Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

Driver Drowsiness Detection using Microservices and Convolutional Neural Network

Shrut Shah
Department of Computer Engineering
Pune Institute of Computer Technology
Maharashtra, India

***Abstract*—Road accidents are one of the main contributors to net fatality rates in India. According to a recent survey in 2020, 43% of road accidents come from drowsy driving. Driving over hours and being in the same state makes the driver feel exhausted and fatigue leading them to drowsiness. A report from Road Transport of India stated that on average 5210 tragedies occur each year alone on the highways of India. A primary system to measure and alert the driver must be mandatory for any moving vehicle. In this paper, a modern approach is proposed for real-time drowsiness detection. A production-grade application with microservice architecture is one of the main focus of this paper. The process of building up the data, augmenting it to a desired level and finally labeling is presented. The customized state of art model is proposed that can achieve an accuracy of 83.65%.**

Keywords—Deeplearning; microservices; drowsiness detection system; real-time application; kubernetes

I. INTRODUCTION

Out of all the ways to commute and transport, roadways still tops the list. A short sleep or long journey makes us stiff and sluggish leading to drowsiness. It is an eventual process where the driver slowly and gradually becomes unconscious. In such an out of control situation, vulnerability is at max. This may lead to a crash causing loss in some or the other form. Transport drivers who spend their most of life driving are susceptible to such tragedies.

Preventing a scenario is better than to cure it later. Many approaches are put forward to solve the problem in hand. Nevertheless, it is still an industrial as well as an academic challenge.

The field of Machine Learning - traces to centuries back but now with digitalization and big data getting generated - is now revealing new ways of cognitive thinking. The self-learning capability of system is the most powerful energy AI can impart. With the unfolding of Deep learning concepts, spectrum of thinking has reached greater heights. Google providing GPU to researchers made it feasible to overcome hardware dependability. It's possible now to containerize every single module. Kubernetes has the capability to manage these containers. With these technological advancements along with API, microservices were born. They provide a powerful alternative to old conventional monolithic architecture.

II. LITERATURE SURVEY

The goal to provide efficient driver drowsiness detection has been in existence for quite a few years now. This section has a primary aim to summarise approaches, thought process, implementation and measurement metrics of previous researchers. In the first way based on driving skills, characteristics of vehicles, driving patterns were observed. Krajewski et al. [1] were successful to obtain 86% of accuracy. In the second approach data from sensors is collected. These sensors were mainly related to human monitoring. From this data, information about brain activity is obtained. Also, there are signals which have a relation with driver drowsiness. This approach is more logically sound. When Mardi et al. [2] used this approach, accuracy obtained was above 90%. But this approach had a big disadvantage associated with it. To follow it the driver has to decorate himself with quite a number of sensors which is not at all practical. Another approach was to use the duration of yawn and eyes when closed. Danisman et al. [3] measured the relative distance between the screen and the driver and that of human figures. Dwivedi et al. [4] and Park et al. [5] worked on binary classification for drowsiness detection.

III. METHODOLOGY

A. Microservices

Before Microservices, monolithic architecture was used for application design. In Monolithic all the software components of application are assembled and tightly coupled in a single big container. There are advantages to it like it is simple to develop, deploy but everything comes with pros and cons. In a large, complex application if we are required to service or modify some component it would be increasingly difficult eventually slowing down the development process. Also quality of code degrades. It will hamper developer productivity. Rapid updates are not possible with monolithic because the entire application needs to be redeployed again. In consideration of scalability too, monolithic is not suitable because there may be a case where only few components require scaling in a period of time and with this architecture only thing can be done is scaling the entire application. This results in overload on memory, CPU and other software and hardware. In monolithic if one component fails, the entire application fails. It's not flexible because the entire application needs to be designed in one language. With the more and more adoption of web apps over desktop apps and quite a number of drawbacks in monolithic, there was a need for a new approach.

Microservice architecture constructs an application as independent services around the business domain. These microservices are autonomous which means they have their structure and communicate with other microservices using API calls. With the structure in hand every service can work on its functions and capabilities. It increases the quality of

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

code. It is also easy to understand. In microservices, every service has a separate codebase that can be handled independently by different development teams. These services are loosely coupled. All of them are deployed independently. Now here if there is a need to update or modify then individual service can be done without affecting other services. After updating only that service requires redeployment and not all the services. In terms of scaling, during peak hours particular service which requires scaling can be achieved keeping all other others as it is. The internal implementation details are abstracted in technology stack and all the services can use different languages and libraries. It makes the architecture more flexible. In such architecture, if some component or service is down it doesn't affect the overall application. It helps in troubleshooting, debugging and quick fix of the bug. Hence the adoption of microservices is justified.

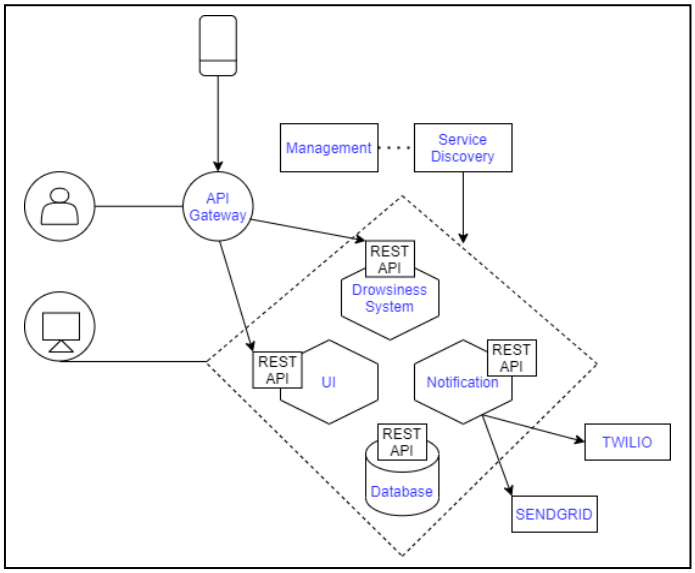

Fig. 1. Microservice Architecture

Microservice architecture for the Application is as shown above in the Figure 1. API gateway is the main management unit that resides middle way between client and application. It has the duty to manage all the API calls to and fro. It works on a reverse proxy mechanism, where it gathers all the required services and replies with a relevant response. Kubernetes is used as a container orchestration tool. It is the best container management tool as it has functions like load balancing, deployment, rolling updates, security, networking. Through API gateway clients get connected to microservice structure. This is possible through REST API. I have used GRPC for managing RPC which is a more efficient way of handling API calls. Service discovery manages all the in and out calls for interoperability. Management block is for managing the nodes assigned and taken off as per the requirement. There is a UI for signup/login to the application. It checks for authenticity of the user and helps in session management. On the portal, there is static data for about, contributors page. The drowsiness detection system will be covered in the next section. There is the database unit which is written in two languages. Redis: for caching, low latency and session management. SQLite: for storing registration details and analysis data. There is a notification unit that has two ways of handling it. Twilio adapter is for in-app alert notification. This can be used in emergency situations where the driver is notified in seconds. Sendgrid adapter is for sending over detailed reports via email through a secured SMTP protocol.

B. Dataset and Preprocessing

For training the model, a dataset needs to be generated. I created my own dataset by running a python script. In the dataset, I've captured images of 10 family members in different lighting conditions essentially in day and night time. I was able to generate 1000 images from it. I removed the unwanted images to remove noise from the system. As this is less in number I went for image augmentation through which I could get 5000 images. Data labeling was done manually on these images into different categories like day or night, drowsy or not drowsy.

C. Model Preparation

When it comes to image processing, Convolutional Neural Network (CNN) is the most suitable choice for model preparation. CNN can be divided into two major categories. In the first one, it extracts features by template matching. It is a sequence of filters (which results in feature maps) followed by a normalize and resize phase that keeps on repeating until it's passed as a vector to the second part. The second part is more like a typical Artificial Neural Network (ANN). It accepts the vector. The main aim of this part is classification. Relu activation function is used in all the layers but the last layer uses softmax to classify. There are 5 layers to consider. Convolutional layer: In this layer extraction of features take place by the use of different filters. As a result, we obtain feature maps. Pooling layer: In this layer feature maps get accepted, which undergo size reduction. All the essential features are stored. Relu correction layer: It is the activation function which has one primary purpose to keep positive values as it is and zero the negative values. In this way, it gets the job done of normalization. Fully Connected layer: It is the same layer that we see in ANN i.e. input layer as a vector, hidden layers and finally output layers. Softmax Activation layer: It is used for classification in numerous categories day or night, drowsy or not drowsy.

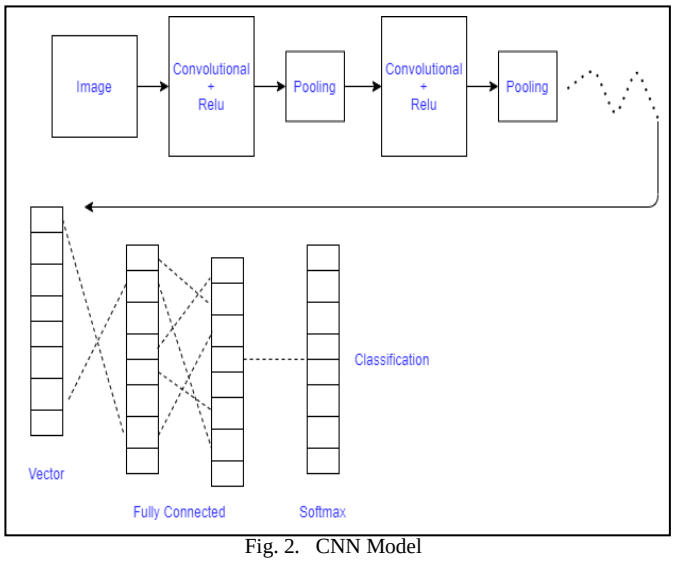

Fig. 2. CNN Model

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 12, December-2020

D. Mobile Architecture

The other side of API gateway is mobile application. The Dlib library is used for extraction of facial landmarks from the image. The data gathered is then sent to drowsiness detection application. The app communicates with the camera using hardware abstraction layer. After this process through API gateway it connects to the microservice architecture.

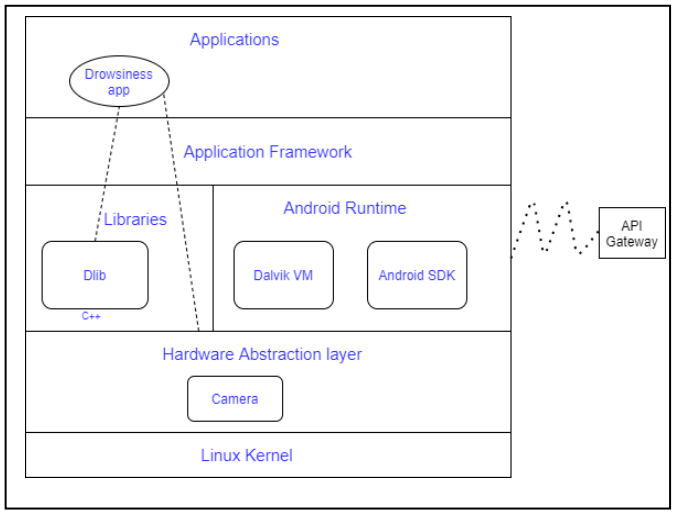

Fig. 3. Mobile Architecture

IV. COMPUTATION RESULTS

The 5000 images in dataset is divided into training and testing data using a 7:3 ratio. This is to ensure and test the model before using it in a production environment. The main categories in which the Softmax layer will categorize are drowsy or not drowsy in day and night conditions. The table below gives the accuracy metrics under different conditions on train and test data. For model generation, I've used a GPU provided by Google Colab. After training the model, weights were captured and stored in '.h5' file extension. This step is necessary to work in a real-time scenario.

TABLE I. ACCURACY METRIC

Dataset	Category	Accuracy
Train	Day, Drowsy	0.87
	Night, Drowsy	0.83
	Day, Not drowsy	0.86
	Night, Not drowsy	0.84
Test	Day, Drowsy	0.85
	Night, Drowsy	0.80
	Day, Not drowsy	0.84
	Night, Not drowsy	0.80

V. CONCLUSION

This paper provides a modern approach for driver drowsiness detection. In the implementation of such a system new practically accepted technologies are used. In a fast and rapidly changing world, microservices is the solution to any application architecture. Such service mesh is proposed in this paper. Convolution Neural Network is used for image processing and classification into different categories. Databases like Redis, SQLite are used for advanced functionality in memory storage. With this entity as a whole, an average accuracy of 83.65% was possible.

REFERENCES

[1] Krajewski J, Sommer D, Trutschel U, Edwards D, Golz M, "Steering wheel behavior based estimation of fatigue", The fifth international driving symposium on human factors in driver assessment, training and vehicle design, 2009, pp. 118-124.

[2] Mardi Z, Ashtiani SN, Mikaili M, "EEG-based drowsiness detection for safe driving using chaotic features and statistical tests", Journal of medical signals and sensors, 2011,vol. 1, pp. 130–137

[3] Danisman T, Bilasco IM, Djeraba C, Ihaddadene N, "Drowsy driver detection system using eye blink patterns", Machine and Web Intelligence (ICMWI) IEEE, 2010, pp. 230-233.

[4] Dwivedi K, Biswaranjan K, Sethi A, "Drowsy driver detection using representation learning", Advance Computing Conference (IACC), IEEE, 2014, pp. 995-999

[5] Park S, Pan F, Kang S, Yoo CD, "Driver Drowsiness Detection System Based on Feature Representation Learning Using Various Deep Networks", Asian Conference on Computer Vision Springer, 2016, pp. 154-164.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)