Driver Drowsiness Detection with Deep Learning and Computer Vision

Andreas Wild
Stellenbosch University
Department of Applied Mathematics
24991104@sun.ac.za

Supervisor: Dr H Coetzer

Abstract

In this paper a system that automatically detects driver drowsiness is proposed. Yearly over 100,000 accidents in the United States involve a fatigued driver. Historically this problem has been solved with tedious physiological sensors or ambiguous vehicle monitoring systems. In this report the driver alertness is determined through deep learning techniques. A custom convolutional neural network that determines driver drowsiness based on the eyes of an individual is trained. The proposed model is less than one megabyte in size and detects driver fatigue in real-time. This allows the system to run on low level hardware and provide alerts to the driver before an accident can occur. The use of this model would reduce the number of drowsiness associated accidents on the road.

1 Introduction

This paper discusses the implementation of a driver drowsiness detection system. Deep learning and computer vision techniques are applied to determine whether a driver is fatigued or not. A fully automatic model is developed that, given video input, alerts the driver in real time if drowsiness is detected. The system automatically isolates the eye regions of a person and transforms these eye regions into feature vectors for classification with a self-trained convolutional neural network (CNN). The CNN returns an output from which the level of fatigue is deduced.

The necessary background information to reproduce the system is provided. The implementation methodology is provided thereafter. The training procedure and architecture tuning are carefully documented. This project is built in Python and utilises large open source machine learning (ML) libraries. The model training is performed through NVIDIA's state-of-the-art CUDA architecture with the TensorFlow Keras framework. These frameworks allow the user to rapidly test various neural network (NN) architectures due to fast training times.

Section 4 discusses possible bottlenecks in the system and provides results. The proposed model achieves a high accuracy of 98% on more than 10,000 test images, while the full project is less than one megabyte in size. The model is a real-time detection system and should run on low level hardware. For this reason, the accuracy is not the only benchmark metric but the speed and size of the system are also considered. Finally, a conclusion with closing remarks and future improvements to the system is given.

2 Background

All the necessary background information needed to implement the project from scratch is discussed here.

2.1 Problem History

Driver drowsiness is a critical safety concern that significantly contributes to road accidents worldwide. In the United States research by the national highway traffic safety administration (NHTSA) reports that annually approximately 100,000 road accidents occur due to driver drowsiness [11]. These incidents result in 72,000 injuries and 800 deaths each year. In South Africa the situation is equally as concerning. Almost a quarter of all heavy-duty vehicle accidents have a drowsy driver involved [8].

In literature there have been three approaches to the drowsy driver problem. The first is the use of physical sensors to monitor the fatigue levels of the driver. These sensors use techniques from electroencephalography (EEG), electrooculography (EOG) and electrocardiography (ECG) to monitor brain, eye and heart activity respectively. Ensemble systems have been proposed that combine two or more of these techniques. The model developed by Noori *et al.* [10] combines EEG and EOG information to infer the drowsiness level. Although accurate, these systems are generally impractical, since their set up is tedious and requires some domain knowledge. Users might not attach the sensor correctly or at all.

Another approach is to investigate the vehicle instead of the driver. Sudden and sporadic changes in direction are recorded and is known as the steering wheel movement (SWM) technique. Motor companies Nissan and Renault have implemented SWM systems [15]. In this approach, inaccuracies are a result of the driving skills of an individual and the quality or geometry of the road. This approach sees little implementation in practice and in literature.

Lastly, recent advancements in computer vision and deep learning have opened new avenues for addressing this issue. The basis of this approach is to take video input of the driver and perform feature extraction. These features are used as input for a deep learning model that determines whether the driver is fatigued or not. The model from Jabbar *et al.* [5] is such an example. The system developed by Jabbar focuses on creating a real-time drowsiness detection system for Android applications using deep neural networks (DNN). Their model places an emphasis on mobile platform compatibility and accessibility. The most common features that are used to determine fatigue are: whether the individual's

eyes are open or closed, the presence of frequent yawning and lastly head tilt.

2.2 Neural Network History

The earliest NN research dates back to the 1940s. McCulloch and Pits introduced the first computational model of how the human brain might work in their famous paper "A logical calculus of the ideas immanent in nervous activity" [9]. This model is very simple compared to what came in the following years. The McCulloch-Pits neuron is a simple threshold-based system. Another revolutionary development came 15 years after the work of McCulloch and Pits. In 1958 American psychologist Frank Rosenblatt developed the perceptron [13], which is capable of solving linearly separable problems. Researchers' excitement was short lived. It was shown that a singular perceptron is unable to solve even the simplest non-linear problems due to linear activation functions. Research in the field began to stagnate. Luckily, the influential work of Hinton *et al.* [14] sparked new interests among researchers. Hinton, Rumelhart and Williams developed the famous backpropagation algorithm now synonymous with NN literature. They showed that multi-layer perceptrons were, in fact, able to learn complex relationships in data by back-propagating errors through layers in the training process. Provided that the models are large enough and use non-linear activation functions. The discussion of different artificial intelligence (AI) algorithms can become extremely convoluted. A custom Venn diagram is included for clarity. CNNs are by definition DNNs as seen in Figure. 1 .

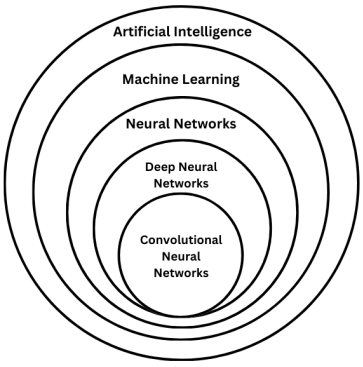


Figure 1: Artificial intelligence Venn diagram.

The turn of the millennium saw a rapid improvement in computational power that would continue throughout the 21st century. Powerful graphics processing units (GPU) and large datasets would ignite an entirely new field of research, deep learning. Up to this point NNs have been small in size compared to DNNs. The key difference between DNNs and normal NNs is their size and more importantly the number of hidden layers. In 2012 researchers would see another breakthrough development in the computer vision field. Ilya Sutskever[1] *et al.* [7] developed the CNN image classifier AlexNet that would blow previous technologies out of the water. This model was developed to classify more than 14 million images as one of 22,000 classes from the ImageNet dataset. This sparked a wave of research and innovation in the computer vision and AI fields. Sutskever would later go on to co-found the notorious AI company Open AI.

[1]At the time a PhD student under the 2024 physics laureate Geoffrey Hinton.

2.3 Convolutional Neural Networks

CNNs are a special types of DNNs that excel at processing structured grid-like data. The success of CNNs in the context of image recognition can be attributed to two fundamental assumptions.

1. **Local connectivity** assumes that nearby pixels in an image are more related than distant pixels. Local pixels may be bunched together and entire sub-regions may be considered to extract meaningful features.

2. **Translation invariance** assumes that useful features likely occur in multiple locations across an image. The same parameters may be shared across the entire input image. CNNs recognise features regardless of their position in the image.

The first NN that introduced shared parameters was the Neocognitron, developed by Fukushima [2] in 1980. The Neocognitron laid the foundation for CNN architecture, although modern CNNs are far more advanced. The basic CNN typically has four types of network layers. In convolutional layers, kernels are applied across the input data. Kernels are small matrix-like filters that detect a particular feature in the input. Mathematically convolution is defined as

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n) \tag{1}$$

at position $(i,\ j)$ for input I and kernel K. The kernels in this layer produce feature maps that represent locations where certain features are detected. After convolution a non-linear activation layer is introduced. This layer enables the model to learn complex relationships in the data. The third layer type is a pooling layer. The purpose of these layers is to reduce the dimensions of the feature map produced by the previous two layers. Pooling is performed by either taking the maximum or average value from a pool. A pool is simply a $n \times n$ window that is moved over the feature map. One might opt to skip certain indices in the feature map with some specified stride to further reduce dimensionality. The result reduces the feature map size by disregarding less significant values. These three layers form a convolutional stage.

Finally, a dense layer combines the features learned by the network after one or more convolutional stages. This layer is fully connected. Each neuron in the previous layer is connected to every neuron in the current layer. Fully connected layers share no parameters and greatly increase the size of the model. For multi-class classification tasks the final layer typically feeds into a softmax layer that returns a probability distribution over the possible classes. For binary classification tasks, the sigmoid function is common practise. A regularisation technique popular in DNNs is the use of dropout layers. During training the dropout layer randomly cancels the connection between neurons with some specified probability. This forces the model to become less reliant on specific neurons and encourages it to learn more generalised patterns. During testing all neurons are used and their outputs are scaled to account for the dropout applied during training.

2.4 Training and Accuracy Metrics

Here data preprocessing and the model training procedure is discussed. The necessary background information on benchmark metrics is also provided.

2.4.1 Data Preprocessing

Data preprocessing is an important step in the training process of a NN. Transformation of the data, in a way that is easy for the NN to use, speeds up convergence and may improve the generalisation ability of the model. In the context of CNNs the effect of common data quality issues is lessened. The CNN is relied on to learn appropriate kernels for the data. The kernels inherently perform image processing while the correct functional mapping between the input and output space is learned. Strictly speaking, the CNN learns the appropriate kernels that produce a feature map that contains the relevant information instead of performing image processing. As an example, in past systems feature vectors had to be extracted by clever edge detection or clustering methods. The modern approach relies on the CNN to extract these features automatically.

CNNs have the ability to perform many preprocessing steps automatically but one technique that will be used is standardisation. The z-score of data point x is given by

$$z = \frac{x - \mu}{\sigma} \tag{2}$$

for mean μ and standard deviation σ. Z-score normalisation may be performed globally or in batches and transforms the features of a dataset to have a mean of zero and a standard deviation of one. Many ML algorithms like NNs are sensitive to the scale of input features. If input features vary in range, the model may assign a larger weight to larger inputs although they are not inherently more important. In the context of images, standardisation adjusts the image brightness and contrast to be closer to the mean image in the dataset.

2.4.2 Model training

A popular training algorithm for NNs is the adaptive moment estimation (ADAM) optimiser shown in Algorithm 1. This optimisation algorithm was first introduced by Kingma and Ba [6] in 2014. The ADAM optimiser combines ideas from the root mean square propagation (RMSPROP)[3] and stochastic gradient decent with momentum (SGDM) optimisers [14]. The ADAM optimiser changes the learning rate α for parameters θ based on the moments of the gradients $\hat{m}_t$ and $\hat{v}_t$. The adaptive learning rate allows for good convergence in noisy problems. The computation is also memory efficient because only two additional variables are required per parameter computation.

In order to train a NN some measure of how well the model is currently performing is required. Such an estimate is obtained with the objective function L. This paper only discusses the binary cross-entropy (BCE) objective function although many different functions may be chosen. The BCE objective function lends itself well to binary classification

Algorithm 1 Adam Optimization Algorithm

Require: Learning rate α, exponential decay rates for moment estimates β_1, β_2, small constant ϵ for numerical stability, initial weights θ_0 and objective function $L(\theta)$

Initialize $m_0 = 0$, $v_0 = 0$, and $t = 0$
while not converged **do**
  $t \leftarrow t + 1$
  Compute gradient $g_t = \nabla_\theta L(\theta_{t-1})$
  Update biased first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  Update biased second raw moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  Compute bias-corrected first moment estimate: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
  Compute bias-corrected second moment estimate: $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  Update parameters: $\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$
end while
return θ_t

problems. In the context of driver drowsiness detection there are only two classes, drowsy or not drowsy. The BCE loss for a dataset with N samples is defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N} (y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)) \tag{3}$$

with true class label y and predicted class label $\hat{y}$.

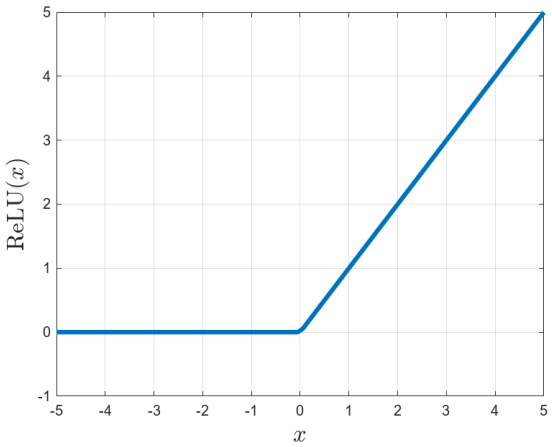
Activation functions are used in NNs to introduce non-linearity and enable a model to learn complex relationships. The rectified linear unit (ReLU) activation function is widely used in modern DNNs. The ReLU function is defined as

$$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x}) \tag{4}$$

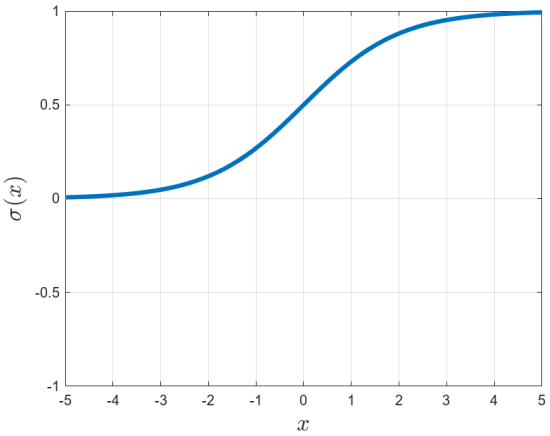
and was first introduced by Fukushima [1] in 1969 long before the rise of deep learning. ReLU is a popular choice for DNNs because the gradient is equal to one for all positive inputs regardless of the input size. This mitigates the vanishing gradient problem [4] that is present in large DNNs. In other popular activation functions like the sigmoid function, the gradient decreases as the magnitude of the input increases. Figure 2 shows both activation functions on the domain $x \in [-5, 5]$. The sigmoid function is defined as

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{5}$$

and asymptotically approaches zero and one for large negative and positive inputs respectively. The weights of the NN are updated based on the gradient of the activation functions. For large input values the gradient approaches zero. This does not render the sigmoid activation function useless. The sigmoid activation function is commonly used in the context of binary classification problems as it outputs a value between zero and one. The function is ideal for the use in decision boundaries since it has a y-intercept at 0.5. The output may interpreted as a confidence percentage that an instance belongs to a specific class. If the output is greater than 0.5 the class one is assigned, otherwise the instance is assigned class zero.



(a) ReLU activation function.



(b) Sigmoid activation function.

Figure 2

2.4.3 Accuracy metrics

In this paper four different accuracy metrics applicable to classification problems are discussed.

1. **Accuracy**:
   Accuracy calculates the ratio of correct predictions to the total number of predictions and represents the overall percentage of correctly classified instances. Accuracy is calculated as
   $$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}, \tag{6}$$
   where true positives indicate the number of instances where the model correctly predicted the positive class and false positives indicate the number of instances where the model incorrectly predicted the positive class. A similar definition follows for true negatives and false negatives.

2. **Precision**:
   Precision is the ratio of correctly predicted positive instances to the total instances predicted as positive.
   $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{7}$$
   Precision is useful when the cost of false positives is high.

3. **Recall**:
   Recall measures the ability of the model to correctly identify all positive instances. Recall is the ratio of correctly predicted positive instances to the total actual positive instances and is given by
   $$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{8}$$
   Recall is useful when missing positive cases have a high cost.

4. **F1-Score**:
   The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that balances both. This accuracy metric is especially useful for imbalanced classes.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

The importance of each metric is problem specific. In the context of driver drowsiness detection the accuracy and precision are the most important metrics. Classification of an individual as awake when the driver is, in fact, drowsy is costly compared to the inverse case. This situation is likely if the system has low precision.

2.5 Face Detection

Face detection is integral to many image processing applications. Face detection serves as a foundation for numerous applications in security and surveillance systems. The first face detection models were simple statistical models that relied heavily on orientation and facial symmetry. The earliest algorithms for face detection date back to the 1960s. These methods relied on clever linear algebra techniques like the principal component analysis (PCA) method developed at MIT by Turk and Pentland [16] in 1991 known as Eigenface. In 2001 the revolutionary Viola-Jones object detection framework [18] was first introduced. Up to this point real time face detection was not possible due to the computationally expensive algebraic operations required to perform PCA.

The Viola-Jones algorithm was state-of-the-art until the deep learning boom in the 2010s. The You-Only-Look-Once (YOLO) object detection algorithm developed by Redmon *et al.* [12] in 2016 gained widespread popularity due to the speed and scalability of the model. The most recent development in face detection techniques is the YuNet model. This model was introduced in 2023 by Yu *et al.* [19] and was specifically designed for embedded applications. The resulting model is small in size, robust and extremely fast. YuNet is designed as a single-stage, anchor-free face detector optimised for real-time applications. The model architecture includes a lightweight backbone and a tiny feature pyramid network (TFPN) neck. A discussion of the TFPN architecture falls outside the scope of this report but more information may be found in [19].

3 Methodology

This section discusses the specific system implementation details. First the overall pipeline of the project is outlined. Then, feature extraction, training data and finally the model training procedure are discussed.

3.1 Overall Project Pipeline

The driver drowsiness detection system should take in video input and return whether the individual is drowsy or not. The assumption that drowsiness can be inferred from an

individual's eye state is used. That is, given that both eyes remain closed for a period longer that one second[2] the drowsy label is assigned to the individual. A threshold of one second is chosen, however in theory any threshold may be used. Video input is simply many sequentially shown images. The problem reduces to extraction of the eye regions given image input. In Figure 3 a flow diagram of the overall pipeline is provided. Here

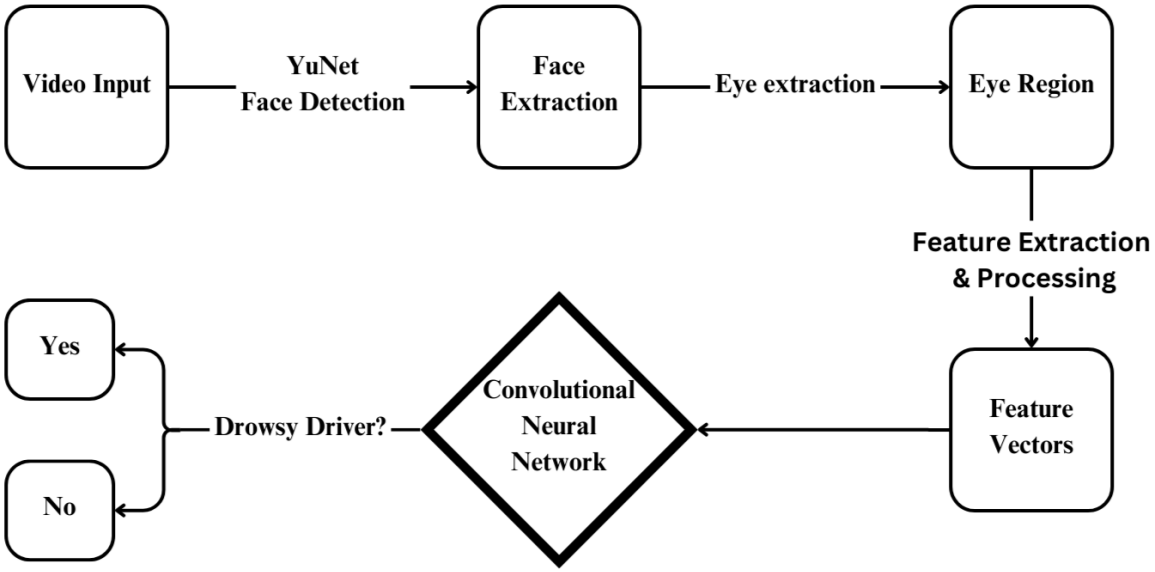


Figure 3: Model pipeline.

the YuNet face detection model is used to find the coordinates of the driver's eyes. These coordinates are used to determine how far the eyes are apart. The appropriate eye region is extracted based on this measurement.

The eye region is used to create feature vectors for the CNN with image processing techniques. The feature vectors are passed into a self-trained CNN that solves the binary classification problem. A timer variable is created that restarts if both eyes are open. If the timer exceeds one second the individual is labeled drowsy. In the event that no face is detected the most recent eye state is stored and is used as the measure of drowsiness until a new eye state is provided. This is done to limit false drowsiness classifications while turning or reversing. However, if the eyes are closed and then no face is detected, the timer continues and alerts the driver. This ensures that if a person's head drops as a result of fatigue the system labels them drowsy.

3.2 Feature Extraction

The colour image of the eye can be represented as a three-dimensional matrix $\mathbf{I}_{RGB}$ of size $h \times w \times 3$, where h is the height, w is the width, and the third dimension corresponds to the color channels: Red, Green, or Blue. To convert this RGB image to a greyscale image, the standard weighting

$$\mathbf{I}_{\text{grey}}(i,j) = 0.299 \cdot \mathbf{I}_{\text{R}}(i,j) + 0.587 \cdot \mathbf{I}_{\text{G}}(i,j) + 0.114 \cdot \mathbf{I}_{\text{B}}(i,j) \tag{10}$$

[2]Blinking is not considered drowsy behavior.

is used. Here $\mathbf{I}_{\mathrm{R}}(i,j)$, $\mathbf{I}_{\mathrm{G}}(i,j)$, and $\mathbf{I}_{\mathrm{B}}(i,j)$ are the pixel values at position (i,j) in the Red, Green, and Blue channels, respectively. The result is a one-dimensional matrix of pixel values[3] between 0 and 255. The greyscale image is resized to a 64×64 square image with the area-based resizing method. This resizing method averages pixel values within a specified window and is optimal for the downsampling of an image. Finally, the pixel values are normalised to be in the interval [0, 1].

$$\mathbf{I}_{\mathrm{normalized}}(i,j) = \frac{\mathbf{I}_{\mathrm{grey}}(i,j)}{255} \tag{11}$$

The input layer of the NN performs global z-score normalisation shown in Eq. 2. Here the mean is subtracted from the pixel intensity and the result is scaled by the standard deviation at pixel position (i,j). The mean and standard deviation values are only calculated once during the training procedure and act as the non-trainable parameters of the model. When images have high variance in brightness, z-score normalisation adjusts these differences and allows the model to learn features that are invariant to lighting conditions.

3.3 Training Data

The MRL eye dataset is used as training data [17]. This dataset has a total of 84,898 greyscale images of eyes. The eyes are labeled with six different conditions:

- gender [0 - man, 1 - woman]
- glasses [0 - no, 1 - yes]
- eye state [0 - closed, 1 - open]
- reflections [0 - none, 1 - small, 2 - big]
- lighting conditions [0 - bad, 1 - good]
- sensor ID [01 - RealSense, 02 - IDS, 03 - Aptina][4]

In Figure 4 two training examples from the MRL eye dataset are shown. The dataset is split with a ratio of 70:15:15 between training, validation and test data. Since the data is already monochromatic, the images are only resized to 64×64 and the pixel values are normalised to a range of [0,1]. The image is not unraveled into a one dimensional feature vector, since the implementation uses two dimensional convolutional layers. Fortunately the MRL eye dataset is sufficiently large and there are no data quality issues that need addressing. Notably, the dataset accounts for poor lighting conditions and individuals with glasses, which eases data collection.

[3]By convention, pixel intensities are stored as unsigned eight bit integers, so the interval [0,255] is used.

[4]This label refers to the sensor type used to capture the image and relates to the resolution of the image.

(a) Open eye example.

(b) Closed eye example.

Figure 4

3.4 Training

The proposed model is trained with NVIDIA's state-of-the-art CUDA platform and the TensorFlow ML library. NN training lends itself well to parallelisation[5] and the CUDA platform enables the use of GPUs instead of central processing units (CPU) for simple computations. NN training may be vectorised and the backpropagation algorithm reduces to many simple matrix multiplications. These are exactly the types of computations that GPUs excel at. For illustration the same model was trained twice, once with the CUDA framework and once without. The CUDA implementation finished 50 epochs in just under four minutes, while the default implementation finished in 44 minutes. An 11-fold increase.

The mini-batch training scheme is used. The entire training set is split into smaller mini-batches. The training set comprises 59,428 samples and a mini-batch size of 128 is used; this equates to 465 batches. Per iteration, each batch is fed through the model, gradients are calculated and the model weights are updated. An epoch is defined as one complete pass through the entire dataset. The model is trained for a maximum of 50 epochs. Here each epoch comprises 465 iterations, with one iteration per mini-batch.

In practice the maximum number of epochs is rarely reached. An early stopping condition is used to save time and prevent overfitting. Every 100 iterations the accuracy of the model is tested on the validation set. If a new optimal model is not found within 500 iterations, training is stopped and the model that resulted in the best validation accuracy is chosen.

The BCE objective function shown in Eq. 3 is used during training. The ADAM optimiser is used as training algorithm. A learning rate $\alpha = 0.001$ and exponential decay rates for moment estimates $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are used. All hidden layers use the ReLU activation function in Eq. 4, while the final output layer uses the sigmoid activation in Eq. 5.

[5]A computation framework that allows tasks to be executed simultaneously across multiple processors.

The self-trained CNN is made up of nine hidden layers of varying complexity and is a mixed-dimensionality model. The aim is to create a model that is as small as possible, while maintaining sufficient accuracy. To this end model architectures of different size are investigated. A grid search is performed to find the optimal number of kernels, kernel size, stride and the number of neurons in the first dense layer. A summary of this grid search is given in Section 4, while the full grid search results are provided in Appendix II.

4 Results

In this section, the accuracy of the CNN and the overall performance of the driver drowsiness detection system are discussed. The search for an optimal model architecture, that is both small in size and sufficiently accurate, is also described.

4.1 Model Architecture

In Figure 5 a visual overview of the model architecture is shown and Table 1 provides a numeric summary of the model. A two stage CNN with ReLU activation was developed. The normalisation layer contains 8,192 non-trainable parameters[6]. The first hidden layer is a convolutional layer with eight learnable kernels of size 5×5. This layer will create eight

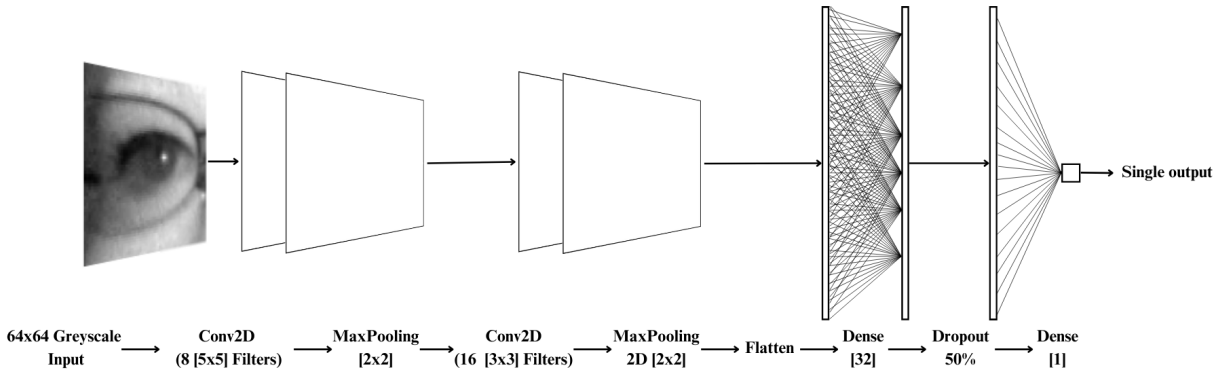


Figure 5: Model architecture.

feature maps and is responsible for the detection of basic features like edges and textures. Each convolutional layer is followed by a ReLU activation layer, not shown in Fig. 5. The activation layer feeds into a two dimensional max pooling layer with a 2×2 pool size and a stride of one. The pooling layer halves the spatial dimensionality. The convolutional stage is repeated but now the kernel is shrunk to 3×3 and the convolutional layer is allowed to learn 16 kernels instead of 8. After pooling, the output now has dimension $13 \times 13 \times 16 = 2,704$, which is flattened into a one dimensional vector. A dense layer with 32 neurons that connects to each of the 2,704 neurons of the flattened vector is used. Next a 50% dropout layer is introduced. This does not affect the size of the final model but it does improve generalisation ability at the cost of an increased training time. The final

[6]For each pixel index in the 64×64 image, one mean and one standard deviation must be stored for use during standardisation.

Table 1: CNN Model Architecture.

Layer (type)	Output Shape	# of Parameters
Normalisation	(64, 64, 1)	8,192 (non-trainable)
Conv2D	(30, 30, 8)	208
MaxPooling2D	(29, 29, 8)	0
Conv2D	(14, 14, 16)	1,168
MaxPooling2D	(13, 13, 16)	0
Flatten	(2,704,1)	0
Dense	(32,1)	86,560
Dropout	(32,1)	0
Dense	(1)	33
Total Parameters		**96,162**

output layer only produces a single value[7] between zero and one. An instance is labeled as drowsy if the output is less than 0.5 and non-drowsy otherwise. The size of the model may be estimated by counting the number of learnable parameters. The first convolutional layer uses a $5 \times 5 = 25$ kernel. With the bias term, each kernel has $25 + 1 = 26$ learnable parameters. Since 8 kernels are used, the first layer has $8 \times 26 = 208$ learnable parameters. The total size of the model is less than 500 KB.

4.2 Model Accuracy

The training and validation accuracy is provided in Figure 6, while the training and validation loss is shown in Figure 7. The final model achieved an accuracy 98.43% on the

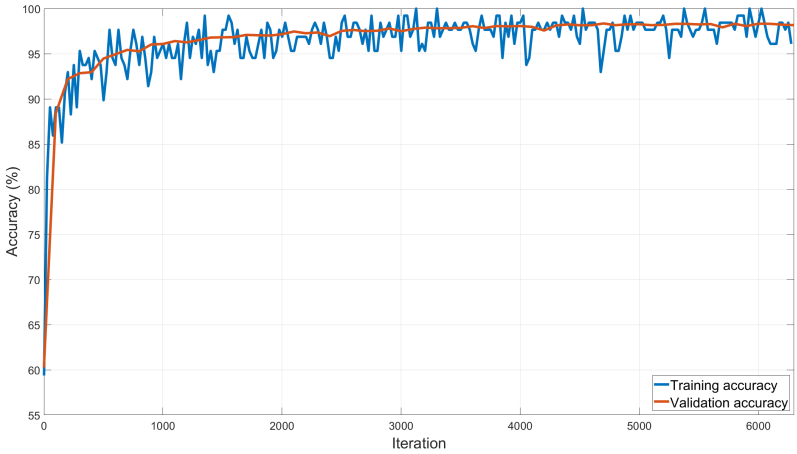


Figure 6: Training and validation accuracy.

training set. The early stopping condition was triggered at iteration 6,300 during epoch 15.

[7]Here an open interval is used, since an input of ∞ would be required for an output of one.

The model that achieved the best validation accuracy was selected. This network achieved a validation accuracy of 98.53%. The model achieved a test accuracy of 98.39% on 12,251 test images.

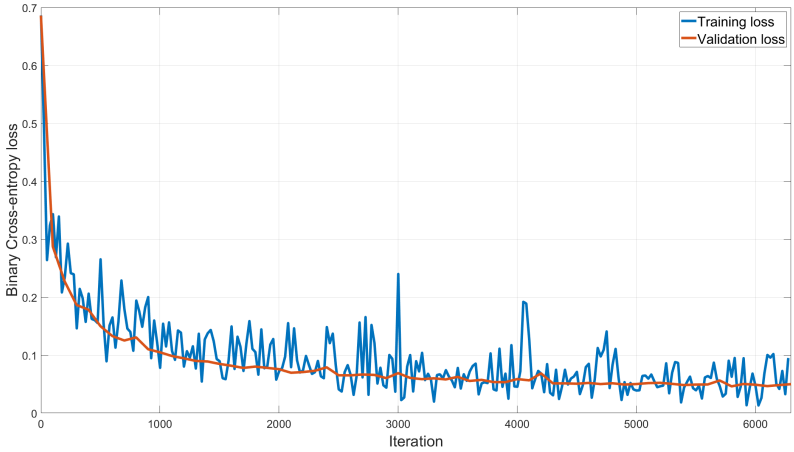


Figure 7: Training and validation loss.

Seen from the confusion matrix in Figure 8, 91 eyes were misclassified as open when the true label was closed. Similarly, 106 eyes were misclassified as closed when the true label was open. Two misclassifications of each type are provided in Figure 9. The model achieved a 0.99 and 0.98 precision and recall score respectively. Here the precision metric in Eq. 7 is important. The misclassification of an individual as not drowsy, with ground truth drowsy is costly and makes the model untrustworthy. The model achieved a F1-score of 0.98.

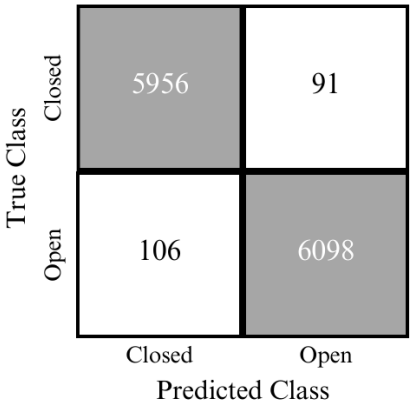
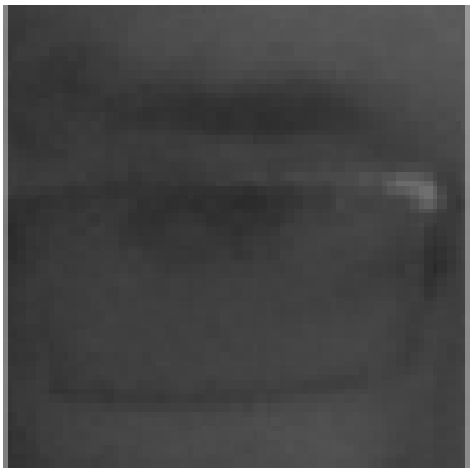
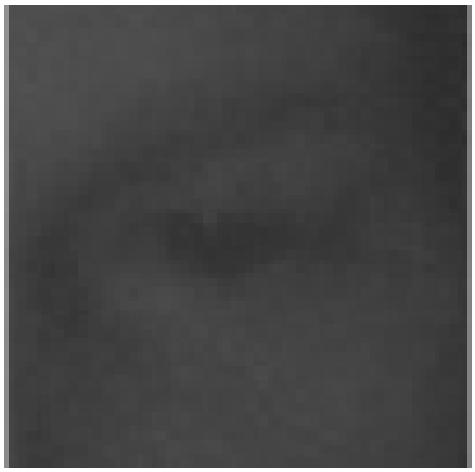


Figure 8: Confusion matrix.

Some of the images in the dataset, seen in Fig. 9, have debatable labels. The image in Figure 9d shows one of these cases. One might notice that the original images greatly differ in brightness. This fact is accounted for by the input layer, where the input image is normalised with z-score normalisation. This layer ensures that the system is robust to variations in pixel intensity at an added cost of 8,192 parameters. The addition allows the



(a) True = Open, Predicted = Closed



(b) True = Open, Predicted = Closed

(c) True = Closed, Predicted = Open

(d) True = Closed, Predicted = Open

Figure 9: Misclassifications.

model to generalise better to extreme lighting conditions. In the context of the drowsy driver problem this is clearly worthwhile, since the probability that a driver is fatigued at night is much higher than at daytime.

4.3 Grid Search

The final model architecture was determined by performing a grid search. The search space comprises the optimal number and size of filters in the convolutional stages, stride size and the number of fully connected units in the first dense layer. Table 2 shows a condensed version of the grid search where the stride is one and the kernel sizes for stage one and two are 5×5 and 3×3 respectively. The full grid search was completed in 3 hours with an average training time of just over two minutes. Table 2 is sorted from small to large with respect to model size. The final implementation is indicated in bold font.

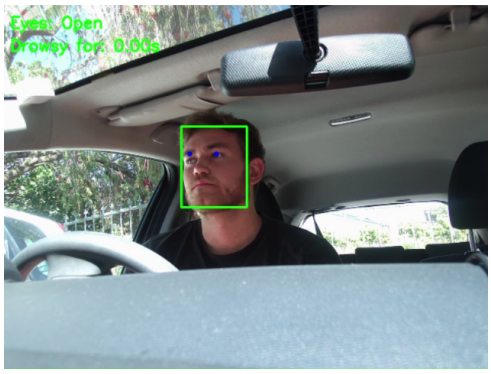
Table 2: Model architecture grid search.

# Stage 1 Filters	# Stage 2 Filters	# Fully Connected Units	Validation Accuracy
2	4	8	94.1393
2	8	8	95.9514
2	16	8	96.0167
4	4	8	95.3636
4	8	8	96.2370
4	16	8	96.5544
8	4	8	95.4779
8	8	8	96.4493
8	16	8	96.4085
2	4	16	95.8697
2	8	16	96.2126
2	16	16	97.6573
4	4	16	96.6533
4	8	16	97.0370
4	16	16	97.5022
8	4	16	96.9062
8	8	16	96.6044
8	16	16	97.3880
2	4	32	96.5064
2	8	32	97.5757
2	16	32	98.0002
4	4	32	97.0533
4	8	32	97.5349
4	16	32	98.1094
8	4	32	97.4451
8	8	32	97.9838
8	**16**	**32**	**98.5328**

Even the smallest model, shown in Table 2, performed exceptionally well on the validation set with an accuracy of 94.14%. The final implementation used the largest model. This was decided upon informal testing of the smaller models. The conclusion was that the system did not run considerably faster when the smaller models are used. For this reason the more accurate model is used. The search space of the grid search was capped at 8 and 16 filters for stage one and two respectively, since the resultant model with 32 fully connected units was sufficiently accurate with a validation accuracy of 98.53%. An extended grid search, not presented in Table 3, revealed that an increase in scale only marginally increased validation accuracy. An increase in scale would undoubtedly increase the accuracy of the model, but the added scale does not justify an accuracy increase of two percent at best. This fact coupled with ambiguities in the dataset labels concluded the grid search.

4.4 Final Implementation

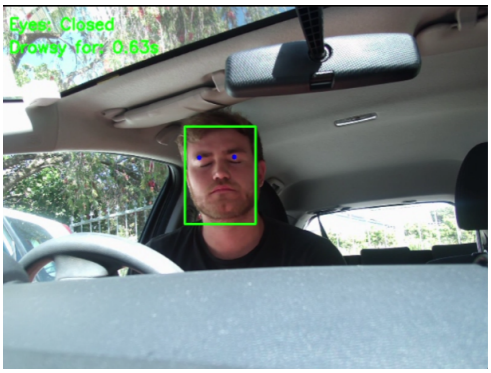
It is important that the size and speed of the model is also considered as a performance metric. The total size of the system is 718 KB[8]. In Figure 10 snapshots of the model in action are provided[9]. The base implementation runs at 6 frames per second. Per second 12



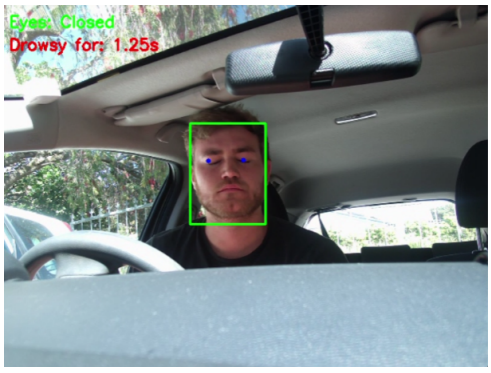
(a) Eyes: Open, Drowsy: 0.00s

(b) Eyes: Not Detected, Drowsy: 0.00s

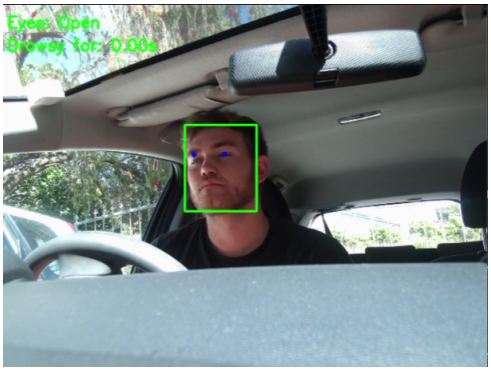
(c) Eyes: Not Detected, Drowsy: 0.00s



(d) Eyes: Closed, Drowsy: 0.63s



(e) Eyes: Closed, Drowsy: 1.25s



(f) Eyes: Open, Drowsy: 0.00s

Figure 10: Final model implementation

eyes are processed. The speed of the system may be easily doubled. Instead of processing

[8]Provided that the necessary python packages are already installed.

[9]A video example of the system may be found at https://github.com/Andreas-Wild/DriverDrowsiness.git.

both eyes each frame, only a single eye is used to determine drowsiness. This change was made under the assumption that one does not sleep with one eye open. The choice of eye alternates between left and right. This simple change doubled the frame rate and produced the same accuracy.

The snapshots in Fig. 10 were captured approximately half a second apart. In Fig. 10b and 10c no eyes were detected, because the eye state in the previous snapshot Fig. 10a was open, the drowsiness timer does not start. For illustration when the threshold is reached the drowsiness timer is included as red text, seen in Figure 10e. Upon close inspection of Fig. 10f one can see that the eyes of the driver are barely open and the system still correctly identified the eyes as open.

5 Conclusion

In this paper a driver drowsiness detection system is proposed that provides real-time alerts to the driver. The proposed system makes use of two connected deep learning models that determine whether an individual is drowsy or alert. A custom convolutional neural network (CNN) was trained with NVIDIA's state-of-the-art CUDA platform in Python. The CNN correctly predicted 98.34% of 12,251 test images. The final system allows for embedded applications due to the lightweight yet robust implementation.

The system is modular and models of varying size my be used dependent on the hardware available. In future this model may be adapted to run on mobile devices due to their accessibility. Additionally a high speed embedded C++ implementation on an Arduino or Raspberry Pi chip may be considered. The CNN could also be improved by considering a single colour image that contains both eyes. This would lower the image processing demands of the system and allow information from both eyes to enter the model with a single query. Currently the system has no way of detecting drowsiness if the eyes are not visible, like in the event that sunglasses are worn. Future work would include the addition of new drowsiness metrics, such as head bobbing or yawning.

The proposed system successfully detects driver drowsiness and provides timely alerts to the driver. The model is lightweight, modular and most importantly accurate in predictions. The full project may be readily installed with the procedure outlined in Appendix I. The software may be used to create an embedded system that directly solves the real-world problem of driver fatigue.

Acronyms

ADAM Adaptive Moment Estimation

AI Artificial Intelligence

BCE Binary Cross-Entropy

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DNN Deep Neural Network

ECG Electrocardiography

EEG Electroencephalography

EOG Electrooculography

GPU Graphics Processing Unit

ML Machine Learning

NHTSA National Highway Traffic Safety Administration

NN Neural Network

PCA Principal Component Analysis

ReLU Rectified Linear Unit

RMSPROP Root Mean Square Propagation

SGDM Stochastic Gradient Decent with Momentum

SWM Steering Wheel Movement

TFPN Tiny Feature Pyramid Network

YOLO You-Only-Look-Once

References

[1] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4): 322–333, 1969. doi: 10.1109/TSSC.1969.300225.

[2] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[3] Geoffrey Hinton. Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2012. URL `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Coursera Lecture.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[5] Rateb Jabbar, Mohammed Shinoy, Mohamed Kharbeche, Khalifa Al-Khalifa, Moez Krichen, and Kamel Barkaoui. Driver drowsiness detection model using convolutional neural networks techniques for android application. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 237–242. IEEE, 2020.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Claudia Maldonado, Duncan Mitchell, Taylor SR, and Helen Driver. Sleep, work schedules and accident risk in south african long-haul truck drivers. *South African Journal of Science*, 98:319–324, 07 2002.

[9] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[10] Seyed Mohammad Reza Noori and Mohammad Mikaeili. Driving drowsiness detection using fusion of electroencephalography, electrooculography, and driving quality signals. *Journal of Medical Signals & Sensors*, 6(1):39–46, 2016.

[11] Ajinkya Rajkar, Nilima Kulkarni, and Aniket Raut. Driver drowsiness detection using deep learning. In *Applied Information Processing Systems: Proceedings of ICCET 2021*, pages 73–82. Springer, 2022.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[13] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[15] Vandna Saini and Rekha Saini. Driver drowsiness detection system and techniques: a review. *International Journal of Computer Science and Information Technologies*, 5 (3):4245–4249, 2014.

[16] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[17] MRL Laboratory University of Ljubljana. Mrl eye dataset. `https://mrl.cs.vsb.cz/eyedataset`, 2016. Accessed: 2024-03-01.

[18] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–511. IEEE, 2001.

[19] Wei Wu, Hanyang Peng, and Shiqi Yu. Yunet: A tiny millisecond-level face detector. *Machine Intelligence Research*, 20(5):656–665, 2023. doi: 10.1007/s11633-023-1423-y.

Appendix I

An installation guide is included in Appendix I. The project may be downloaded from the GitHub repository https://github.com/Andreas-Wild/DriverDrowsiness.git. Prerequisites for installation include:

- A Python version 3.10-3.12.
- Git is required to clone the repository.
- Optionally Pip may be used as a Python package manager.

Installation Steps
In the command window:

1. Clone the repository with:
   `git clone https://github.com/Andreas-Wild/DriverDrowsiness.git`
2. Set the working directory with:
   `cd DriverDrowsiness`
3. Install the necessary dependencies (a virtual environment is recommended but not necessary):
   `pip install -r requirements.txt`
4. Ensure that a webcam is available and working:
   `python test_webcam.py`
5. Run the main python script:
   `python main.py`

The project is distributed under the MIT License. This project is fully documented, with all functions and methods containing docstrings following the Google convention. Further details of the project and example usage may be found on GitHub. A simple troubleshooting guide and `README.md` is also included in the the documentation.

Appendix II

Appendix II presents the full results of the grid search conducted to optimize the architecture of the convolutional neural network (CNN) model. The grid search explored different configurations of filter sizes, kernel sizes, strides, and fully connected units to identify the combinations that maximize validation accuracy, while minimising model size. Table 3 below shows the performance of each configuration, with percentage accuracy on the validation set reported for each set of parameters.

Table 3: Full grid seacrh results.

# Filters1	# Filters2	KernelSize1	KernelSize2	Stride	# FC Units	ValAccuracy (%)
2	4	3	3	1	8	92.9
2	4	3	3	3	8	79.5
2	4	3	5	1	8	95.2
2	4	3	5	3	8	88.8
2	4	5	3	1	8	93.4
2	4	5	3	3	8	86.2
2	4	5	5	1	8	95.0
2	4	5	5	3	8	89.2
2	4	3	3	1	16	94.5
2	4	3	3	3	16	78.2
2	4	3	5	1	16	96.2
2	4	3	5	3	16	89.4
2	4	5	3	1	16	95.5
2	4	5	3	3	16	86.2
2	4	5	5	1	16	96.5
2	4	5	5	3	16	86.4
2	4	3	3	1	32	95.1
2	4	3	3	3	32	82.6
2	4	3	5	1	32	96.6
2	4	3	5	3	32	88.2
2	4	5	3	1	32	96.6
2	4	5	3	3	32	82.7
2	4	5	5	1	32	95.9
2	4	5	5	3	32	88.7
2	8	3	3	1	8	94.0
2	8	3	3	3	8	84.5
2	8	3	5	1	8	95.7
2	8	3	5	3	8	91.1
2	8	5	3	1	8	95.3
2	8	5	3	3	8	79.2
2	8	5	5	1	8	95.3
2	8	5	5	3	8	91.5
2	8	3	3	1	16	95.2
2	8	3	3	3	16	82.8
2	8	3	5	1	16	94.9
2	8	3	5	3	16	86.9
2	8	5	3	1	16	95.7
2	8	5	3	3	16	82.6
2	8	5	5	1	16	96.8
2	8	5	5	3	16	91.6
2	8	3	3	1	32	95.9
2	8	3	3	3	32	81.9
2	8	3	5	1	32	97.2
2	8	3	5	3	32	86.9
2	8	5	3	1	32	96.3
2	8	5	3	3	32	85.8
2	8	5	5	1	32	97.3
2	8	5	5	3	32	92.8
2	16	3	3	1	8	94.6
2	16	3	3	3	8	82.4

# Filters1	# Filters2	KernelSize1	KernelSize2	Stride	# FC Units	ValAccuracy (%)
2	16	3	5	1	8	95.4
2	16	3	5	3	8	91.2
2	16	5	3	1	8	96.0
2	16	5	3	3	8	86.4
2	16	5	5	1	8	95.7
2	16	5	5	3	8	94.1
2	16	3	3	1	16	96.4
2	16	3	3	3	16	83.8
2	16	3	5	1	16	96.0
2	16	3	5	3	16	92.4
2	16	5	3	1	16	97.0
2	16	5	3	3	16	87.2
2	16	5	5	1	16	96.7
2	16	5	5	3	16	91.2
2	16	3	3	1	32	97.2
2	16	3	3	3	32	83.9
2	16	3	5	1	32	97.4
2	16	3	5	3	32	90.7
2	16	5	3	1	32	96.6
2	16	5	3	3	32	85.7
2	16	5	5	1	32	97.8
2	16	5	5	3	32	93.7
4	4	3	3	1	8	93.9
4	4	3	3	3	8	82.9
4	4	3	5	1	8	95.0
4	4	3	5	3	8	90.0
4	4	5	3	1	8	94.6
4	4	5	3	3	8	84.0
4	4	5	5	1	8	95.6
4	4	5	5	3	8	89.7
4	4	3	3	1	16	94.9
4	4	3	3	3	16	83.7
4	4	3	5	1	16	96.1
4	4	3	5	3	16	90.0
4	4	5	3	1	16	95.9
4	4	5	3	3	16	86.4
4	4	5	5	1	16	95.4
4	4	5	5	3	16	90.0
4	4	3	3	1	32	95.5
4	4	3	3	3	32	82.8
4	4	3	5	1	32	96.3
4	4	3	5	3	32	90.1
4	4	5	3	1	32	96.3
4	4	5	3	3	32	86.0
4	4	5	5	1	32	97.1
4	4	5	5	3	32	90.8
4	8	3	3	1	8	94.9
4	8	3	3	3	8	85.0
4	8	3	5	1	8	96.4
4	8	3	5	3	8	90.1
4	8	5	3	1	8	95.4
4	8	5	3	3	8	89.1
4	8	5	5	1	8	95.7
4	8	5	5	3	8	91.9
4	8	3	3	1	16	95.5
4	8	3	3	3	16	84.7
4	8	3	5	1	16	96.2
4	8	3	5	3	16	91.1
4	8	5	3	1	16	96.5
4	8	5	3	3	16	87.2
4	8	5	5	1	16	95.9
4	8	5	5	3	16	92.4
4	8	3	3	1	32	96.2

# Filters1	# Filters2	KernelSize1	KernelSize2	Stride	# FC Units	ValAccuracy (%)
4	8	3	3	3	32	85.0
4	8	3	5	1	32	97.2
4	8	3	5	3	32	91.5
4	8	5	3	1	32	96.5
4	8	5	3	3	32	87.0
4	8	5	5	1	32	97.4
4	8	5	5	3	32	93.2
4	16	3	3	1	8	96.3
4	16	3	3	3	8	88.5
4	16	3	5	1	8	96.2
4	16	3	5	3	8	92.8
4	16	5	3	1	8	95.1
4	16	5	3	3	8	88.2
4	16	5	5	1	8	96.2
4	16	5	5	3	8	94.4
4	16	3	3	1	16	96.4
4	16	3	3	3	16	86.4
4	16	3	5	1	16	97.1
4	16	3	5	3	16	92.5
4	16	5	3	1	16	96.7
4	16	5	3	3	16	88.6
4	16	5	5	1	16	97.2
4	16	5	5	3	16	93.2
4	16	3	3	1	32	97.1
4	16	3	3	3	32	86.9
4	16	3	5	1	32	97.3
4	16	3	5	3	32	94.1
4	16	5	3	1	32	96.6
4	16	5	3	3	32	90.3
4	16	5	5	1	32	97.6
4	16	5	5	3	32	94.2
8	4	3	3	1	8	94.7
8	4	3	3	3	8	81.5
8	4	3	5	1	8	94.9
8	4	3	5	3	8	90.7
8	4	5	3	1	8	95.2
8	4	5	3	3	8	86.7
8	4	5	5	1	8	95.5
8	4	5	5	3	8	87.1
8	4	3	3	1	16	95.8
8	4	3	3	3	16	83.9
8	4	3	5	1	16	95.8
8	4	3	5	3	16	90.3
8	4	5	3	1	16	96.3
8	4	5	3	3	16	86.0
8	4	5	5	1	16	96.4
8	4	5	5	3	16	89.0
8	4	3	3	1	32	96.1
8	4	3	3	3	32	86.2
8	4	3	5	1	32	96.2
8	4	3	5	3	32	90.4
8	4	5	3	1	32	96.5
8	4	5	3	3	32	86.4
8	4	5	5	1	32	97.6
8	4	5	5	3	32	92.6
8	8	3	3	1	8	95.8
8	8	3	3	3	8	86.8
8	8	3	5	1	8	96.0
8	8	3	5	3	8	92.6
8	8	5	3	1	8	96.0
8	8	5	3	3	8	88.1
8	8	5	5	1	8	97.0
8	8	5	5	3	8	92.5

# Filters1	# Filters2	KernelSize1	KernelSize2	Stride	# FC Units	ValAccuracy (%)
8	8	3	3	1	16	96.9
8	8	3	3	3	16	87.4
8	8	3	5	1	16	96.6
8	8	3	5	3	16	92.1
8	8	5	3	1	16	96.0
8	8	5	3	3	16	89.3
8	8	5	5	1	16	97.2
8	8	5	5	3	16	92.9
8	8	3	3	1	32	96.1
8	8	3	3	3	32	85.9
8	8	3	5	1	32	96.9
8	8	3	5	3	32	92.3
8	8	5	3	1	32	97.3
8	8	5	3	3	32	91.0
8	8	5	5	1	32	97.5
8	8	5	5	3	32	94.5
8	16	3	3	1	8	95.4
8	16	3	3	3	8	87.9
8	16	3	5	1	8	96.2
8	16	3	5	3	8	93.8
8	16	5	3	1	8	96.0
8	16	5	3	3	8	90.4
8	16	5	5	1	8	97.1
8	16	5	5	3	8	93.4
8	16	3	3	1	16	96.6
8	16	3	3	3	16	86.2
8	16	3	5	1	16	97.4
8	16	3	5	3	16	93.1
8	16	5	3	1	16	97.1
8	16	5	3	3	16	90.7
8	16	5	5	1	16	97.4
8	16	5	5	3	16	94.4
8	16	3	3	1	32	96.6
8	16	3	3	3	32	89.8
8	16	3	5	1	32	96.1
8	16	3	5	3	32	93.8
8	16	5	3	1	32	98.5
8	16	5	3	3	32	91.5
8	16	5	5	1	32	97.6
8	16	5	5	3	32	93.4