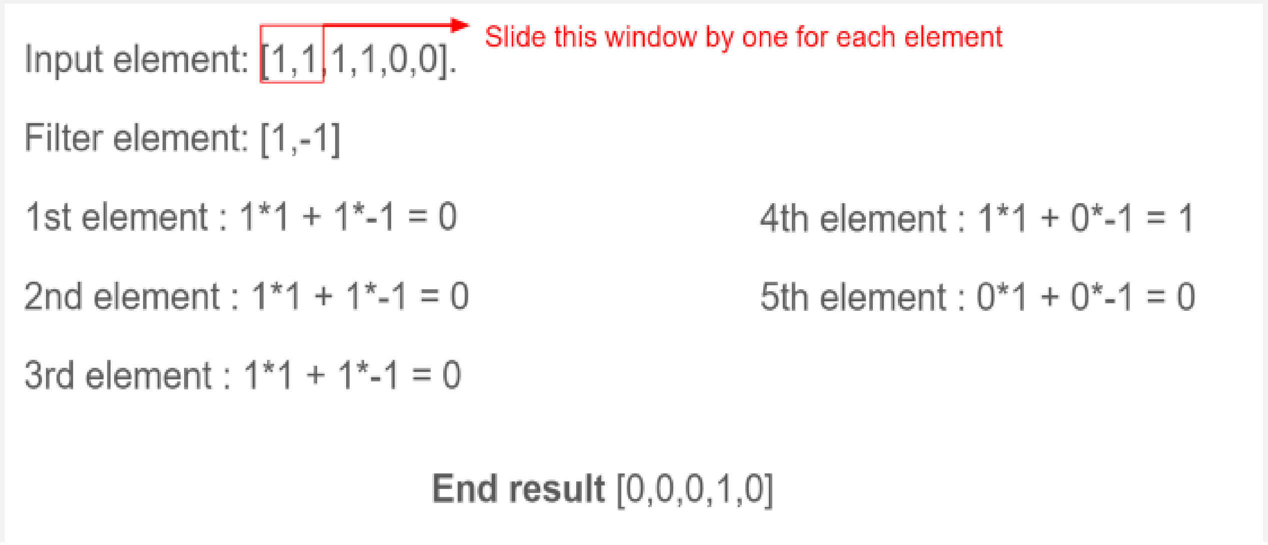
Highgui

This is an easy-to-use interface with simple UI capabilities. In the Java library of OpenCV, the features of this module is included in two different packages namely, **org.opencv.imgcodecs** and **org.opencv.videoio**.

Principles of CNN

Convolution

A convolution sweeps the window through images then calculates its input and filter dot product pixel values. This allows convolution to emphasize the relevant features.



Slide this window by one for each element

Input element: [1,1,1,1,0,0].

Filter element: [1,-1]

1st element : 1*1 + 1*-1 = 0

2nd element : 1*1 + 1*-1 = 0

3rd element : 1*1 + 1*-1 = 0

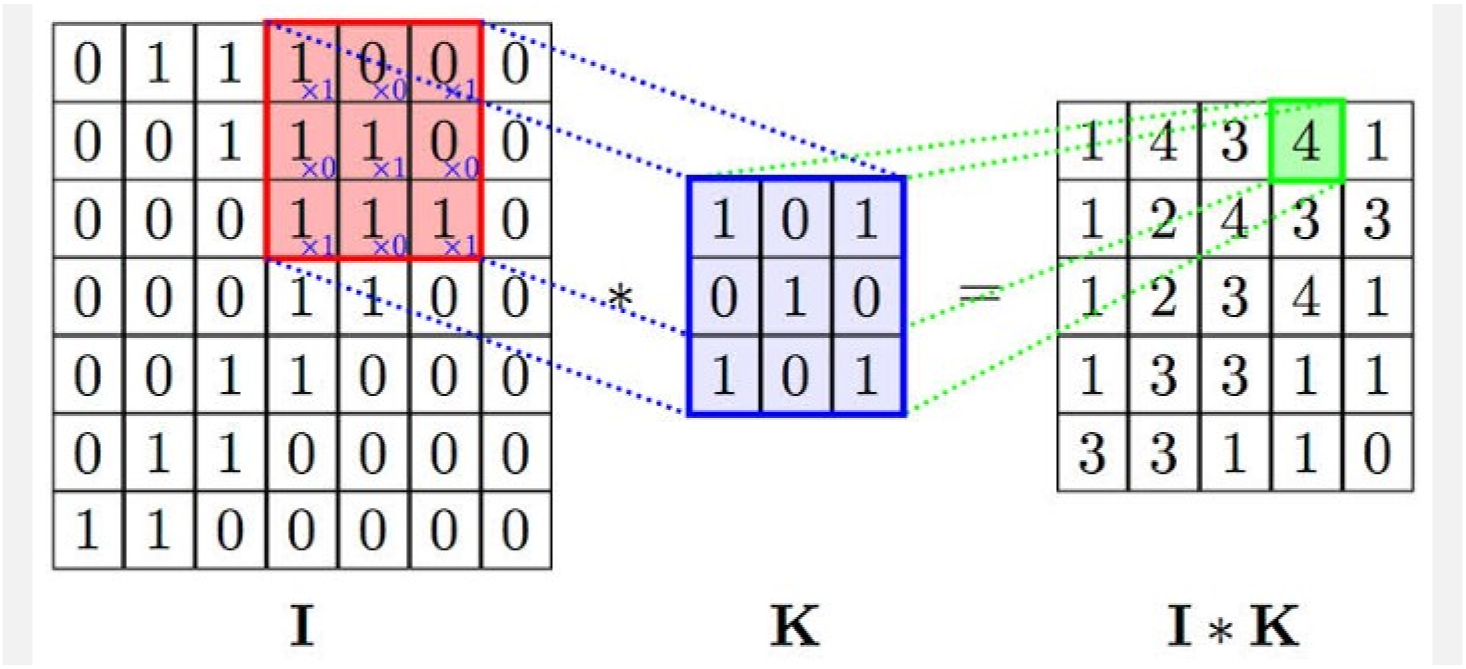
4th element : 1*1 + 0*-1 = 1

5th element : 0*1 + 0*-1 = 0

End result [0,0,0,1,0]

1D Convolution Operation with features(filter)

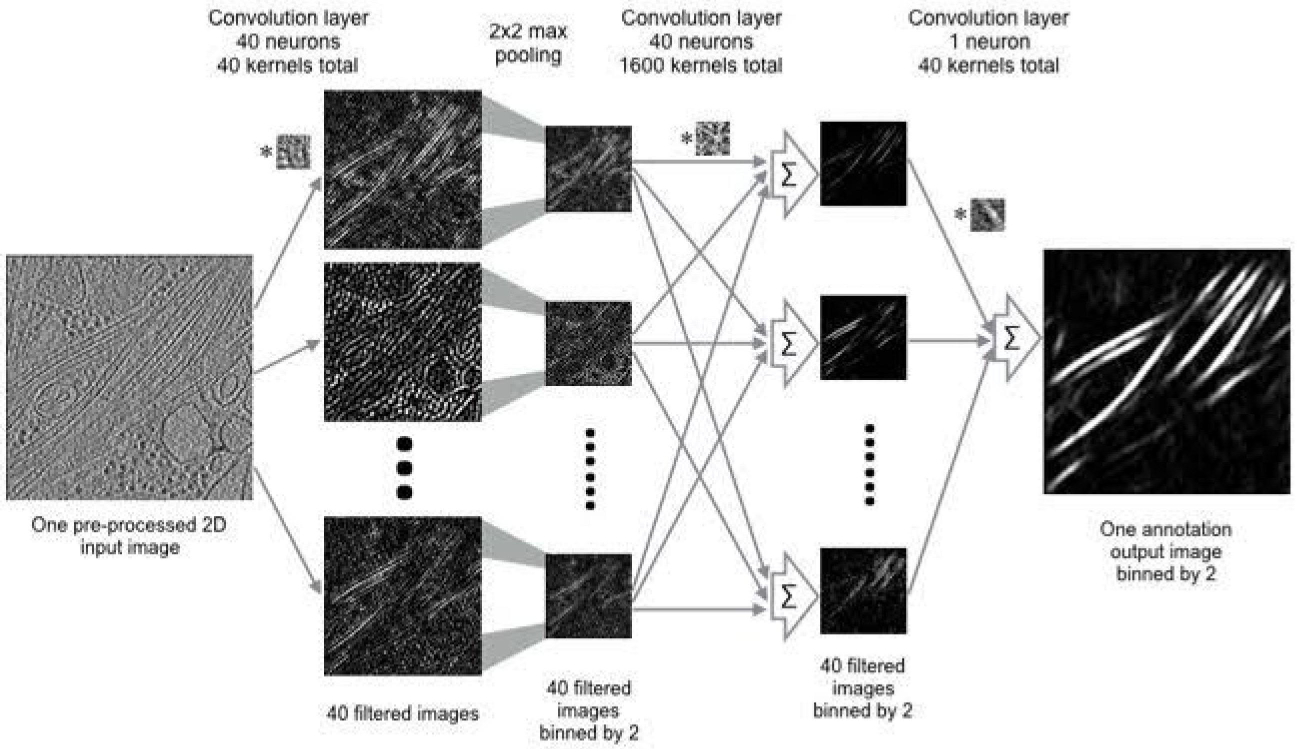
Look at this input. We will encase the window elements with a small window, dot multiplies it with the filter elements, and save the output. We will repeat each operation to derive 5 output elements as [0,0,0,1,0]. From this output, we can know that the feature change(1 becomes 0) in sequence 4. The filter has done well to identify the input values. Similarly, this happened for 2D Convolutions as well.



2D Convolution Operation with features(filter) — [Source]

With this computation, you detect a particular feature from the input image and produce **feature maps** (convolved features) which emphasizes the important features. These convolved features will always change depending on the filter values affected by the gradient descent to minimize prediction loss.

Furthermore, The more filters deployed, the more features that CNN will extract. This allows more features found but with the cost of more training time. There is a sweet spot for the number of layers, usually, I will put 6 for 150 x 150 size of image.



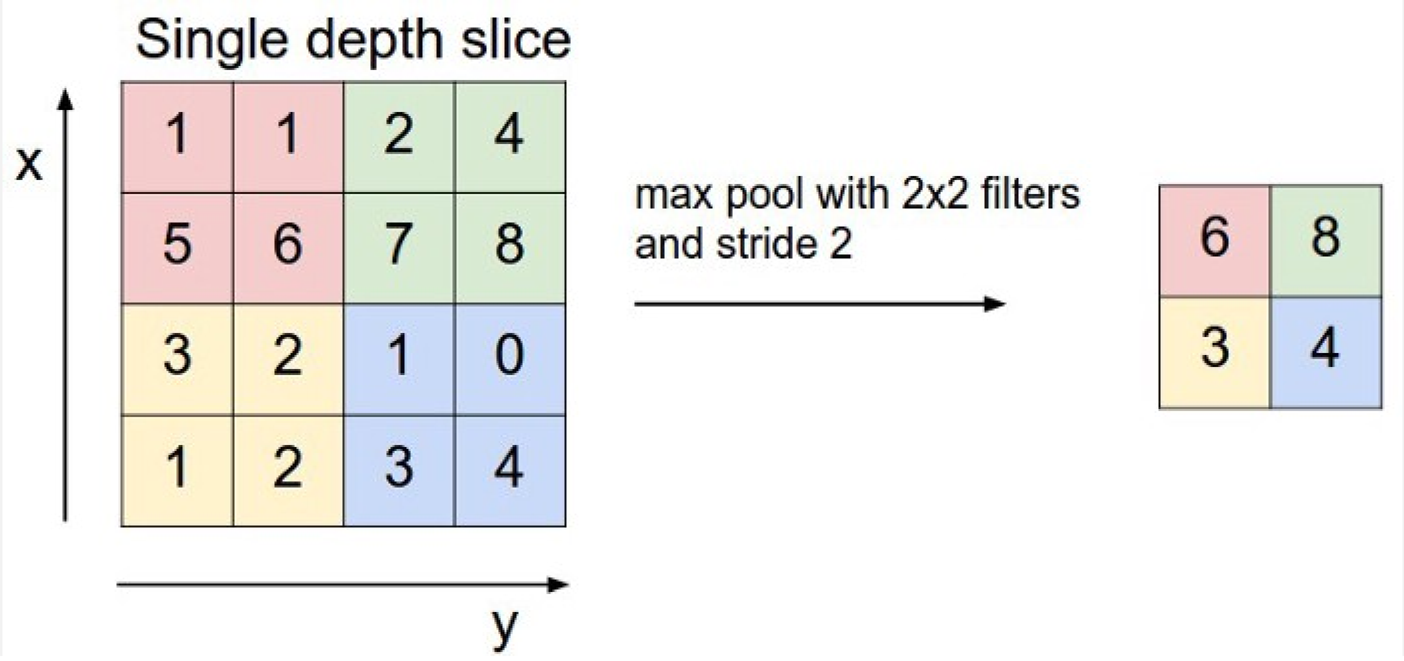
Feature map in each layer of CNN (source)

However, what about the corner or side values. They do not have enough adjacent blocks to fit the filter. Should we remove them?

No, because you would lose important information. Therefore, what you want to do instead is **padding;** you pad the adjacent feature map output with 0. By inserting 0 to its adjacent, you no longer need to exclude these pixels.

Essentially, these convolution layers promote **weight sharing** to examine pixels in kernels and develop visual context to classify images. Unlike Neural Network (NN) where the weights are independent, CNN's weights are attached to the neighboring pixels to extract features in every part of the image.

Max Pooling



We take the maximum max pooling slices of each 2x2 filtered areas (source)

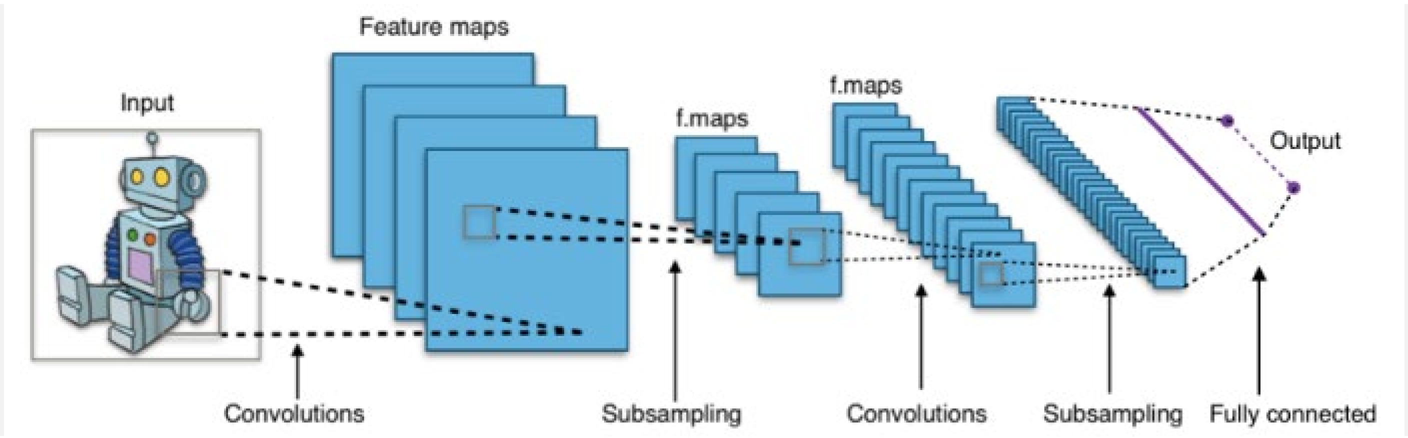
CNN uses **max pooling** to replace output with a max summary to reduce data size and processing time. This allows you to determine features that produce the highest impact and reduces the risk of overfitting.

Max pooling takes two **hyperparameters**: stride and size. The stride will determine the skip of value pools while the size will determine how big the value pools in every skip.

Activation Function (ReLU and Sigmoid)

After each convolutional and max pooling operation, we can apply Rectified Linear Unit (ReLU). The ReLU function mimics our neuron activations on a "big enough stimulus" to introduce nonlinearity for values x>0 and returns 0 if it does not meet the condition. This method has been effective to solve diminishing gradients. Weights that are very small will remain as 0 after the ReLU activation function.

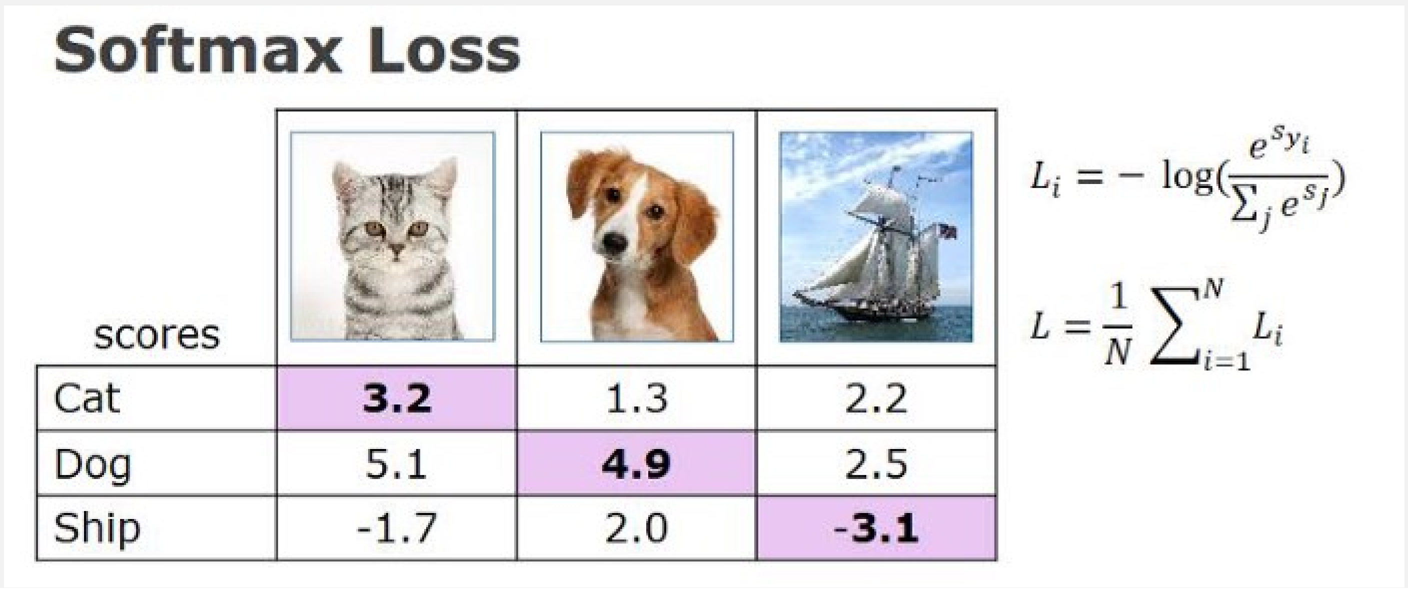
The CNN Big Picture + Fully Connected Layer



CNN architectures with convolutions, pooling (subsampling), and fully connected layers for softmax activation function

Finally, we will serve the convolutional and max pooling feature map outputs with Fully Connected Layer (FCL). We flatten the feature outputs to column vector and feed-forward it to FCL. We wrap our features with **softmax** activation function which assign decimal probabilities for each possible label which add up to 1.0. Every node in the previous layer is connected to the last layer and represents which distinct label to output.

The end results? You will be able to classify the dogs and cat images as below.



Softmax Loss

scores			
Cat	**3.2**	1.3	2.2
Dog	5.1	**4.9**	2.5
Ship	-1.7	2.0	**-3.1**

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

Finding the perfect image classification with softmax (Source)

Cleaning and Preventing Overfitting in CNN

Unfortunately, CNN is not immune to overfitting. If not monitored properly, the model can get trained too much that it could not generalize unseen data. Through my experiences, I have made many beginner overfitting mistakes and how I resolve them as following:

Using test set as the validation set to test the model

Even though we do not use the test set to train the model, the model could adjust the loss function with the test set. This will base the training on the test dataset and is a common cause of overfitting. Therefore, during the training, we need to use **validation sets** then ultimately test the finished model with the unseen test set.

Dataset is relatively small

When dataset is small, it is very easy to specialize onto a few set of rules and forget to generalize. For example, if your model only sees boots as shoes, then the next time you show high heels, it would not recognize them as shoes.

Therefore, in the case of small training data set, you need to artificially boost the diversity and number of training examples. One way of doing this is to add **image augmentations** and creating new variants. These include translating images and creating dimension changes such as zoom, crop, flips, etc.

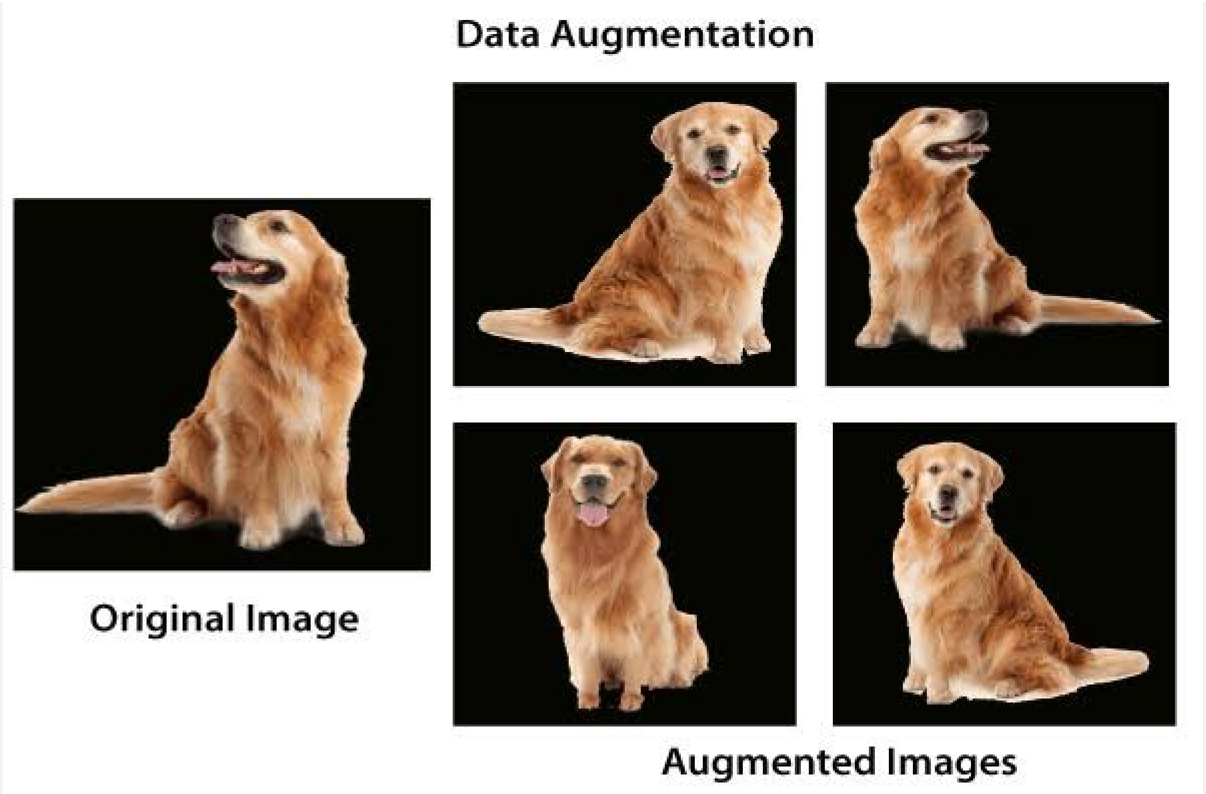


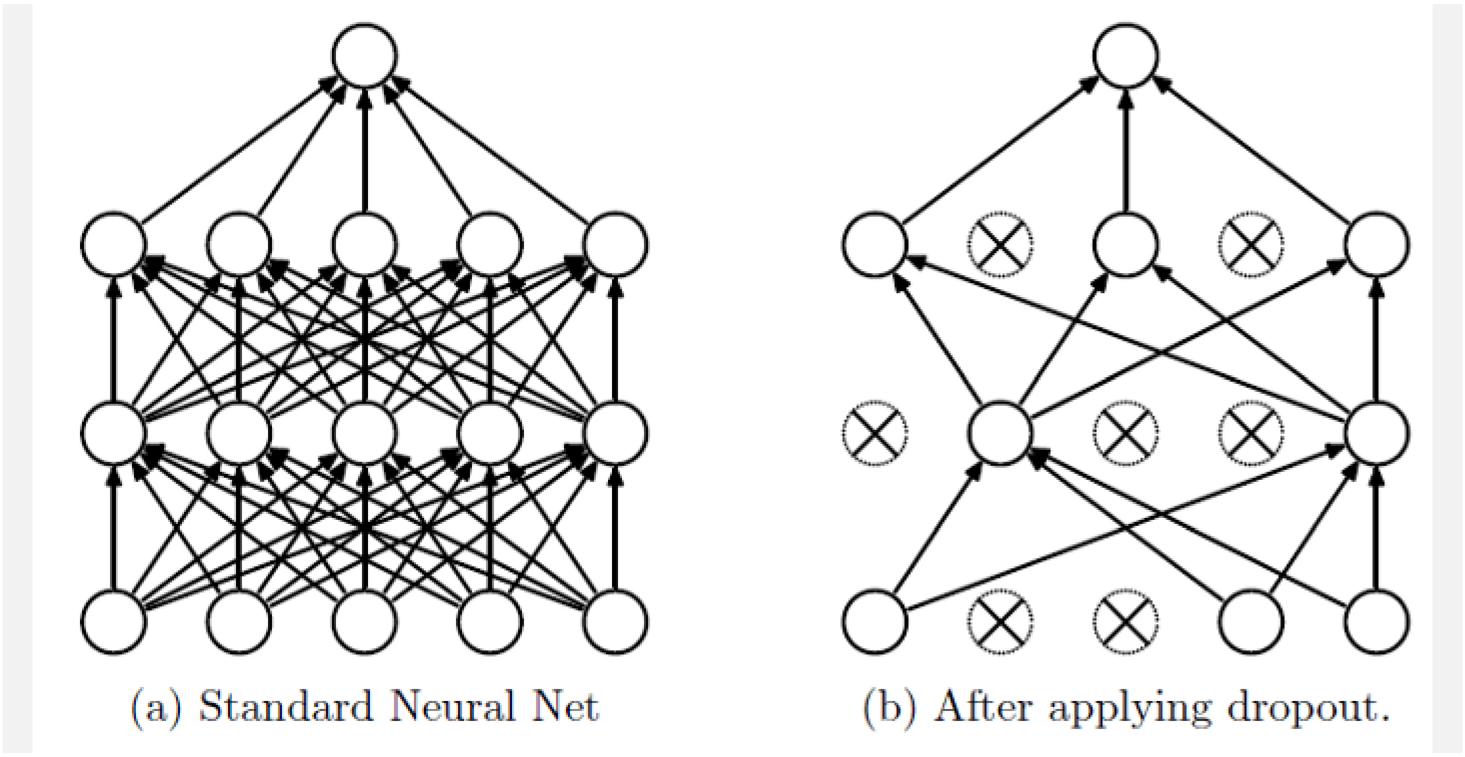
Image augmentation Source

Over Memorization

Too many neurons, layers, and training epochs promote memorization and inhibit generalize. The more you train your model, the more likely it becomes too specialized. To counter this, you could reduce the complexity by removing a few hidden layers and neurons per layer.

Alternatively, you could also use regularization techniques such as **Dropout** to remove activation unit in every gradient step training. Each epoch training deactivates different neurons.

Since the number of gradient steps is usually high, all neurons will averagely have same occurrences for dropout. Intuitively, the more you drop out, the less likely your model memorizes.



Drop out images

Dealing with color images

You can also easily include images with 3 layers of color channels: Red Green Blue (RGB). During convolution, you use 3 separate convolutions for each color channel and train 3-level stack of filters. This allows you to retrieve 3D feature maps.

How could we do better? — Transfer Learning

As the use cases become complex, the complexity of the model needs to improve as well. With a few layers of CNN, you could determine simple features to classify dogs and cats. However, at the deep learning stage, you might want to classify more complex objects from images and use more data. Therefore, rather than training them yourself, **transfer learning** allows you to leverage existing models to classify quickly.

Transfer learning is a technique that reuses an existing model to the current model. You could produce on top of existing models that were carefully designed by experts and trained with millions of pictures.

However, there are a few caveats that you need to follow. First, you need to modify the final layer to match the number of possible classes. Second, you will need to freeze the parameters and set the trained model variables to immutable. This prevents the model from changing significantly.

One famous Transfer Learning that you could use is MobileNet. It is created for mobile devices which have less memory and computational resources. You can find MobileNet in Tensorflow Hub which gathers many pretrained models. You can just simply add your own FCL Layer on top of these models.

Conclusion: CNN to perceive our visual world

CNN is a tough subject but a rewarding technique to learn. It teaches us how we perceive images and learn useful applications to classify images and videos. After learning CNN, I realized that I could use this for my project at Google to detect phishing attacks.

REFERENCE

1. DRIVER FATIGUE AND ROAD ACCIDENTS A LITERATURE REVIEW and POSITION PAPER" (PDF). Royal Society for the Prevention of Accidents. February 2001. Archived from the original (PDF) on 2017-03-01. Retrieved 2017-02-28.
2. 4.1.03. Driver Drowsiness Detection System for Cars". Retrieved 2015-11-05.
3. Sgambati, Frank, Driver Drowsiness Detection
4. Hupp, Stephen L. (October 1998). "Landmark Documents in American History. Version 2.0". Electronic Resources Review. **2** (10): 120–121. doi:10.1108/err.1998.2.10.120.111. ISSN 1364-5137.
5. Walger, D.J.; Breckon, T.P.; Gaszczak, A.; Popham, T. (November 2014). "A Comparison of Features for Regression-based Driver Head Pose Estimation under Varying Illumination Conditions" (PDF). Proc. International Workshop on Computational Intelligence for Multimedia Understanding. IEEE: 1–5. doi:10.1109/IWCIM.2014.7008805. ISBN 978-1-4799-7971-4. S2CID 14928709. walger14headpose.
6. Wijnands, J.S.; Thompson, J.; Nice, K.A.; Aschwanden, G.D.P.A.; Stevenson, M. (2019). "Real-time monitoring of driver drowsiness on mobile platforms using 3D neural networks". NeuralComputing and Applications. **32** (13): 9731–9743. arXiv:1910.06540. Bibcode:2019arXiv191006540W. doi:10.1007/s00521-019-04506-0. S2CID 204459652.
7. Hossain, M. Y.; George, F. P. (2018). "IOT Based Real-Time Drowsy Driving Detection System for the Prevention of Road Accidents". 2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS). **3**: 190–195. doi:10.1109/ICIIBMS.2018.8550026. ISBN 978-1-5386-7516-8. S2CID 54442702.
8. https://www.audi-mediaservices.com/publish/ms/content/en/public/hintergrundberichte/2012/03/05/a_state ment_about/driver_assistance.html Driver assistance systems
9. "BMW model upgrade measures taking effect from the summer of 2013". BMW. 2013- 06-05. Retrieved 2015-11-05.
10. "Driver drowsiness detection". Robert Bosch GmbH. Retrieved 2015-11-05.