Drowsiness Detection

1st Hsin-Hung Wu
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ
hwu50@stevens.edu

2nd Monish Kanna Suresh
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ
msuresh1@stevens.edu

3rd Aman Sandal
Department of Mathematical Sciences
Stevens Institute of Technology
Hoboken, NJ
asandal@stevens.edu

***Abstract*—Drowsiness is the root cause of majority of road accidents. Drowsiness detection is a car safety system that can alert a snoozing driver in hopes of preventing an accident. Various drowsiness detection methods already exist to monitor drivers' drowsiness and alarm them if they are not concentrating on driving. Some of the latest cars even come equipped with such safety features which notify the driver for small breaks during long drives. Levels of tiredness can be inferred from information extracted from a variety of facial characteristics such as mouth and eyes. We aim to build a real-time drowsiness detection system to reduce car accidents by keeping track of drivers' eyes and alerting them when they are detected as drowsy. By implementing and bench-marking various machine learning models, we are able to come up with a suitable algorithm for drowsiness detection.**

I. Introduction

Most drivers understand the dangers of drinking and driving, which is in no doubt a dangerous task to carry out and almost everyone is aware of the mishappenings caused by texting while driving. However, most people underestimate the dangers of drowsy driving. Drowsy driving refers to the act of driving a vehicle while being tired, fatigued, or sleepy. Driving for lengthy periods of time without rest can lead to accidents. Tiredness hugely impacts the attentiveness of the driver. The driver's decision-making ability is labored, which may result in road accidents. According to the National Safety Council (NSC), drowsy driving accounts for about 100,000 crashes, 71,000 injuries, and 1,550 fatalities each year. Most of the accidents caused due to drowsy driving can be prevented if the driver is alerted in a split second with a system that detects the drowsiness of the driver and alerts the driver to take some rest. [1]

Humans experience drowsiness frequently between midnight and 6 a.m., or in the late afternoon. At both times of the day, people experience dips in their circadian rhythm—the human body's internal clock that regulates sleep. Irrespective of the time of the day, we aim to reduce accidents caused by drowsiness by building the most suited classifier for drowsiness detection from real-time video input and alerting them as necessary. In order to find the most suited classifier, we built five different machine learning algorithms, convolutional neural network, support vector machine with two different kernals, k-nearest neighbors, decision tree, and random forest, and compared their performance to find the one that best suit our task.

II. Related Work

Machine learning allows computers to learn without having to be explicitly programmed. The process of machine learning is comparable to that of data mining. Both systems filter through information in search of feature patterns. Instead of extracting data for manual operation, the machine learns from it, detects hidden patterns, and adjusts kernel ram operations in accordance with the objective function. Machine learning algorithms are frequently classified as either supervised or unsupervised. During training, supervised algorithms learn from a complete collection of labeled data. And there are two types of supervised learning: classification and regression. Unsupervised algorithms, on the other hand, deal with a dataset without specific instructions on what to do with it.

In order to achieve better accuracy in the detection of driver drowsiness systems, many approaches were developed. Mardi et al., [2] have proposed a model to detect drowsiness based on Electroencephalography (EEG) signals. Extracted chaotic features and logarithm of energy of signal are extracted for differentiating drowsiness and alertness. An artificial neural network was used for classification and yielded 83.3% accuracy.

Krajewski et al., [3] developed a model to find drowsiness based on steering patterns. In this model, to capture the steering patterns, they generated three feature sets by using advanced signal processing methods. Performance is evaluated using five machine learning algorithms like SVM and K-Nearest Neighbor and achieved an accuracy of 86% in detecting drowsiness.

Danisman et al. [4] developed a method to detect drowsiness based on changes in eye blink rate. Here Viola Jones detection algorithm was used to detect face region from the images. Then neural network-based eye detector was used to find the location of the pupils. Calculated the no of blinks per minute if the blinks increase, indicated that the driver becomes drowsy.

Abtahi et al. [5] presented a method for drowsiness detection through yawning. In this method, first face was detected then the eye and mouth regions were detected. In the mouth, they calculated the hole as a result of the wide mouth open. The face with largest hole indicates a yawning mouth.

Dwivedi et al. [6] developed a model to find the drowsiness using CNNs. In this method convolutional neural network was used to capture the latent features then SoftMax layer was used for classification and yielded 78% of accuracy.

An Eye-tracking based driver drowsiness system was proposed by Said et al. [7] . In this work, the system finds the driver's drowsiness and rings the alarm to alert the driver. In this work, Viola Jones model was used to detect the face region

and eye region. It has produced an accuracy of 82% in indoor tests and an accuracy of 72.8% for the outdoor environment.

III. Our Solution

A. Description of Dataset

The dataset we used is from two different Kaggle sources and consists of open and closed eyes images, as we believe eyes are the best indicator of drowsiness. We have 4000 training data [8] with 80/20 validation split and 1452 testing data [9]. For each image prior to training, we normalized, centered, standardized, and resized it to 50×50 in order to improve training performance, efficiency, and model stability. For each image prior to testing, we normalized, resized, and converted it to grayscale in order to match the training data set, as these data came from two different sources. Below in figure 1 are some sample data for our training.

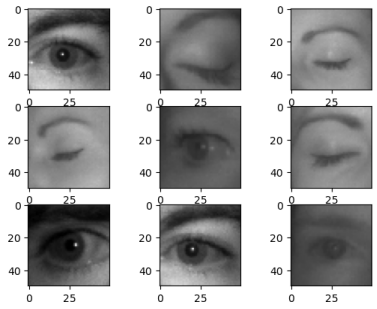

Fig. 1. Sample Data

B. Machine Learning Algorithms

- Convolutional Neural Network (CNN)
- Support Vector Machine (SVM)
- K-Nearest Neighbours (KNN)
- Decision Trees
- Random Forest Classifier

Convolutional Neural Network (CNN) is a deep learning algorithm, and a class of artificial neural networks that is specifically designed to analyze pixel data and is most commonly used for image recognition tasks as it is effective in reducing the dimension and the number of parameters without lose of information.

Support Vector Machine (SVM) is a classification algorithm that draws a decision boundary that separates two classes and is great for our binary image classification task as it allows us to use kernel tricks to project non-linearly separable data into higher dimension space. It has lots of applications, such as web pages, intrusion detection, face identification, email categorization, gene classification, handwriting recognition, and etc. SVM allows us to uncover intricate connections in the data without much manual manipulation. When working with smaller dataset with tens of thousands to hundreds of thousands of features, it's a great alternative as it excels in dealing with small and complex dataset [10].

K-Nearest Neighbors (KNN) algorithm is a data classification method for estimating the likelihood that a data point will become a member of one group or another based on what group the data points nearest to it belong to. It is a type of supervised machine learning algorithm used to solve classification and regression problems, but is most commonly used as a classifier. In short, KNN classifies a new data point by looking at its nearest data points.

The decision tree is a flowchart-like structure consisting of a hierarchy of nodes with a tree-like model that can be used for both supervised learning and unsupervised learning to make decisions throughout the branches of the tree. Decision trees use various algorithms to split a dataset into homogeneous sub-nodes [11].

Random Forest classification algorithm utilizes ensemble learning, which is a technique that combines various decision trees to provide solutions to complex problems. By developing multiple decision trees and considering the majority of the output values from these decision trees, we have a more robust model which gives better classification. Figure 2 shows a simple random forest classifier.

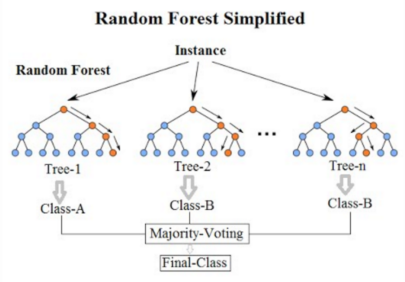

Fig. 2. Working of Random Forest algorithm

C. Implementation Details

We will now discuss the implementations of the machine learning algorithms.

For CNN, prior to training, we resized the images to 50 by 50 so it decreases our training time, standard normalized the images which allows different features to be trained on a similar scale, zero centered the images since we are using ReLu and Sigmoid activation functions and it prevents gradient saturation. Finally we randomly shuffle the data and split them into 80% training and 20% validation.

We trained the model from scratch with the CNN architecture from figure 3 with one input layer, 11 hidden layers, and one output layer. We used Relu for the hidden layers and Sigmoid for the output layer. ReLu allows our model to learn faster and prevent the vanishing gradient problem. The Sigmoid function is a great fit for our task as we are working with a binary classification which can be represented as a probability. The initial parameters have 10 batch size, 100 epochs, and 0.0001 learning rate, which were tuned throughout training. The model is trained with Adam optimizer, an extended version of stochastic gradient descent, and binary cross entropy loss function.

In the process of training and optimizing the CNN model [12], we realized that our initial architecture and hyperparameters led to over-fitting, and it also took too long to train

without GPU clusters. We, therefore, changed the architecture by tuning the hidden layers and reducing the number of hidden nodes. We applied batch normalization after each convolutional layers to decrease over-fitting, standardize the input, and speed up training. We applied max pooling to reduce the number of parameters we needed to learn and reduce over-fitting. We tuned the hyper-parameters and reduced the learning rates to 0.00001, as a higher learning rate tended to overshoot and skipped the optimal solution. For the batch size and epoch, we tuned them by trying different combinations of the learning rate, batch size, and epoch, as they all affect each other. Ultimately, we chose a batch size of 32 and an epoch of 30, which works best for our model.

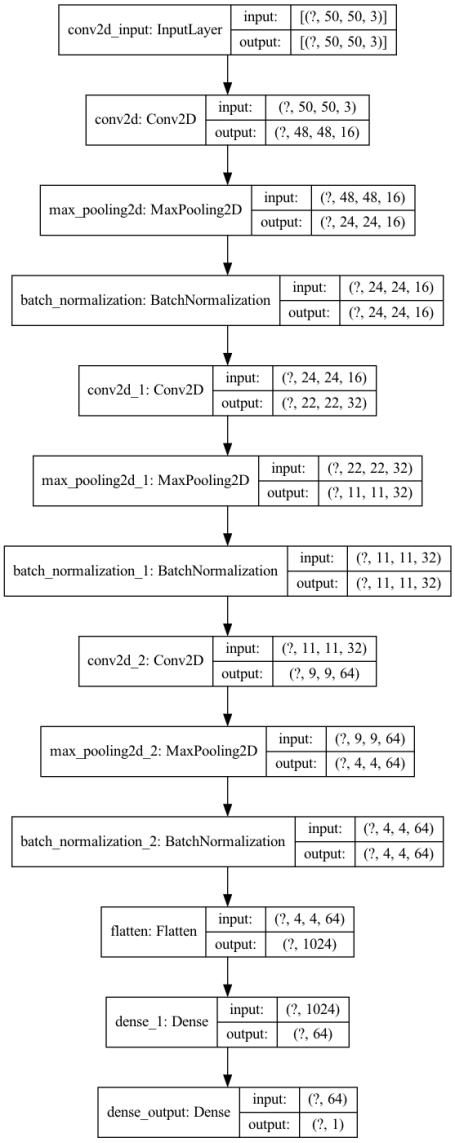

Fig. 3. CNN Architecture

From figure 4 and 5, we can see that the accuracy and loss improve as we train, and we are able to achieve 99.91% training accuracy and 96.25% validation accuracy at the end of training. We tried to reduce the false positive rate, as not alerting someone who is drowsy is way more consequential than alerting someone who is awake.

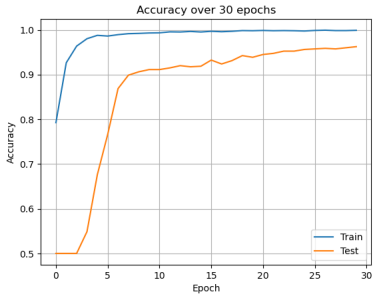

Fig. 4. CNN Training Accuracy

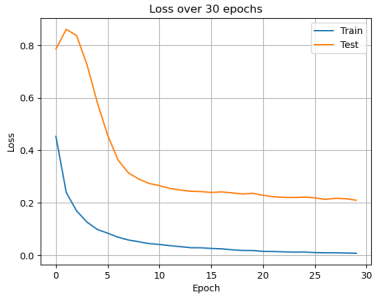

Fig. 5. CNN Training Loss

For SVM, there are two varieties, each with specific applications. The first is simple SVM, which usually applies to classification and linear regression problems. Second is kernel SVM, which allows us to add more features to suit a hyperplane rather than a simple two-dimensional space. It gives us greater flexibility for non-linear data.

We implemented two types of SVM. the first one is linear, and the idea is that the algorithm creates a line or a hyperplane which separates the data into classes. In other words, the data points on one side of the line will all be assigned to one category, while the data points on the other side of the line will be assigned to a different category. This implies that the number of possible lines is unlimited. An example of Linear SVM is shown in figure 6.

Because it selects the optimal line to categorize your data points, the linear SVM algorithm is superior to some other algorithms, such as k-nearest neighbors. It selects the line that divides the data that is far from the nearest data points.

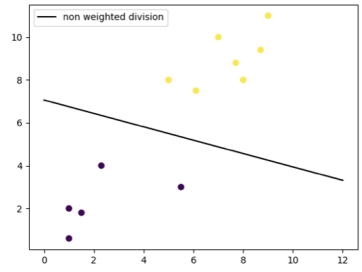

Fig. 6. SVM Linear

The other SVM implementation, the Radial kernel support vector machine, is a good approach when the data is not linearly separable. The idea behind generating non-linear

decision boundaries is that we need to do some non-linear transformations on the features, which transforms them into a higher dimensional space, as we can see from figure 7.

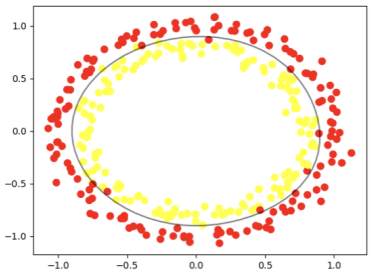

Fig. 7. SVM Non-Linear

In the process of training and optimizing the SVM model, we tried out both the linear and radial basis function kernels. The former has 80.4% and the latter has 76.1% accuracy.

For KNN, during implementation, we mainly focused on picking the best K as it is a crucial parameter in the algorithm. As we decrease K too much, there are overfitting and high variance issues. As we increase K too much, we can see that after a certain K value, there are underfitting, bias issues, and hight test error. The initial test data error is high due to variance, then it decreases and stabilizes, and lastly again increases due to bias with further increase in K value. The K value when test error are low and stabilizes is considered as the optimal value for K.

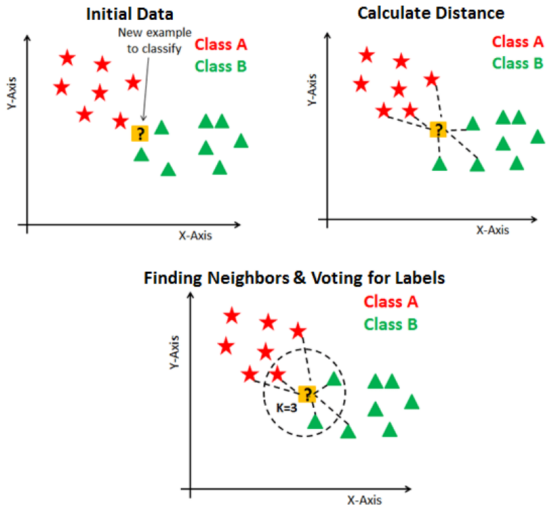

Fig. 8. Working of KNN algorithm

In the process of training and optimizing the KNN model, we tried out multiple neighbors from $1-15$ and found that there isn't much improvement in training and testing accuracy as we increased the number of neighbors beyond 2, figure 9, so we chose 2 as the number of KNN neighbors and we are able to achieve an accuracy of 67.4%.

For the Decision Tree, during implementation, the main challenge is to select the best attribute for the root node and sub-nodes. In order to solve such problems, there is a technique called attribute selection measure or ASM. With ASM, we can

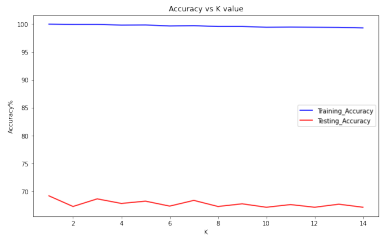

Fig. 9. KNN Accuracy for different N

easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are Information Gain and Gini Index.

In the process of training and optimizing the Decision Tree, we tried out multiple depths from $1-10$, figure 10, and found that with the depth of 3, we are able to achieve 68.06% testing accuracy.

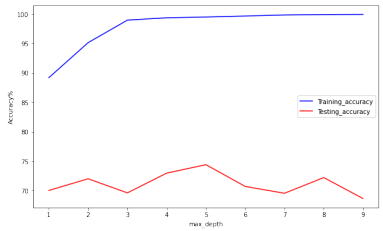

Fig. 10. Decision Tree Different Depth

For the Random Forest, it combines various decision trees to provide a more robust model than one decision tree alone and we are able to obtain 76.74% accuracy.

For the overall drowsiness detection, we piped lined the entire process from retrieving the driver's real-time video input to alerting the driver if they are drowsy altogether. We first extract our region of interest, which are the eyes from the face, as we can see from figure 11, with a pre-trained Cascade Classifier taken from OpenCV [13]. We then process the eye images and feed them into our trained CNN model to detect whether the driver is drowsy or not. The drivers will then get alerted if they are detected as drowsy for 3 seconds. Something we struggled with at first was the extraction of eyes from the face as the Cascade Classifier often misclassifies the mouth as the eyes. We solve this problem by only detecting features that are in the upper half of the face.

IV. Comparison

We implemented convolutional neural network, support vector machines, k-nearest neighbors, decision tree, and random forest for our drowsiness detection task. In this section, we will discuss and compare their performances. For performance measurements, we used binary accuracy, confusion matrix, and ROC curve, which measures how often the predictions match the labels and also how well the model distinguishes between the two classes. We will focus not just on the accuracy, but most importantly, the false positive rate, as not alerting someone who is drowsy is way more consequential than alerting someone who is awake.

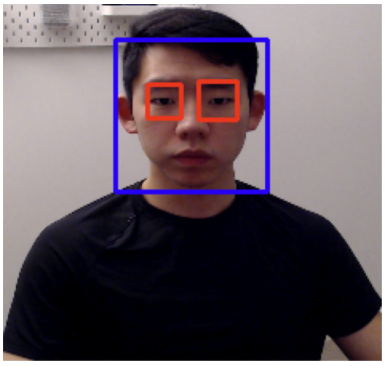

Fig. 11. Real-Time Eyes Detection

For CNN, we are able to achieve 88.5% accuracy on our testing data set. From figure 12 and 13, the confusion matrix and the ROC curve shows us that the model distinguishes between the two classes well and has drastically reduced the false positive and false negative rates. It has sensitivity of 88.55%, specificity of 88.45%, precision of 88.43%, and F1 Score of 88.49%.

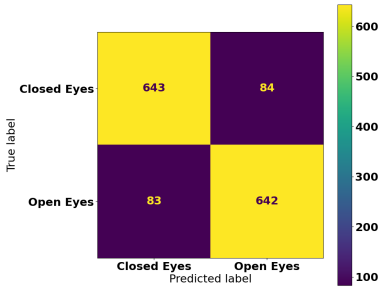

Fig. 12. CNN Confusion Matrix

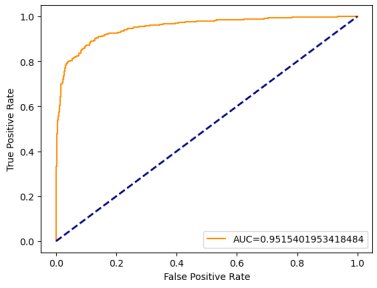

Fig. 13. CNN ROC Curve

For SVM with the linear kernel, it has 80.4% accuracy with the confusion matrix and ROC curve in figure 14 and 15. It has sensitivity of 81.37%, specificity of 79.44%, precision of 78.79%, and F1 Score of 80.06%.

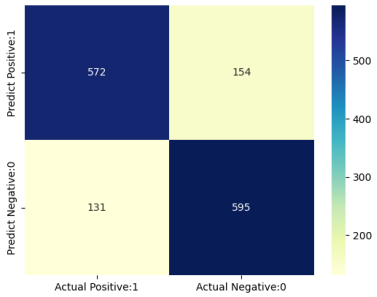

Fig. 14. SVM Linear Confusion

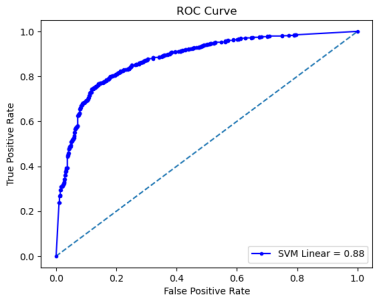

Fig. 15. SVM Linear ROC

For SVM with the radial basis function kernel, it has 76.1% accuracy with the confusion matrix and ROC curve in figure 16 and 17. It has sensitivity of 83.98%, specificity of 71.03%, precision of 64.81%, and F1 score of 73.16%.

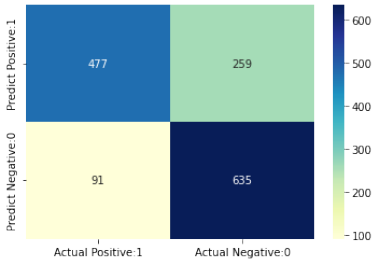

Fig. 16. SVM RBF Confusion

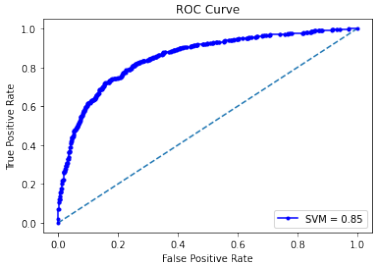

Fig. 17. SVM RBF ROC

For KNN, we have testing accuracy of 67.4% with the confusion matrix and ROC curve shown in figure 18 and 19. It has sensitivity of 63.44%, specificity of 74.7%, precision of 82.74%, and F1 score of 71.82%.

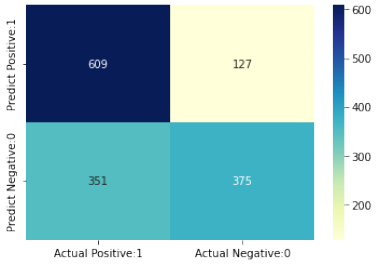

Fig. 18. KNN Confusion Matrix

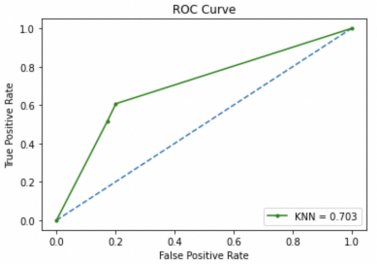

Fig. 19. KNN ROC

For Decision Tree, we have testing accuracy of 68.06% with the confusion matrix and ROC curve shown in figure 20 and 21. It has sensitivity of 69.75%, specificity of 66.58%, precision of 64.54%, and F1 score of 67.04%.

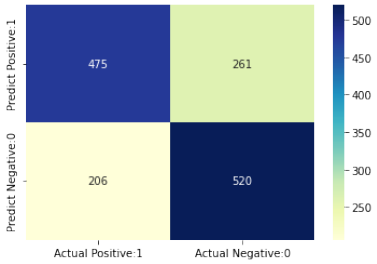

Fig. 20. Decision Tree Confusion

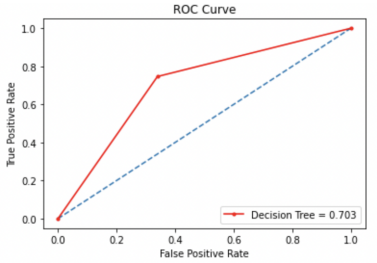

Fig. 21. Decision Tree ROC

For Random Forest, we have testing accuracy of 76.74% with the confusion matrix and ROC curve shown in figure 22 and 23. It has sensitivity of 80.94%, specificity of 73.48%, precision of 70.38%, and F1 score of 75.29%.

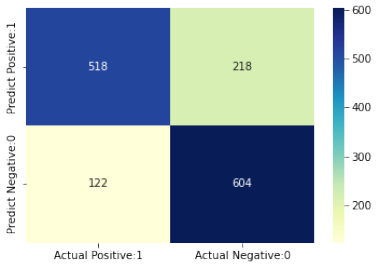

Fig. 22. Random Forest Confusion

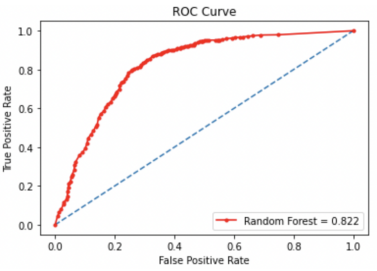

Fig. 23. Random Forest ROC

Overall, in terms of accuracy, CNN outperforms the rest of the algorithm with 88.5% test accuracy whereas SVM, KNN, Decision Tree, and Random Forest have corresponding test accuracy of 80.4%, 67.4%, 68.06%, and 76.74% as we can see from figure 24.

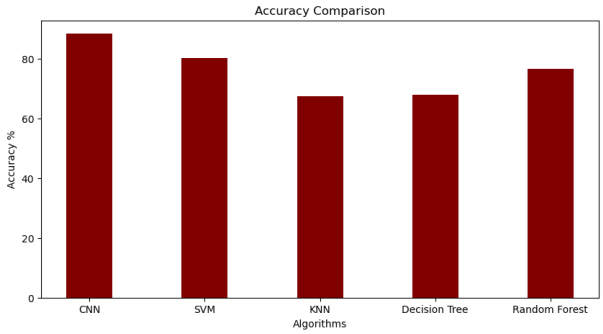

Fig. 24. Accuracy Comparison

In terms of precision, CNN once again outperforms the rest of the algorithm with precision of 88.43% whereas SVM, KNN, Decision Tree, and Random Forest have corresponding precision of 78.79%, 82.74%, 64.54%, and 70.38% as we can see from figure 25.

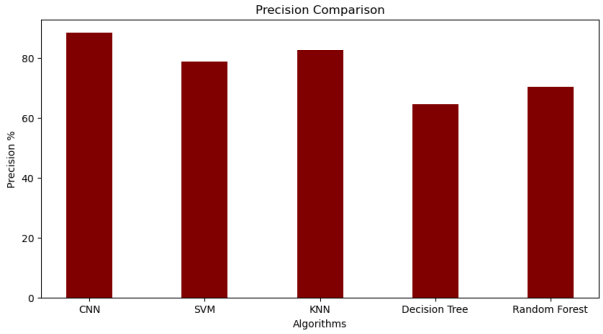

Fig. 25. Precision Comparison

CNN has AUC of 95% whereas SVM, KNN, Decision Tree, and Random Forest have corresponding AUC of 88%, 70.3%, 68.1%, and 68.1% as we can see from figure 26.

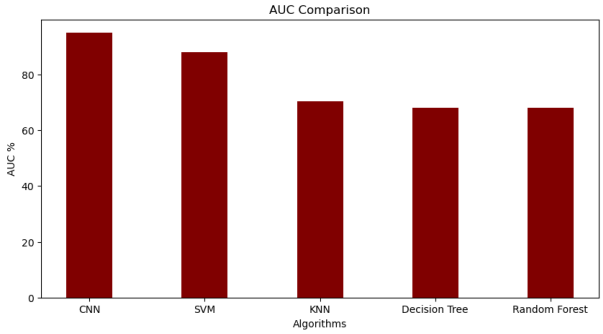

Fig. 26. AUC Comparison

CNN stands out among all the algorithms we implemented because it is primarily suited for image recognition tasks as it is able to reduce high dimensional images without loss of information and store lots of information as parameters. We are also able to provide large enough training data sets as it requires.

The runner-up, SVM, also provides good performance, but it still lacks behind CNN. SVM works better when there is a clear margin of separation between the two classes and target classes are not overlapping, which is not our case. We also have a large data set, which increases the training complexity.

For KNN, it is a really simple algorithm for image classification tasks as it simply classifies unknown data through the most common class among the k-nearest existing data. However, it is expensive for large data sets, and it has difficulty dealing with high dimensionality. It has poorer accuracy, but high precision, which is good for drowsiness detection as we favor a low false positive rate.

For Decision Tree, it performs a lot worse than CNN in terms of both accuracy and precision as it doesn't take spatial locality between features into account, is expensive for large data sets, and tends to overfit with high dimensional data sets.

For Random Forest, it is better than the Decision Tree as it is more robust, but it is still worse than CNN for image classification for a similar reason as the decision tree.

V. Future Directions

We were able to achieve 88.5% testing accuracy with the CNN model which is good with the given amount of time and resources we had. However, there are still lots of improvement and extension that can be made.

First, we are only using eyes as drowsiness predictor, which works well, but not optimal. There are more indicators of drowsiness in addition to eyes. For examples, the driver can be yawning, which makes mouth a good predictor for drowsiness. The driver can be dozing off with the head rocking back and forward, which makes unusual head movement a good predictor for drowsiness. The driver can have lack of eye ball movement due to zoning out and an eye ball tracker in this case can predict drowsiness.

Second, we can improve our model performance with more data set and computing resources. We had to downsize our images since we didn't have the time and resource to train from scratch, which caused loss of information. The model can retain more image information and better performance with less downsizing.

Lastly, we would like to integrate the model into a car camera system and see how it performs with different camera angles and with the possibility of the camera falsely detecting the passenger in the backseats. For this case, we need to work with camera depth to only detect the driver.

VI. Conclusion

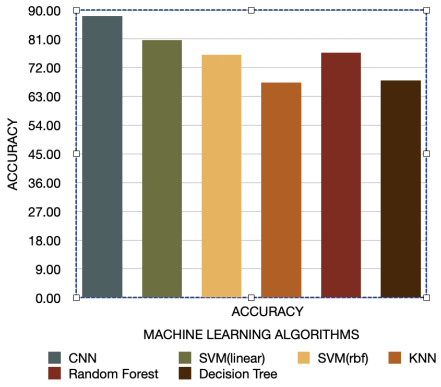

Fig. 27. Machine Learning Algorithms

After the implementations of all five machine learning algorithms, we noticed that Convolutional Neural Network (CNN) gives us the best accuracy, precision, and efficiency on drowsiness detection. The accuracy achieved with the CNN algorithm is 88.25 percent and figure 27 gives us an overview of how all algorithms perform.

References

[1] M. Ramzan, H. U. Khan, S. M. Awan, A. Ismail, M. Ilyas, and A. Mahmood, "A survey on state-of-the-art drowsiness detection techniques," *IEEE Access*, vol. 7, pp. 61 904–61 919, 2019.

[2] Z. Mardi, S. N. M. Ashtiani, and M. Mikaili, "Eeg-based drowsiness detection for safe driving using chaotic features and statistical tests." [Online]. Available: https://doi.org/10.4103/2228-7477.95297

[3] J. Krajewski, D. Sommer, U. Trutschel, D. Edwards, and M. Golz, "Steering wheel behavior based estimation of fatigue," Jun 2009. [Online]. Available: https://doi.org/10.17077/drivingassessment.1311

[4] Danisman, Ihaddadene, Djeraba, and Bilasco, "Drowsy driver detection system using eye blink patterns," 2014. [Online]. Available: https://doi.org/10.1109/icmwi.2010.5648121

[5] A. S, H. B, and S. S, "Driver drowsiness monitoring based on yawning detection," 2011. [Online]. Available: https://doi.org/10.1109/imtc.2011.5944101

[6] D. K, B. K, and S. A, "Drowsy driver detection using representation learning," 2014. [Online]. Available: https://doi.org/10.1109/iadcc.2014.6779459

[7] "Real time eye tracking and detection- a driving assistance system," Aug 2022. [Online]. Available: https://doi.org/10.25046/aj030653

[8] P. V. Patil, "Drowsiness detection dataset," Dec 2020. [Online]. Available: https://www.kaggle.com/datasets/prasadvpatil/mrl-dataset

[9] D. Perumandla, "Drowsiness dataset," Sep 2020. [Online]. Available: https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset

[10] [Online]. Available: https://course.ccs.neu.edu/cs5100f11/resources/jakkula.pdf

[11] J.-C. C. S. a. Tripadvisor, "Decision trees in machine learning, with examples (python)." [Online]. Available: https://www.jcchouinard.com/decision-trees-in-machine-learning/

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[13] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.