CS48007- Term Project

Part 1: Anomaly Detection in ECG Data using CNN-based Autoencoders with Various Approaches on Error Propagation

Introduction:

The project focuses on employing Convolutional Neural Networks (CNN) based Autoencoders for the task of anomaly detection in Electrocardiogram (ECG) data. The primary objective is to train an autoencoder to learn the intrinsic representation of normal ECG patterns and utilize various loss functions, including Mean Squared Error (MSE), Huber loss, Mean Absolute Error (MAS), and Cosine Similarity, to assess the model's performance. The dataset used is The MIT-BIH Arrhythmia Database, providing a diverse collection of ECG recordings.

Dataset Description:

The MIT-BIH Arrhythmia Database comprises 48 half-hour excerpts of two-channel ambulatory ECG recordings from 47 subjects. This dataset, collected between 1975 and 1979, includes recordings selected to represent a mix of common and less common clinically significant arrhythmias. The recordings were digitized at 360 samples per second per channel, providing a comprehensive representation of various heart conditions.

Constructing the Model:

The chosen approach involves a CNN-based Autoencoder, consisting of an encoder and a decoder. The model is trained on normal ECG data to learn the underlying patterns. During training, different loss functions, including MSE, Huber loss, MAS, and Cosine Similarity, are applied to measure the dissimilarity between the actual and reconstructed data.

Training the Data and Evaluation:

The model undergoes training on a subset of the MIT-BIH Arrhythmia Database, focusing on normal ECG samples. Various loss functions are employed to evaluate and compare the model's performance. The reconstruction error, specific to each loss function, is calculated for both normal and abnormal data. Anomaly detection is performed by analyzing these reconstruction errors, with higher errors indicating potential abnormalities. Evaluation metrics such as accuracy and average reconstruction error are utilized to assess the model's effectiveness.

Comparative Analysis of Loss Functions:

The project conducts an in-depth analysis of different loss functions to understand their impact on model performance. MSE is a standard choice, while Huber loss provides a robust alternative by mitigating the influence of outliers. MAS focuses on the absolute differences between actual and predicted values, and Cosine Similarity measures the cosine of the angle between the actual and predicted vectors. Comparative results across these loss functions offer insights into the model's sensitivity to different types of errors and its ability to handle variations in ECG data.

Outcomes and Findings:

The project generates visualizations, including sample plots for each loss function, to facilitate a comparative understanding of the model's performance. These plots, showcasing actual ECG data compared to the reconstructed data by the CNN autoencoder, are saved as HTML files for easy integration into the project. The visualizations enable a comprehensive assessment of how each loss function influences the model's ability to distinguish between normal and abnormal heart rhythms.

Conclusion:

The use of CNN-based Autoencoders, coupled with a thorough exploration of various loss functions, demonstrates a promising approach to anomaly detection in ECG data. The project highlights the importance of selecting an appropriate loss function based on the nature of the data and the desired model behavior. The insights gained contribute to the advancement of deep learning techniques for medical anomaly detection, particularly in cardiovascular health.

Part 2: Drowsiness Detection System

Introduction:

Accidents caused by driver fatigue constitute a significant proportion of traffic accidents, leading to the loss of thousands of lives, millions of hours in time, and billions of dollars in financial losses each year, solely in the US[1]. This makes it a highly crucial issue. There are various possible methods to minimize this problem according to our research. For instance, creating a prediction using machine learning models based on the path the car is following, thereby measuring the driver's driving style and reflexes. Other methods include monitoring the driver's pulse or tracking body symptoms using techniques like EEG. However, among these, a system employing Computer Vision for blink, yawn detection, and head estimation is likely to be the most useful and effective, as it directly monitors the driver's physical condition. Additionally, it

[1] Elizabeth Rivelli, "Drowsy Driving 2021 Facts & Statistics," Bankrate, last modified September 13, 2022, https://www.bankrate.com/insurance/car/drowsy-driving-statistics/#drowsy-driving-statistics

can be noted that the system is highly cost-efficient since it only requires a Raspberry Pi or Arduino-like system, a camera, and an alarm.

Methodology:

Step 1: To enhance the mentioned system, the first step would be to detect a person using the camera, followed by identifying the person's face and subsequently detecting their eyes and mouth. For this purpose, there is an excellent machine learning library called dlib, written in C. With dlib, it is possible to access real-time cropped images of the left eye, right eye, and mouth from the original image, facilitating the development of the desired features.

Step 2: At this point, since we have cropped images for the person's eyes and mouth, efforts should be focused on how to determine whether the person is drowsy from these images. The features used here are to serve these purposes as follows: **2.1) Blink detection** - the condition of the eyelids being closed over a certain period of time **2.2) Yawn detection** - the condition of the mouth being open for an extended period of time so that yawn can be detected **2.3) Head pose detection** - measuring the tilt of the head to right or left sides, which may indicate drowsiness since awake person would stand straight **2.4) Excessive yawning** within a certain time frame since yawning is a sign of tiredness and it will increase for drowsy persons **2.5) Reduced blink rate** within a certain time frame since the blink rate decreases in sleepy individuals. According to a study I found at this point, while the rate of blinking per minute is 10 in a normal person, it drops to 4-6 or less in a tired person[2].

[2] Arafat Islam et al., "A Study on Tiredness Assessment by Using Eye Blink Detection," Jurnal Kejuruteraan 31, no. 2 (2019)

Step 3: At this point, the next step should be measuring the level of fatigue in the individual. Our system has four status levels according to the priority level of the drivers' drowsiness status. It is completely determined using the features described in the previous part. It goes from a normal situation to the most emergent one in which an alarm is triggered with an increasing priority therefore the system checks the highest priority condition first and goes lowest ones. Here are the priority descriptions: **3.1)** The driver is perfectly alert, therefore no precaution is taken. In this case "You are okay" message is printed **3.2)** The driver shows signs of fatigue measured by having blinked less than the normal rate or yawned more than usual within a certain period of time. In this case "You look tired!" message is printed **3.3)** The driver is in a completely drowsy state, having yawned for an extended period or with eyes closed for a certain duration. In this case "You are drowsy!!" message is printed **3.4)** The driver is dozing off, keeping their eyes closed for an extended period to 3rd case or tilting their head to the sides over 20 degrees. In such cases, an alarm will be triggered to wake the driver and "Wake up!!!" message will be printed.

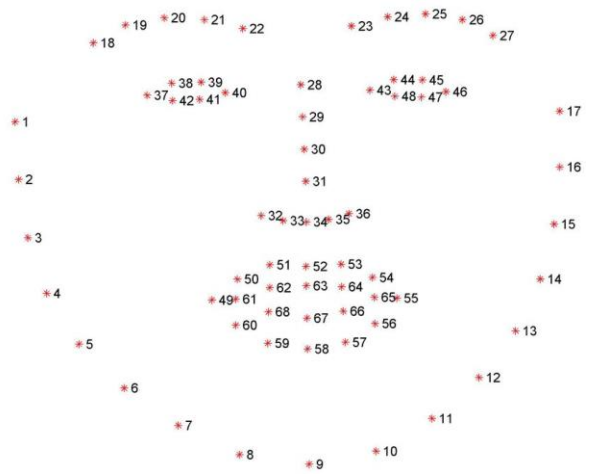
Step 4: The most crucial part is how to determine if the eyes are closed, the mouth is open, and the head is tilted to the sides actually. The most basic one comes to mind is using the Transfer Learning model by finding necessary eye and mouth datasets. This is a more flexible method since it only depends on the dataset that is trained and CNN model for it to have different features. There is also another method too which points certain landmarks in the face and measures the distance between these landmarks according to the mathematical specifications. This method is firstly developed in the article I found and I will describe more about it later. In the system, we aimed to enhance sensitivity by simultaneously using these two models, thereby increasing the recall value and decreasing false negatives even though it increases false positives and precision value. I also had a plan to develop my system for Raspberry Pi and I made some

experiments using it along with LED and buzzer but even though I tried I couldn't manage to successfully connect it and make it work without an error. Therefore my system is developed completely for my laptop using the laptop's camera and Jupyter Notebook. Here are detailed descriptions for the 2 methods I used as a final part of my report

4.1) Transfer Learning: The first method involves creating a model using Transfer Learning and using this model to make decisions. Kaggle datasets were utilized for this purpose for eyes[3] and mouth[4] and MobileNet was chosen as a lightweight Convolutional Neural Network (CNN). Then binary classification and Adam optimizer is implemented to create the model. This allowed the trained machine learning model to categorize in real-time using the camera, detecting whether the eyes are closed, the mouth is open, or the head is tilted.

4.2) Facial Landmarks: Additionally, I utilized a model suitable with the dlib library that treats every significant point on the face (a total of 68 points) as landmarks. This is very good feature for many applications in the face since with these facial landmarks, many activities in the face can be detected as well by comparing the distances between these facial landmarks. Accordingly, the right eye is represented between 36 and 41th marks while the left eye is between 42 and 47th marks for example. Here is the visual representation of the method:

[3] https://www.kaggle.com/datasets/tauilabdelilah/mrl-eye-dataset
[4] https://www.kaggle.com/datasets/davidvazquezcic/yawn-dataset



The method I created mathematically measures the distance between the eyes and lips to determine if they are open or not. I used the formula from the article I mentioned above for this purpose to calculate these distances. [5]

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

According to this model, surpassing a certain threshold (such as 0.25 for eyes or 0.6 for mouth) indicates blinking or yawning. Finally, the head pose estimation is obtained by calculating the angle between the eyes' centers (which I calculated by taking the average for 6 landmarks of each eye), and using these two points as a reference. So my code first calculates these 2 points and then it calculates the mathematical counterpart as a degree between these 2 points and if it's absolute value is more than 20, it gives an alarm. So that my code works successfully for each feature I mentioned in part 2 and part 3.

[5] Tereza Soukupova and Jan Cech, Machine Vision Laboratory, accessed January 19, 2024, https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf.