GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

WEBCAM-BASED EYETRACKING USING DEEP LEARNING APPROACHES

Henning Siept Dinkela
11337652

GEORG AUGUST UNIVERSITÄT GÖTTINGEN

Programme: M.Sc. Angewandte Statistik
Module: M.WIWI-QMW.0020
Course: Practical Statistical Training

Assessor: Prof. Dr. Thomas Kneib
Supervisors: Dr. Alexander Silbersdorf & Ruven Zapf

Working Period: October 2024 to March 2025
Date of Submission: 15.03.2025

Contents

Abstract

Reliable detection of eye blinks is crucial for various applications, including attention monitoring, human-computer interaction, and fatigue assessment. Traditional eye-tracking technologies, although accurate, generally require specialized hardware, resulting in high costs and limited accessibility. This study proposes a cost-effective, real-time blink detection approach utilizing a ResNet50-based Convolutional Neural Network (CNN) trained on eye images captured by a standard webcam. Given the inherent imbalance in blink datasets—closed-eye states occurring significantly less frequently than open-eye states—the Matthews Correlation Coefficient (MCC) was adopted as a balanced evaluation metric, complementing traditional accuracy measures. A straightforward, threshold-based definition of blinks was applied, achieving promising results. Nevertheless, future implementations could further benefit from incorporating temporal modeling techniques such as Long Short-Term Memory (LSTM) networks to better capture blink dynamics. Overall, the approach presented demonstrates the potential of deep learning methods combined with inexpensive webcam technology for robust blink detection, suggesting broad feasibility and accessibility for practical use.

Keywords: Blink Detection; Deep Learning; Convolutional Neural Network

1 Introduction

Blink detection plays a central role in various applications, such as user attention analysis, driving safety, and human-computer interaction (Fusek, 2018). It occupies a complementary position to research on gaze tracking and pupillometry, which must contend with eye blinks as artifacts that lead to distortions in gaze and pupil data (Culemann et al., 2023). We are specifically interested in the attention analysis regarding the learning behaviour of students consuming digital educational content, more on that in subsection 6.1. One approach might be to detect the students' blinking behaviour while watching lectures or tutorials in video form. The resulting blinking frequency may serve as a quantitative metric for identifying presentations that are challenging, unengaging, or insufficiently prepared from a didactic or pedagogical perspective (Kim et al., 2018). Furthermore, the research conducted by Frank et al. (2020) establishes a connection between blink rate and the performance of cognitive tasks. Additionally, Mazzone et al. (2010) demonstrate that blinks can be utilized as a potential tool for analyzing learning processes.

By developing robust and cost-effective systems that automatically detect blinks with high precision—without relying on costly, specialized sensors or proprietary, closed-source software—we can facilitate more accessible research tools. Therefore, the objective of this project is to develop and implement a Convolutional Neural Network (CNN) capable of classifying eye images into "open" and "closed" categories, thereby enabling the automatic detection of blinks. This leads to the following research question:

How to efficiently identify a blink using deep learning approaches?

2 Literature Review

The purpose of this literature review is to provide a comprehensive overview of the methodologies and techniques that have been developed for blink detection. This section systematically examines both traditional approaches—relying on handcrafted features and threshold-based classification—and modern methods rooted in deep learning, with a particular focus on convolutional neural networks (CNNs). The review highlights seminal contributions that have advanced the accuracy and robustness of blink detection systems across various application domains, such as driver safety, cognitive performance measurement, and human-computer interaction. By critically assessing the evolution of these techniques, the review aims to identify the strengths and limitations of current approaches, elucidate research gaps, and suggest directions for future investigations in the field.

As already mentioned in the section 1 eye blink detection is a longstanding problem in computer vision with applications ranging from human–computer interaction and driver drowsiness monitoring, cybersecurity to medical diagnostics (Xiong et al., 2025). Early blink detection methods relied on hand-crafted features (e.g., eye aspect ratios, optical flow or HOG descriptors) and classical classifiers or heuristic rules (Lalonde et al., 2007; Schillingmann and Nagai, 2015). The eye aspect ratio (EAR) for example is a quantitative measure used to evaluate the state of an eye—whether it is open or closed—by analyzing the distances between specific facial landmarks. This measure remains employed in contemporary re-

search, as evidenced by its use in Culemann et al. (2023). Typically, the EAR is computed by taking the sum of the distances between the vertical eye landmarks and dividing it by twice the distance between the horizontal landmarks. Formally, if p_1 through p_6 represent the eye landmarks, the EAR is defined as:

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

In Python this can be achieved with the dlib library. In this framework, face detection is initially performed using a histogram of oriented gradients (HOG) based detector combined with a linear classifier, which efficiently identifies frontal faces by leveraging hand-crafted features. Once a face is detected, a pre-trained shape predictor—typically based on an ensemble of regression trees—is applied to accurately localize 68 facial landmarks Kazemi and Sullivan (2014).
These traditional approaches often struggled under varied lighting, head poses, non-neutral facial expression or with different eye appearances—e.g. glasses, ethnic variability, etc (Cortacero et al., 2019).

Since the advent of classical CNN architectures (e.g., AlexNet, VGGNet, GoogLeNet, ResNet), these models have been effectively applied in the domain of gaze point estimation and, complementarily, blink detection. Anas et al. (2017) apply the LeNet model in two distinct experimental settings. Initially, the network is trained for a binary classification task to distinguish between "open" and "closed" eye states—which they state is the most common approach. Subsequently, the model is extended to a three-class problem, classifying eye states as "opened," "partially-opened," and "closed." According to them, there are several reasons for this extension. For instance, lighting conditions may force the eye to appear partially open, even when it should be classified as open. Additionally, in blink detection scenarios—particularly when using a low frame-rate camera—the rapid movement of the eyelids may result in the “closed” state not being captured reliably. Incorporating a “partially-opened” state helps to address these issues by providing a more nuanced representation of the eye’s state. Zhao et al. (2017) propose a framework—termed the deep integrated neural network (DINN)—consisting of two parallel branches: a deep neural network (DNN) and a deep convolutional neural network (DCNN). The DNN processes vectorized representations of the cropped eye patches, whereas the DCNN extracts appearance-based features directly from the eye images via convolutional and pooling layers. At the top of each branch, fully connected layers are employed to summarize the extracted features. These top layers are subsequently concatenated, and a further fully connected layer with a softmax activation is added to produce the final classification (i.e., open versus closed). Moreover, the framework utilizes a transfer learning strategy to pretrain both the DNN and DCNN components on larger datasets (FER2013 and CIFR-10), thereby facilitating robust feature extraction even on smaller, task-specific datasets. The combined loss function, which integrates the losses of the individual DNN, DCNN, and the fused model, is optimized jointly during fine-tuning, ensuring that the integrated network benefits from the complementary representations provided by both branches. Sanyal and Chakrabarty (2019) propose a two-stream convolutional network architecture in which one branch ("RGB-CNN") is trained on cropped eye RGB images, while the other branch ("Mask-CNN") is trained on corresponding binary masks. These masks are generated by computing a convex hull over six eye landmarks, effectively connecting them and assigning a pixel value of 1 (white) to the region inside the convex hull and 0 (black) to the region outside.[1] Both the

[1] For clarification, it may be important to note that "Mask-CNN" should not be confused with Mask R-CNN as introduced by He, Gkioxari, et al. (2017), which produces binary masks as output.

RGB-CNN and Mask-CNN employ a modified LeNet-5 architecture, each generating a two-class output via a softmax activation. Subsequently, the top fully connected layers from each network are concatenated, and an additional fully connected layer with a softmax function is appended for classifying the eye state as open or closed. During training the weights of the convolutional layers in both pre-trained RGB-CNN and Mask-CNN are kept fixed, while the top layers—including those in the individual networks and the joint concatenated layer—are fine-tuned to optimize performance. Pahariya et al. (2024) employ transfer learning by initializing a MobileNetV2 model with pretrained weights and fine-tuning it on the MRL Eye Dataset (see section 4.1) for eye state classification. This model is integrated into a real-time drowsiness detection system, where OpenCV and Haar Cascade Classifiers extract eye regions from webcam-captured facial images, and the modified MobileNetV2 classifies each eye as either "open" or "closed." If the system detects a sustained period of closed-eye frames, it promptly triggers an alert, indicating potential driver drowsiness. Ata et al. (2024) conduct a comprehensive evaluation of deep learning-based approaches for eye blink detection in video frames by comparing a standard CNN, VGG-19, and ResNet101V2. Their methodology involves extracting and preprocessing eye regions from video frames using traditional face and eye detection techniques, followed by training the models via transfer learning from ImageNet-pretrained weights. In the end, they find that the ResNet101V2 outperforms the other two models based on the achieved accuracy.

Another approach to blink detection using deep learning methods is the use of the recurrent neural network (RNN), which is particularly effective in processing sequential data. RNNs are designed with feedback connections that allow information to persist over time, making them well-suited for modeling temporal dependencies in video sequences. This characteristic enables RNNs, and their advanced variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), to capture the dynamic changes in eye states across consecutive frames. In the context of blink detection, a blink is not a single-frame event but a dynamic process that evolves over a sequence of frames—from the eyes being open, to closing, and then reopening. So, the basic idea here is to capture temporal dependencies that static models like CNNs inherently overlook. de la Cruz et al. (2024) introduces Eye-LRCN, a framework that integrates a Siamese CNN with a bidirectional LSTM network to effectively address blink and blink completeness detection. In this architecture, the Siamese CNN serves as a feature extractor that converts eye images into discriminative 256-dimensional vectors while mitigating class imbalance through pairwise training. These feature vectors are then fed into a bidirectional LSTM, which leverages temporal context over sequences of frames to capture the dynamic nature of blinking. Finally, a fully connected layer with softmax activation produces the classification output, distinguishing between different eye states. Transfer learning is also employed, as the ResNet18 backbone is initialized with ImageNet-pretrained weights and subsequently fine-tuned on domain-specific blink detection data, ensuring robust performance even with limited training samples.

3 Theory

This chapter provides a comprehensive theoretical overview of the methodologies that are subsequently implemented and evaluated in the following chapters of this paper.

3.1 Convolutional Neural Networks

CNNs are among the most influential architectures in modern artificial intelligence, particularly excelling in image recognition and processing tasks. Their origins, however, lie deep within the early development of neural networks. In 1943, Warren McCulloch and Walter Pitts introduced a simplified model of neural computation, establishing a theoretical framework that would inspire future research (McCulloch and Pitts, 1943). Building on this foundation, Frank Rosenblatt's invention of the perceptron in 1958 heralded an era of optimism in machine learning—often regarded as the early golden age of AI (Rosenblatt, 1958). Yet, as the limitations of these early models became apparent—most notably highlighted by Minsky and Papert (1969) in their work from 1969—the field experienced a period of diminished enthusiasm known as the AI winter.

The tide turned in the 1980s with the rediscovery and popularization of the backpropagation algorithm by researchers such as David Rumelhart, Geoffrey Hinton, and Ronald Williams in 1986, sparking renewed interest in neural networks and marking the onset of an AI spring. This resurgence paved the way for more specialized architectures. In the late 1980s and early 1990s, Yann LeCun and his colleagues adapted these ideas to develop convolutional neural networks, which were particularly adept at handling image data. With the advent of greater computational power, larger datasets, and further refinements in learning algorithms, CNNs eventually played a pivotal role in ushering in what many now celebrate as the AI Golden Age.

Convolution kernels, commonly referred to as filters, constitute the primary mechanism for extracting salient information from images by performing localized, parameter-shared operations that detect fundamental visual patterns. These filters, often small matrices of learnable weights, are systematically convolved with the input data by sliding across its spatial dimensions. At each position, the filter performs an element-wise multiplication with the corresponding segment of the input, and the resulting products are summed to generate a single output value in the feature map. This operation effectively detects specific local patterns—such as edges, textures, or simple shapes—by highlighting regions where the learned features are most prominent. The local connectivity and parameter sharing inherent in this process not only reduce the number of parameters (in contrast to fully connected layers, where each neuron is connected to every input unit and thus requires a larger number of learnable weights), but also ensure that the detected features are robust to variations in the input, thereby facilitating the subsequent hierarchical abstraction of more complex representations.

Figure 1 demonstrates how a single-channel 6×6 image is first padded with zeros to become 8×8, making it possible for a 3×3 filter to be applied at every valid position. As the filter slides across the padded image, each of its weights is multiplied by the corresponding entries in a local 3×3 region of the image, and the resulting products are summed to produce a single output value. Repeating this procedure across all valid positions generates a new 6×6 feature map. The bottom panels highlight one example of this convolution operation, showing how the multiplication of each filter element by the corresponding image pixel results in a sum of 7 for that specific position.[2] Mathematically, we can

[2]For clarity, a simplified representation of the dimensions is adopted here, whereas the original notation follows the form: *batch size* $\times$ *channel size* $\times$ *height* $\times$ *width*.

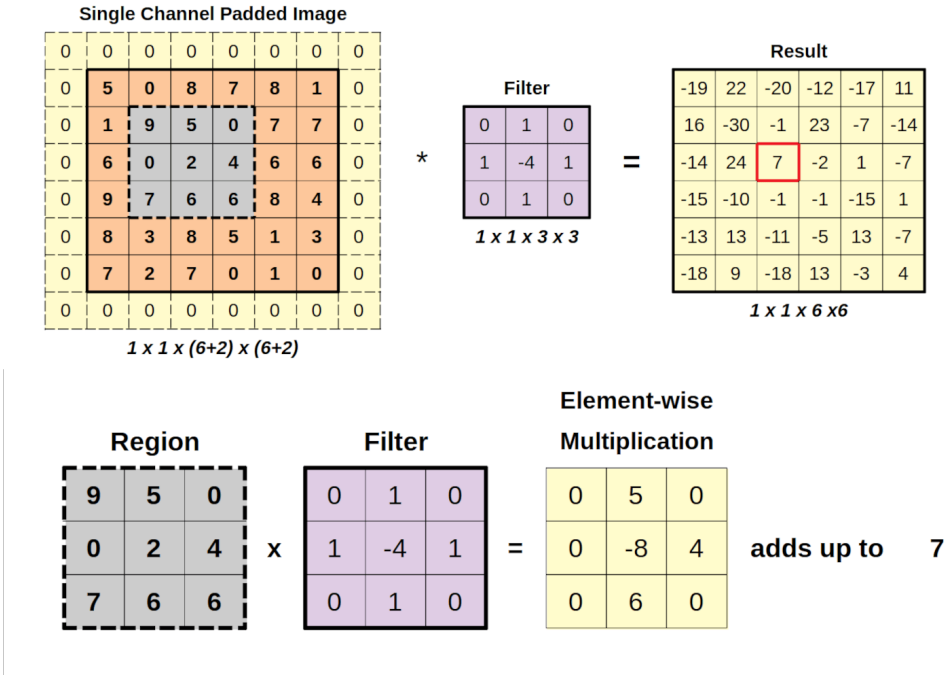


Figure 1: Convolution Kernel (Godoy, 2023a)

express the convolution operation as follows:

$$Y_{i,j,k} = \sum_{l=1}^{C} \sum_{m=1}^{H_k} \sum_{n=1}^{W_k} W_{m,n,l,k} \cdot X_{i+m-1,j+n-1,l} + b_k,$$

where:

- X is the input tensor with dimensions $(\text{height}, \text{width}, C)$,
- W denotes the convolutional kernel with spatial dimensions $H_k \times W_k$ and C input channels,
- b_k is the bias term for the k-th output channel,
- $Y_{i,j,k}$ is the output at position (i, j) for channel k.

Furthermore, the indices in the equation represent the following:

- k: the output channel index, corresponding to each distinct filter in the convolutional layer.
- l: the index over the input channels, ensuring that the convolution operation aggregates information from all channels.
- m and n: the spatial indices within the convolutional kernel, ranging over the kernel's height and width, respectively.

Given our image dimensions of 6×6 and a kernel size of 3×3, the output size can be calculated using the formula:

$$\text{output size} = \frac{width - kernel + 2 \cdot pooling}{stride} + 1 = \frac{6 - 3 + 2 \cdot 1}{1} + 1 = 6.$$

For example, the value at position row 3 and column 3 of the resulting feature map is computed as:

$$Y_{3,3,1} = \sum_{m=1}^{3}\sum_{n=1}^{3} W_{m,n,1,1} \cdot X_{3+m-1,3+n-1,1} = W_{1,1,1,1} \cdot X_{3,3,1} + W_{1,2,1,1} \cdot X_{3,4,1} + ... + W_{3,3,1,1} \cdot X_{5,5,1} = 7$$

Pooling layers are a fundamental component of convolutional neural network architectures, tasked with reducing the spatial dimensions of intermediate feature maps while preserving the most salient information. By applying localized aggregation operations—most notably max pooling, which selects the maximum value within a defined window, and average pooling, which computes the mean of the activations—these layers systematically down-sample the data. This reduction in spatial resolution not only decreases the computational load and memory requirements for subsequent layers but also introduces a level of invariance to minor translations and distortions in the input data. Such invariance is critical for enhancing the model's robustness and its ability to generalize to new, unseen examples. Moreover, pooling layers facilitate the hierarchical abstraction of features, effectively consolidating local details into more global representations as the network deepens.

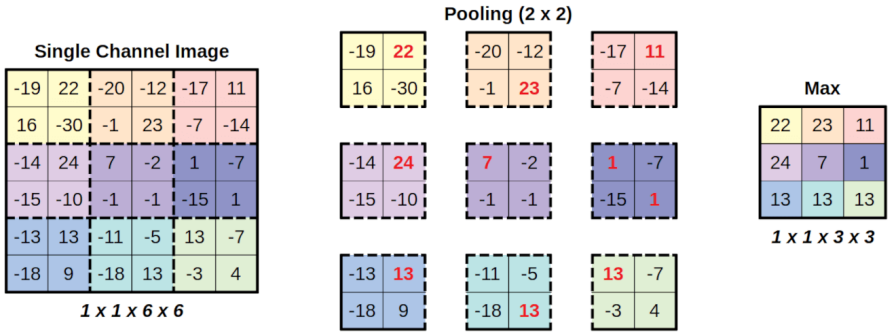


Figure 2: Max pooling (Godoy, 2023b)

Figure 2 illustrates the application of a 2×2 max pooling operation to a single-channel 6×6 image, reducing its spatial dimensions to 3×3. On the left, the original image is segmented into 2×2 regions, each of which is shown separately in the center. From each of these regions, the maximum value is selected—this process is highlighted by the red numbers in the figure—and subsequently placed into the corresponding position in the pooled output. Max pooling thus preserves only the most prominent feature within each local region. It is noteworthy that many newer architectures (like the ResNet architecture) have moved away from the traditional use of fixed pooling layers. Instead, they employ convolutional layers with a stride of 2—often using 3×3 filters—to perform downsampling. This approach allows the network to learn the optimal downsampling operation as part of the training process, potentially preserving more detailed feature information and improving overall performance Springenberg et al. (2014).

Fully connected layers, also referred to as dense layers, function as a classical feedforward neural network appended to the convolutional architecture (illustrated by Figure 3). After the final convolutional or pooling stages have extracted and refined spatial features, these features are flattened into a one-dimensional vector. This flattened vector—essentially a sequence of pixel or feature values—serves as the sole input to the fully connected layers, where each neuron is linked to every element in the input.

Consequently, the network is able to combine all the extracted features into a comprehensive representation. In most CNN architectures, these fully connected layers occupy the final stages, performing the classification or regression task by translating the learned feature maps into a single, definitive output.

Activation functions are used in between the feature extraction and classifier parts of a CNN. For instance, each layer computes a weighted sum of its inputs, followed by the application of an activation function. This step is crucial because, without non-linearity, deep neural networks would collapse into a single linear transformation, severely limiting their capacity to model complex, real-world data. In context of the linear layers, activation functions enable the approximation of intricate functions by allowing the network to learn non-linear mappings from input to output. Similarly, in convolutional layers, activation functions are normally applied after each convolutional and pooling operation. This ensures that the features extracted not only represent simple linear combinations but also capture higher-order interactions and subtle patterns such as edges, textures, and shapes. By interleaving linear operations with non-linear activation functions, the network develops a hierarchical representation of the input data, which is essential for tasks like image classification, object detection, and segmentation. Furthermore, the choice of activation function can influence the training dynamics by affecting gradient propagation, which in turn can lead to faster convergence and improved overall performance.

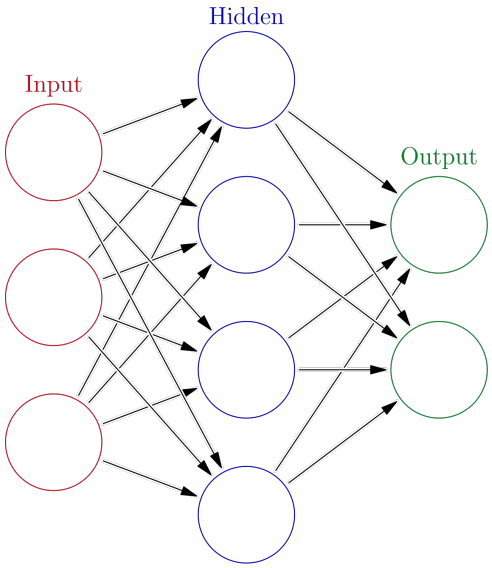


Figure 3: Classic Neural Network (Glosser.ca, 2013)

Batch Normalization is a widely adopted technique that addresses the problem of internal covariate shift by normalizing the inputs to each layer during training. In practice, batch normalization operates by computing the mean and variance of activations over a mini-batch, and then using these statistics to standardize the activations. This process is typically followed by a scaling and shifting step, which allows the network to recover any necessary representations. Batch normalization is typically applied after linear or convolutional operations and before the activation function. By ensuring that the inputs to each layer have a consistent distribution throughout training, batch normalization stabilizes and accelerates the learning process. It mitigates issues such as exploding or vanishing gradients, improves gradient flow, and reduces sensitivity to weight initialization. Additionally, it often provides a regularizing effect,

which can lead to improved generalization.

Residual neural networks, or short ResNet, is an innovative architecture designed to address the degradation problem encountered in very deep neural networks. Rather than attempting to learn an unreferenced mapping directly, ResNet layers are configured to learn residual functions relative to their inputs. This is achieved by incorporating shortcut connections—also known as skip connections—that bypass one or more layers and directly add the input to the output of the subsequent layer (see Figure 4). Mathematically, this concept is expressed as follows. In a typical residual block, the output y is given by:

$$y = F(x, \{W_i\}) + x,$$

where x represents the input to the block, $F(x, \{W_i\})$ denotes the residual function (typically implemented as a series of convolutional layers, batch normalization, and ReLU activations), and $\{W_i\}$ are the corresponding weights. In cases where the dimensions of x and $F(x, \{W_i\})$ differ, a linear projection W_s is applied to the input:

$$y = F(x, \{W_i\}) + W_s x.$$

After the summation, a non-linear activation—most commonly the ReLU function—is applied:

$$y = \mathrm{ReLU}\big(F(x, \{W_i\}) + x\big) \quad \text{or} \quad y = \mathrm{ReLU}\big(F(x, \{W_i\}) + W_s x\big).$$

This residual formulation facilitates more effective gradient flow during backpropagation, thereby mitigating issues such as the vanishing gradient problem that typically hinder the training of deep networks. Consequently, ResNet enables the construction of significantly deeper architectures—often consisting of hundreds of layers—that can learn complex features without a corresponding increase in training difficulty. The practical impact of this design is substantial, as it has set new performance benchmarks across various computer vision tasks, including image classification, object detection, and semantic segmentation, by effectively integrating both low-level and high-level feature representations (He, X. Zhang, et al., 2015).

In contrast to the classical ResNet, the ResNetV2 mentioned in the literature review (section 2) employs a pre-activation scheme, meaning that batch normalization and the activation function (typically ReLU) are applied before the convolutional layers rather than after. This reordering facilitates smoother gradient flow and generally leads to faster convergence and improved performance (He, X. Zhang, et al., 2016).

3.2 Anatomy and Physiology of the Eye

The human eye is a roughly spherical organ with multiple specialized structures. The outermost layer is the cornea, a transparent curved surface at the front that allows light to enter and begins focusing it onto the retina. Behind the cornea is the iris, the colored ring of muscle that adjusts the size of the pupil (the central opening) to regulate how much light enters the eye. Just behind the pupil lies the lens, a transparent, flexible structure that further focuses incoming light by changing shape (accommodation). The focused light is projected onto the retina, the thin layer of neural tissue lining the back of the eyeball.

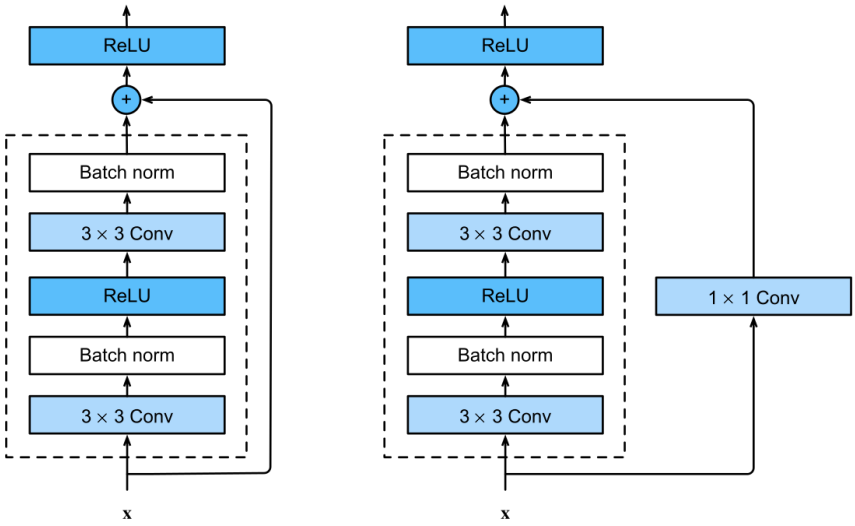


Figure 4: Skip Connection of ResNet (A. Zhang et al., 2023)

The retina contains photoreceptor cells (rods for low-light vision and cones for color and fine detail) that convert light into electrical signals. These visual signals are carried from the retina to the brain by the optic nerve, a thick bundle of over one million nerve fibers. Notably, the retinal image is inverted and reversed, but the brain later interprets and "flips" this image upright as part of visual processing. Surrounding the posterior eyeball is the sclera, the tough white outer coat that maintains the eye's shape and provides protection.

In addition to the eyeball itself, several auxiliary structures support its function. Six extraocular muscles anchor to the sclera and orchestrate the eye's movements within the orbit. These muscles (the medial, lateral, superior, and inferior rectus, and the superior and inferior oblique) allow the eye to rotate up and down, side to side, and torsionally (rolling) to track objects and stabilize gaze. Each muscle is controlled by cranial motor nerves: the lateral rectus is innervated by the abducens nerve, the superior oblique by the trochlear nerve, and the remaining four muscles by the oculomotor nerve.

Another critical structure is the eyelid, a movable fold of skin and muscle (with an upper and lower lid) that protects the eye. The eyelids (or palpebrae) can rapidly close over the eye to shield it from excessive light or foreign objects and open again to restore vision. Blinking of the eyelids also serves to spread tears and mucous secretions across the eye's surface, keeping the cornea moist and clear. The lid's movement is enabled by the levator palpebrae superioris muscle, which lifts the upper lid, and the orbicularis oculi muscle, which encircles the eye and closes the lids. Associated glands (like the lacrimal glands in the upper outer orbit) produce tears that wash across the eye and drain into the nasal cavity, providing lubrication, nourishment, and immune protection to the cornea and conjunctiva. Together, these anatomical structures enable the eye to capture clear images and protect itself in the dynamic environment (Purves, 2001, January).

The physiology of vision begins with phototransduction in the retina and culminates in image perception in the brain. When light hits the photoreceptors in the retinal layer, it is converted into neural impulses. The retina's rod and cone cells synapse onto intermediate neurons and ultimately onto retinal ganglion

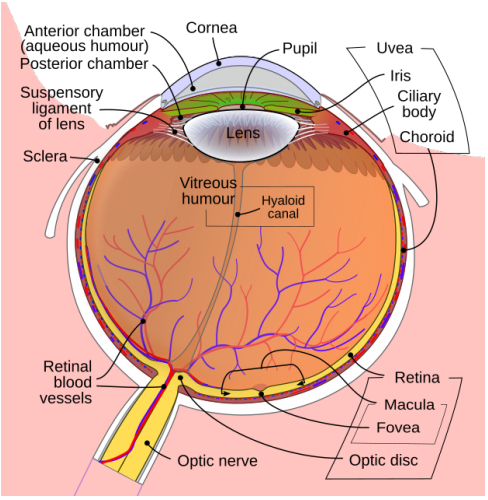


Figure 5: Anatomy of the Eye (Rhcastilhos and Jmarchn, 2007)

cells, whose axons form the optic nerve. The optic nerve transmits the visual information to the brain's visual centers. Along this pathway, signals first reach the lateral geniculate nucleus of the thalamus and then the primary visual cortex in the occipital lobe (and other higher visual areas). It is in the brain that these signals are processed into the images we consciously perceive. Notably, because the optical system of the eye inverts the image on the retina, the brain must reinterpret the image right-side-up. The brain integrates the input from both eyes to create a single three-dimensional percept and also adjusts for changes in lighting and contrast. This complex neural processing allows us to recognize patterns, depth, motion, and color in our environment. The clarity of visual perception depends on the image being sharply focused on the retina (especially the fovea, the central region of the retina with the highest density of cone photoreceptors) and on stable gaze. Any movement of the image on the retina (due to either object motion or eye motion) is quickly compensated by ocular reflexes to prevent blur. Thus, visual perception is intimately linked to the control of eye movements that keep the visual world stable on the retina.

Eye movements are precisely orchestrated by the oculomotor system, which controls the extraocular muscles via neural circuits in the brain. According to Purves (2001, January), there are four primary types of eye movements: saccades, smooth pursuit, vergence, and the vestibulo-ocular reflex (VOR).

Saccades are rapid, ballistic movements that shift the gaze from one point to another, allowing the visual system to quickly sample different regions of the visual scene. In contrast, **smooth pursuit** movements enable the eyes to steadily track moving objects, ensuring that the target remains in focus. **Vergence** movements adjust the angle between the eyes to maintain single binocular vision when viewing objects at different distances, such as converging when focusing on a near object and diverging for distant objects. Finally, the **vestibulo-ocular reflex (VOR)** stabilizes the gaze during head movements by generating compensatory eye movements that counteract the motion, thereby preserving a stable visual image. These eye movements are coordinated by higher cortical areas, such as the frontal eye fields (FEF), which initiate voluntary saccades, and subcortical structures like the superior colliculus (SC), which integrates visual input with motor commands for reflexive responses. Brainstem gaze centers then execute these movements by activating the appropriate cranial nerve nuclei that innervate the extraocular muscles. This integrated control system, which also synchronizes with head movements via the VOR, ensures that the

object of interest is consistently centered on the fovea, thereby facilitating clear and continuous visual perception. This tight integration of diverse eye movements is crucial for everyday tasks, such as reading, driving, or any activity requiring accurate visual tracking.

In light of the research question outlined in section 1, it is pertinent to examine blinking from a biological perspective. Blinking is a semi-automatic action of the eyelids that plays a vital role in maintaining eye health and optimizing vision. A blink consists of a rapid downward and upward movement of the upper eyelid (with a smaller motion of the lower lid), typically completed in a fraction of a second. Humans blink spontaneously about 12 to 20 times per minute under normal conditions, though blink rate can vary with factors like attention, fatigue, or irritation. Each blink serves to protect the eye and lubricate its surface. The simple act of closing the eyelids shields the eye's delicate front surface (the cornea) from potential harm. For example, an approaching object or a sudden bright flash will trigger an involuntary reflex blink in under 0.1 seconds as a defensive response. This reflex is mediated by a neural circuit through the brainstem (the trigeminal nerve senses the threat and the facial nerve induces the orbicularis oculi to contract). Blinking thereby helps prevent foreign particles or intense light from damaging the cornea and internal structures. In the absence of acute threats, spontaneous blinks still occur regularly to perform maintenance functions for the eye. With every blink, a thin layer of tears (the tear film) is spread uniformly across the corneal surface. The tear film consists of an inner mucous layer, a middle aqueous (water) layer, and an outer lipid (oil) layer. This tri-layered film hydrates the corneal epithelium, delivers nutrients (like oxygen) to this avascular tissue, and creates a smooth optical surface for clear vision. By redistributing and refreshing the tears, blinking prevents dryness of the cornea and conjunctiva. It also clears microscopic debris and metabolic waste. In essence, blinking is a natural windshield wiper for the eyes: it keeps the "window" of the cornea clean and moist. Without regular blinks, the tear film would break up, leading to dry spots on the cornea and impaired, uncomfortable vision. Studies confirm that blinking "protects the cornea from damage and prevents drying of the eye" (Drew et al., 2019). For example, when people concentrate on computer screens, their blink rate often decreases, contributing to dry eye symptoms; this underscores how crucial blinking is for ocular surface health.

4 Data & Preprocessing

4.1 The MRL Eye Dataset

The data used to train the models is drawn from the dataset proposed by Fusek (2018), which comprises 84,898 eye region images collected from 37 distinct individuals (33 men and 4 women). These images were captured under real-world driving conditions using three different near-infrared (NIR) cameras with resolutions of 640×480, 1280×1024, and 752×480 pixels, respectively. This multi-sensor acquisition setup, combined with the dynamic in-car environment, introduces significant variability in lighting, head poses, individuals with and without glasses (see Figure 6) and overall image quality.

In addition to the natural diversity inherent in the data collection process, the dataset is accompanied by detailed annotations. Each image is systematically labeled according to several attributes, including subject identity, gender, presence of eyewear, eye state (open or closed), and the degree of reflections.

Overall, the combination of robust image acquisition, extensive annotations, and almost perfectly balanced class distribution (see Table 1) renders this dataset an invaluable resource for advancing research in driver monitoring, gaze estimation, and related applications.

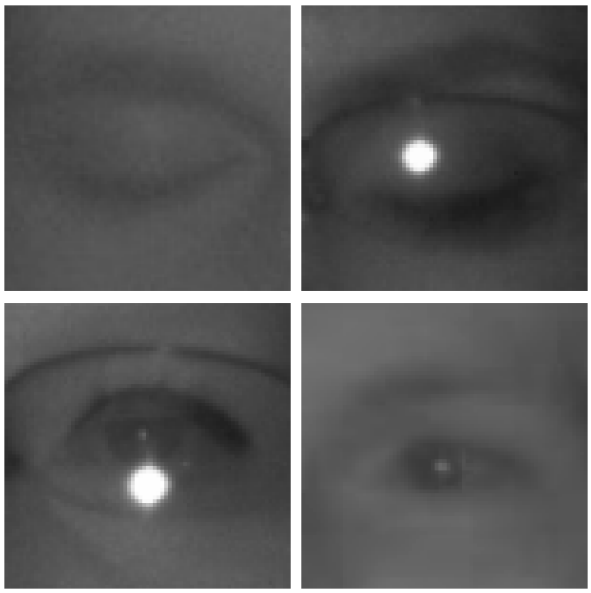


Figure 6: Sample images from the dataset

Label	Count	Fraction
Closed (0)	41946	49.40%
Open (1)	42952	50.60%

Table 1: Distribution of Training Data

4.2 Preprocessing

The dataset is provided as pre-cropped eye region images (see Figure 6) with corresponding labels, thereby significantly reducing the need for extensive preprocessing. Additionally, the images are organized into subdirectories based on their labels—specifically, "closed" and "open"—which streamlines their integration and loading into PyTorch:

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

dataset = ImageFolder(
"/kaggle/input/mrl-eyes/data",
transform=train_transforms)
```

When using PyTorch's `ImageFolder` class, the labeling mechanism is intrinsically tied to the directory structure. Specifically, `ImageFolder` automatically assigns class labels based on the names of the subdirectories within the specified root directory—each subdirectory is treated as a distinct class, and

all images contained within that subdirectory inherit the corresponding label. In a scenario where the dataset does not naturally conform to this subdirectory format, it becomes necessary either to reorganize the dataset to reflect the different classes (for instance, by placing images in separate folders named according to their class) or to develop a custom `ImageFolder` class that assigns labels based on alternative criteria, such as parsing filenames or referencing an external annotation file. Moreover, the classical approach using a custom `Dataset` class typically requires several additional lines of code to manually handle tasks such as file path management, image loading, and label assignment. In contrast, `ImageFolder` streamlines these processes by leveraging the inherent structure of the file system, thereby reducing development overhead and potential sources of error to basically a one-liner.

5 Model Training

5.1 Data Augmentation

Data augmentation is crucial for CNNs as it significantly enhances the robustness and generalization capabilities of the network—i.e. it prevents overfitting (Krizhevsky et al., 2012). By artificially expanding the training dataset through operations such as rotations, scaling, translations, and brightness adjustments, data augmentation simulates a wider variety of real-world conditions. This is especially important for eye images, where subtle variations in lighting, angle, and occlusion—such as reflections from glasses or partial eyelid closures—can otherwise lead to overfitting on the limited available data. As a result, incorporating data augmentation enables the CNN to learn invariant and discriminative features that are essential for reliably distinguishing between open and closed eyes, ultimately improving the model's performance on unseen data.

Before applying data augmentation, the per-pixel mean and standard deviation were computed for all images uniformly resized to 64×64 pixels. These statistics were then used to normalize the dataset in a final transformation step, as illustrated in the following pipeline:

```
from torchvision.transforms import v2 as T

train_transforms = T.Compose([
    T.Grayscale(num_output_channels=1),
    T.Resize((64, 64)),
    T.RandomHorizontalFlip(p=0.5),
    T.RandomRotation(degrees=10),
    T.ColorJitter(brightness=0.1, contrast=0.1),
    T.RandomAffine(degrees=5,
        translate=(0.05, 0.05), scale=(0.95, 1.05)),
    T.ToTensor(),
    T.Normalize(mean=mean, std=std),
])
```

In Figure 7 is illustrated what the model is seeing in the initial layer. What the individual augmentations are doing is the following:

- **RandomHorizontalFlip:** Flips the image horizontally with a specified probability (e.g., 50%). This increases data variability and helps the model become more robust to left-right orientation changes.
- **RandomRotation:** Rotates the image by a random angle within a defined range (e.g., $\pm 10°$). This simulates variations in viewpoint or camera angle, thereby improving the model's generalization capabilities.
- **ColorJitter:** randomly modifies the brightness and contrast of images within a $\pm 10\%$ range, enhancing robustness to variations in lighting conditions
- **RandomAffine:** Additionally rotates the picture by $\pm 5°$, shifts the images vertically & horizontally by 5% and scales it by a factor selected from the interval $[0.95, 1.05]$ randomly
- **ToTensor:** Converts the PIL or NumPy image into a PyTorch tensor, typically normalizing pixel values to the $[0, 1]$ range. This step is essential for further processing by PyTorch.

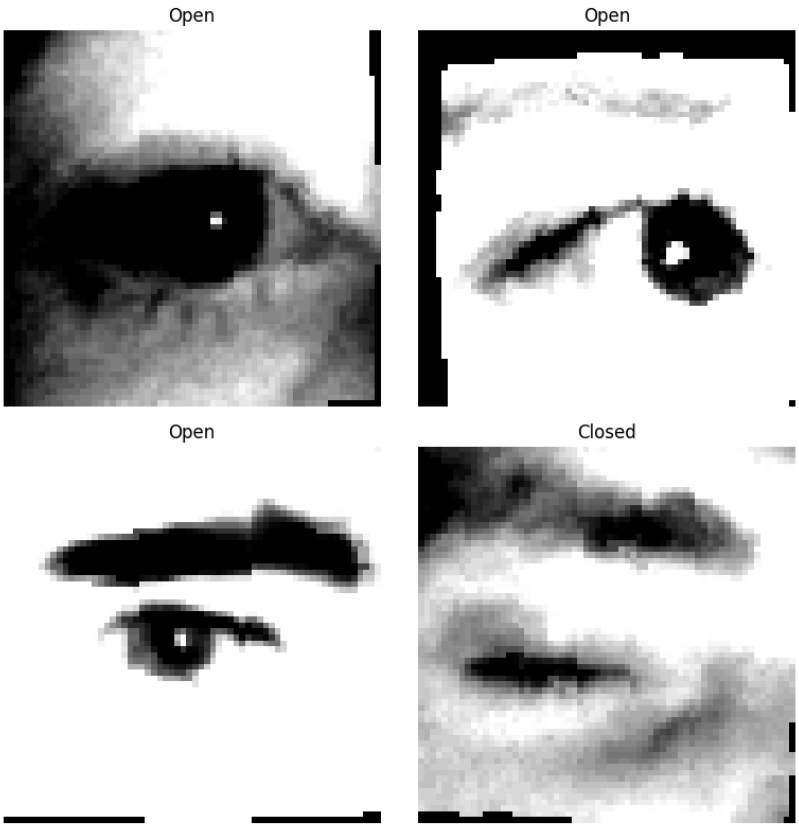


Figure 7: Augmented Images Sample

5.2 Details of Learning

My baseline model is implemented as a plain CNN. Its feature extractor comprises three convolutional blocks, each consisting of a convolutional layer (with increasing channel depth), an exponential linear

unit (ELU) activation, a batch normalization layer, and a max pooling operation. Additionally, dropout is introduced after each pooling step to mitigate overfitting by randomly deactivating a fraction of the neurons. An adaptive average pooling layer followed by flattening is applied to reduce the final feature maps to a one-dimensional representation. Finally, a fully connected layer (without an additional activation) projects the extracted features to a single logit output, making it suitable for binary classification via a sigmoid-based loss.

```
import torch
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ELU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(p=0.25),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ELU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(p=0.25),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ELU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(p=0.25),

            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
        )
        self.classifier = nn.Sequential(
            nn.Linear(128 * 1 * 1, 1)
        )

    def forward(self, x):
        x = self.feature_extractor(x)
        x = self.classifier(x)
        return x
```

The network was trained for up to 100 epochs using the Adam optimizer with a learning rate of 0.001 and a binary cross-entropy loss with logits (BCEWithLogitsLoss). Different batch sizes (128, 256 and 1024) are systematically compared to investigate their impact on convergence and generalization. During each

epoch, the model parameters are updated in mini-batches through backpropagation, while the running average of the training loss is monitored. Validation is conducted after every epoch, with the validation loss serving as the key metric for model selection.
To prevent overfitting, several regularization techniques are employed. These include dropout (p=0.25) in the convolutional blocks, batch normalization after every activation function, and an early stopping mechanism that halts training when the validation loss fails to improve for ten consecutive epochs. The best-performing model parameters are tracked throughout training and restored once early stopping is triggered.
For each batch size configuration, the training and validation losses are recorded at every epoch. This facilitates detailed post-hoc analysis of learning dynamics. Upon completion of training—or earlier if early stopping is invoked—the best model weights are saved to disk. This systematic approach ensures reproducibility and enables direct comparisons between different hyperparameter settings and architectures.

In summary, the presented CNN serves as a robust baseline model, complementing subsequent experiments with deeper architectures such as ResNets. Its design balances simplicity and efficacy, integrating standard techniques—batch normalization, dropout, and early stopping—to enhance both training stability and generalization performance.[3]

For training ResNet50 and ResNet152, you can simply use the pre-defined architectures provided by the PyTorch library. However, you may need to make certain manual adjustments—such as modifying the input or output layers—to accommodate your specific training requirements:

```
import torchvision

model = torchvision.models.resnet50()
model.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=7,
    stride=2, padding=3, bias=False)
model.fc = nn.Linear(2048, 1)
```

In addition to training the models entirely from scratch, transfer learning was employed by initializing the models with pre-trained ImageNet weights. ImageNet is a large-scale dataset containing over 14 million images annotated across thousands of categories, and it has become a benchmark for visual recognition tasks. Pre-training on such a diverse dataset allows the model to learn robust, general-purpose feature representations that can be effectively fine-tuned for more specific tasks, leading to improved convergence and performance. This initialization can be implemented as follows:

```
import torchvision
from torchvision.models import ResNet50_Weights

model = torchvision.models.resnet50(weights=ResNet50_Weights.DEFAULT)
```

A more effective transfer learning strategy for tasks focusing on the human face—such as blink detec-

[3] I wanted to add, that the model training was performed using Kaggle's free accelerator, the NVIDIA TESLA P100 GPU, which provided the necessary computational resources for efficient training.

tion—may be to initialize the models with weights pre-trained on datasets that emphasize facial features. However, such specialized weights are not always readily available. This additional step can equip the model with robust facial representations that are directly relevant to blink detection, potentially improving convergence speed and overall performance during subsequent fine-tuning.

The training loop of the ResNet is similarly to the plain CNN. It is designed to run for a maximum of 100 epochs with an initial learning rate of 0.001, while also exploring the impact of different batch sizes. For each batch size, training and validation datasets were loaded using PyTorch DataLoader objects, with parallel data loading configured via the appropriate number of worker threads. The model was first moved to the designated device (P100 GPU) and initialized accordingly. Similarly I utilized a binary cross-entropy loss with logits (BCEWithLogitsLoss) to accommodate the binary classification nature of blink detection, and the Adam optimizer was chosen for parameter updates.
A learning rate scheduler (ReduceLROnPlateau) was integrated into the training loop to monitor the validation loss and reduce the learning rate by a factor of 0.1 if no improvement was observed over five consecutive epochs (with a threshold of 1e-3). Additionally, early stopping was implemented with a patience of 10 epochs, ensuring that training was halted if the validation loss failed to improve within that period, thereby preventing overfitting.
Within each epoch, the model was set to training mode and the cumulative loss over the training batches was computed and averaged. Subsequently, the model was switched to evaluation mode to calculate the average validation loss without gradient computation. At the end of each epoch, the learning rate scheduler was updated based on the validation loss, and any improvement in validation performance resulted in the current model weights being saved as the best model state. Conversely, if no improvement was detected, an internal counter was incremented, and training ceased once this counter exceeded the defined patience.

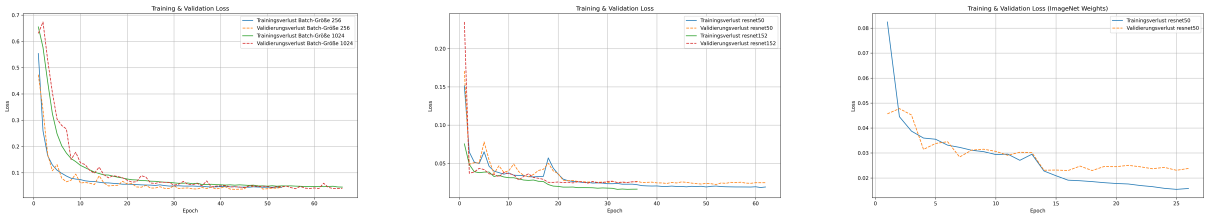
Finally, upon completion of training for a given batch size, the best-performing model weights were reloaded, and the training losses, validation losses, as well as the learning rate history, were saved to disk for further analysis.

5.3 Model Evaluation

The performance of the three investigated architectures—Plain CNN, ResNet50, and ResNet152—was evaluated on the validation set, with results summarized in Figure 8 and Figure 9 as well as Table 2. Overall, all models demonstrated high accuracy in distinguishing between “Closed” and “Open” eyes, as indicated by their confusion matrices. The Plain CNN already performs at a strong baseline level, achieving an accuracy of 0.9872, yet the deeper ResNet variants further improve classification metrics. In particular, the ResNet50 initialized with pre-trained ImageNet weights achieved the highest overall accuracy (0.9924) and F1-score (0.9926), indicating a more balanced performance across both classes.

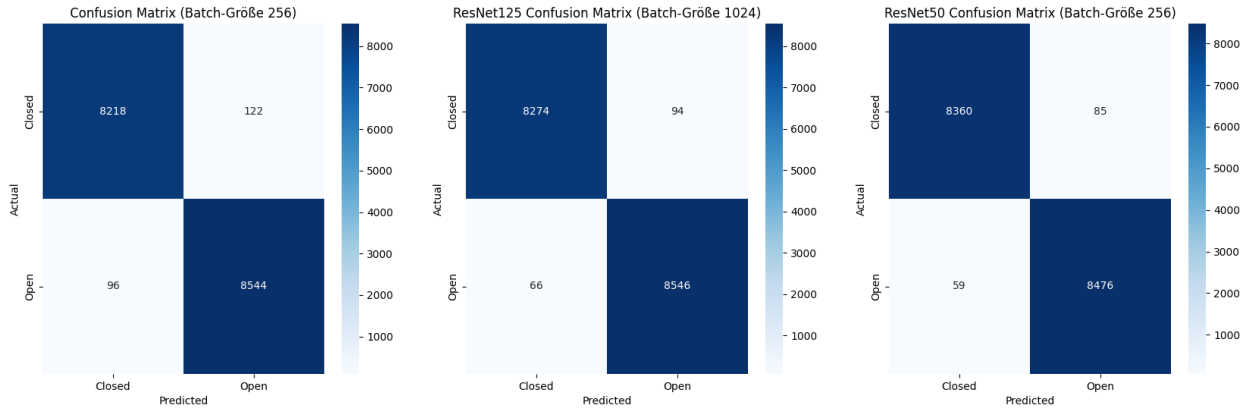
6 Model Application

This chapter details the actual application by first describing the experimental setup and associated dataset, then applying the models on the data derived from the experiment—thereby serving as the quasi



(a) Plain CNN loss curves **(b)** ResNets loss curves **(c)** Transfer learning loss curves

Figure 8: Comparison of loss curves across different network architectures and training strategies.



(a) Plain CNN **(b)** ResNet152 **(c)** ResNet50

Figure 9: Comparison of Confusion Matrices

test split—subsequently evaluating their performance, and finally employing the best-performing model to detect blinks, thus demonstrating that a simple webcam can be effectively utilized in a real-world research setting.

6.1 Experiment data

The data to really test the models is derived from an experiment lead and designed by my supervisors Dr. Alexander Silbersdorf and Ruven Zapf.

The experimental setup was designed to replicate the online learning environment experienced by a typical student engaging with digital educational content. As depicted in Figure 12, the subject was recorded in a university room while seated at a desktop computer and watching a 36-minute lecture on statistics. The procedure alternated between calibration, evaluation, and lecture viewing segments. A Logitech C920 HD webcam—operating at 30 frames per second and capturing video at a resolution of 1920 × 1080 pixels—was used in conjunction with a 27-inch monitor of identical resolution. During the calibration and evaluation phases, the participant was instructed to follow a red dot displayed on a black background, with the movement pattern of the dot varying across segments, to extract gaze point coordinates. The data acquired from this experiment were subsequently utilized to train CNNs for *(a)* predicting gaze points and *(b)* for blink detection, serving as a complement to the former.

Model	Accuracy	Precision	Recall	F1-Score
Plain CNN	0.9872	0.9859	0.9889	0.9874
ResNet50	0.9915	0.9901	0.9931	0.9916
ImageNet-ResNet50	0.9924	0.9906	0.9946	0.9926
ResNet152	0.9906	0.9891	0.9923	0.9907
ImageNet-ResNet152	0.9909	0.9887	0.9937	0.9912

Table 2: Evaluation metrics

Figure 10: Me participating in the experiment

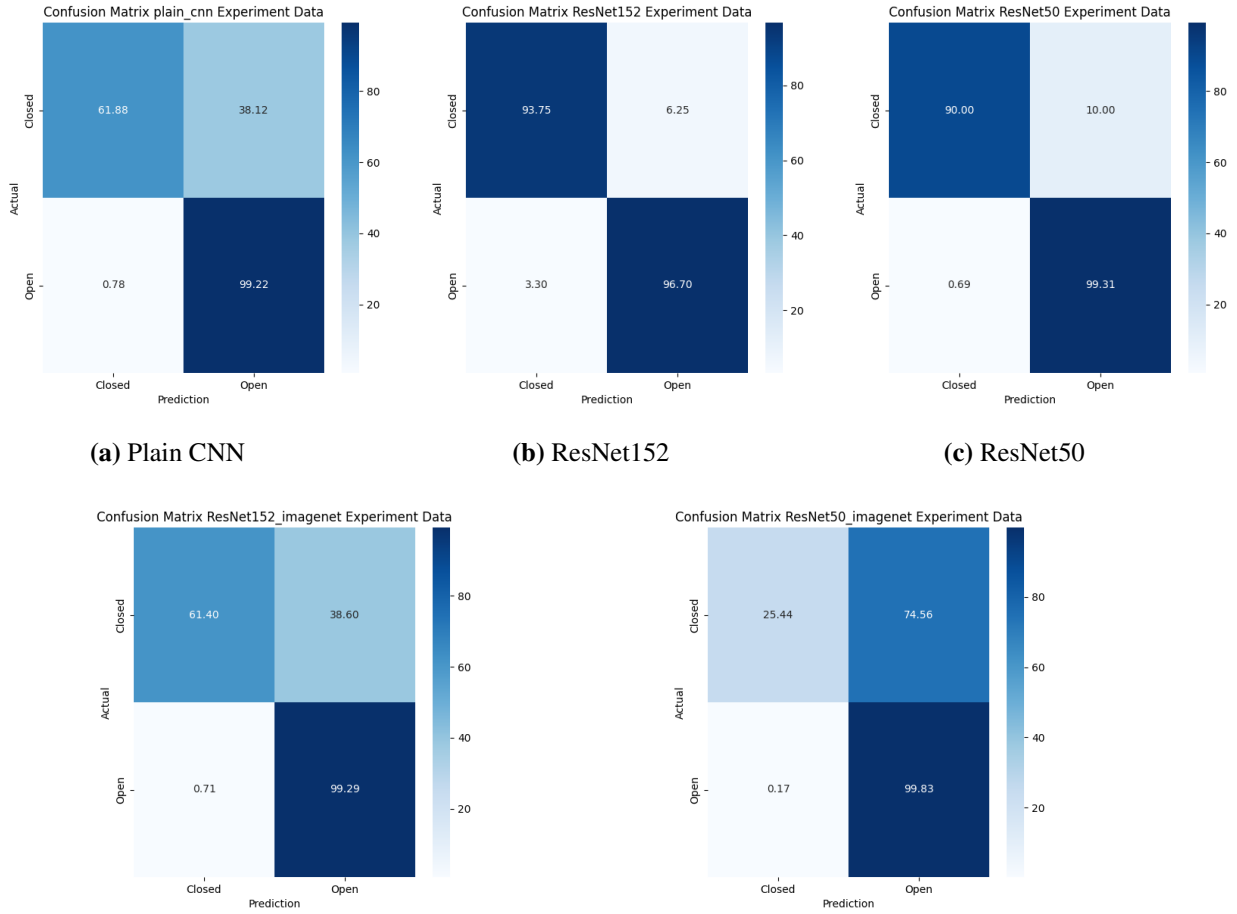
For my purposes, I utilized single frames extracted from the learning video, where my supervisor kindly performed manual cropping and labeling of the eye regions. This was achieved using Google's MediaPipe library, which employs facial landmark detection by overlaying a mesh of 468 predefined points onto the detected face (this is similar to the dlib approach mentioned in section 2). Among these points, specific landmarks around the eye regions were used to precisely crop out consistent eye areas from the original images. This manual preprocessing and labeling were conducted for the initial 20,000 frames, corresponding to ≈ 11.11 minutes of video recorded at 30 frames per second.

6.2 Model Evaluation

The models are fed the transformed experiment images, which were exposed to this base transformations:

```
from torchvision.transforms import v2 as T

base_transforms = T.Compose([
    T.Grayscale(num_output_channels=1)
    T.Resize((64, 64)),
    T.ToTensor(),
    T.Normalize(mean=mean, std=std)
])
```



(a) Plain CNN **(b)** ResNet152 **(c)** ResNet50

(d) ImageNet-ResNet152 **(e)** ImageNet-ResNet50

Figure 11: Comparison of Confusion Matrices applied to the experiment data

In highly unbalanced datasets, conventional performance metrics such as accuracy can be artificially inflated, potentially obscuring deficiencies in the detection of minority class instances. This "phenomenon" is evident in our experimental data and reflects a general challenge in eye blink detection, where blink events naturally occur far less frequently than open-eye states. The experiment dataset comprises 19,829 "open" eyes compared to only 171 "closed" eye images. Mathematically, accuracy is given by:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN},$$

where TP represents true positives, TN true negatives, FP false positives, and FN false negatives. Accuracy is therefore defined as the proportion of correct predictions relative to the total number of predictions. In an unbalanced dataset, a classifier that predominantly predicts the majority class can yield high accuracy, even if it performs poorly on the minority class. [4] This issue becomes evident when examining the plain CNN's (as well as the ResNets initialized with the ImageNet weights) accuracy in Table 3 and its confusion matrix in Figure 11a, where a relatively high overall accuracy masks the substantial pro-

[4]Note that the positive class (i.e., open eyes) was assigned the label '1'. Consequently, the true positives appear in the bottom-right cell (open predicted as open), rather than in the top-left corner as is sometimes conventionally shown.

portion of closed-eye instances misclassified as open.
To address this limitation, the **Matthews Correlation Coefficient (MCC)** is used as it provides a more balanced evaluation by considering all four elements of the confusion matrix. The MCC is defined as:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}},$$

and takes values in the interval $[-1, 1]$. A value of MCC $= +1$ indicates perfect prediction, MCC $= 0$ corresponds to performance no better than random guessing, and MCC $= -1$ signifies total disagreement between predictions and the actual classes. This formula effectively computes a correlation coefficient between predicted and actual classes, capturing the balance between correct and incorrect classifications for *both* the majority and minority classes. By including cross-terms $(TP \cdot TN)$ and $(FP \cdot FN)$, the MCC penalizes models that inflate their accuracy by simply predicting the majority class. In other words, even if a classifier achieves high overall accuracy by overwhelmingly favoring the dominant class, its MCC will remain low unless it can also correctly identify the minority class. Consequently, MCC offers a more robust assessment of model performance in skewed datasets, mitigating the limitations of accuracy-based evaluations.

As shown in Figure 9, ResNet50 achieves the highest MCC among the evaluated models, indicating superior overall performance, particularly in correctly classifying both open and closed eyes. The relatively high MCC value confirms that ResNet50 effectively handles the imbalanced dataset by not only predicting the dominant class (open eyes) but also accurately recognizing the minority class (closed eyes). This makes it more reliable for real-world scenarios in blink detection compared to the other tested architectures. Consequently, ResNet50 will be utilized for the subsequent blink detection.

Model	Accuracy	MCC	Precision	Recall	F1-Score
Plain CNN	0.9893	0.4870	0.9969	0.9922	0.9946
ResNet50	0.9923	0.6760	0.9992	0.9931	0.9961
ImageNet-ResNet50	0.9920	0.3769	0.9936	0.9983	0.9960
ResNet152	0.9668	0.4102	0.9995	0.9670	0.9830
ImageNet-Resnet152	0.9896	0.5064	0.9995	0.9670	0.9830

Table 3: Comparison of classification metrics across different network architectures.

6.3 Webcam-based blink detection

Based on the predictions made by the ResNet50, a pandas dataframe of the results was produced, which looks like this:

	ID	Eye	Label	Prediction	Probability
0	0	left	1	1	0.995454
1	0	right	1	1	0.995353
2	1	left	0	0	0.464313
3	1	right	0	0	0.040280
4	2	left	0	0	0.027213
...	...	...	...	...	...
39995	19997	right	1	1	0.996776
39996	19998	left	1	1	0.999968
39997	19998	right	1	1	0.995408
39998	19999	left	1	1	0.999967
39999	19999	right	1	1	0.995522

Table 4: Results Dataframe

Table 4 comprises six columns. The first column represents the index of each entry, while the `ID` column preserves the original sequential order of the frames. For instance, the first and second rows of the displayed dataframe correspond to the cropped eye pair—left and right, as indicated by the `Eye` column—of the very first frame of the learning experiment video. The `Label` column contains the manually assigned classification, whereas the `Prediction` column records the model's inferred class, determined by the probability value displayed in the `Probability` column.
Now we need to find rules to what defines an actual blink in the data. As mentioned in section 2 there are several methods to do this and these methods come with some arbitrariness. Looking into the literature we can find that for example Elleuch and Wali (2019) state, that a blink can last 100ms-400ms and that Alsaeedi and Wloka (2019) states that during the blink process the closed eye state makes up approximately 40 % of the whole process time. Taking the average of this and remembering that our webcam records with 30 fps, we can derive the following:

$$1\,\text{Blink} \approx 100\,\text{ms} \quad \Longrightarrow \quad \frac{30\,f}{1000\,\text{ms}} \cdot 100\,\text{ms} = 3\,f$$

I applied a rather *naive* approach in which a blink event is defined by a fixed threshold—in our case, three consecutive frames classified as "closed" constitute a blink. Using this definition, we visualize the resulting blink frequency over the first 11 minutes of the experiment:

From Figure 12, we observe that the blink frequency ranged from 0 to a maximum of 10 blinks per minute. Comparing these results to the human average blink rate previously discussed in subsection 3.2, it is evident that the measured frequency consistently remains below this average. This observation can be partly attributed to the initial calibration phase occupying roughly the first five minutes of recording, as well as potential influences from dry eye symptoms, a common phenomenon reported by de la Cruz et al. (2024) in similar experimental settings.

The blink frequency presented in this analysis was calculated based on the state of a single eye. This calculation inherently assumes that both eyes exhibit identical states simultaneously—that is, if one eye

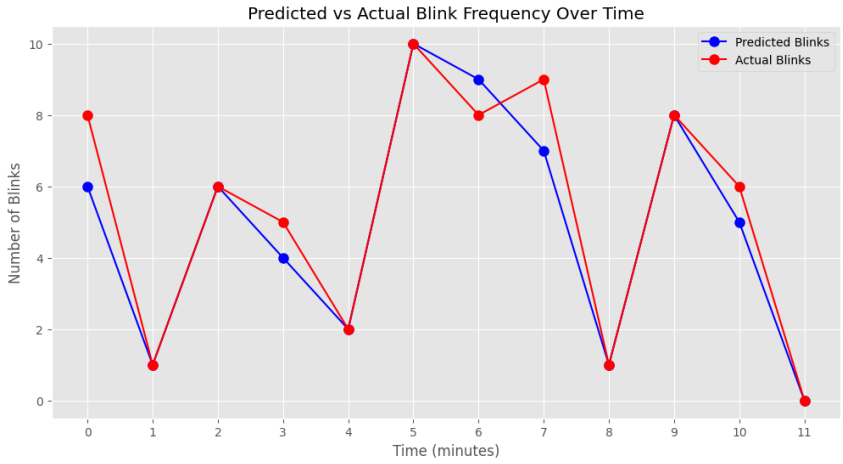


Figure 12: Blink frequency during the first 11 minutes of the experiment

is labeled as closed, the other is presumed closed as well, and vice versa. While generally accurate in typical physiological contexts, this assumption may not hold universally, especially in cases of asymmetrical blinking or partial closure due to fatigue, medical conditions, or measurement noise.
Figure 13 illustrates sample predictions, providing insights into classification performance at the frame level. Each pair of images represents a snapshot of both eyes at the same timestamp, including the true label, the predicted label, and the associated probability. The top row shows a correct classification of open eyes (label 1), with high certainty indicated by probabilities close to 1.00. The middle row highlights a misclassification, where the right eye is labeled as open but incorrectly predicted as closed with relatively low confidence (probability of 0.20). Finally, the bottom row demonstrates correct classification of closed eyes (label 0), also with high confidence.

It is noteworthy that such mismatched predictions account for only 1.11% of the evaluation data, highlighting robust overall classification performance. However, relying on single-eye classifications introduces vulnerability, as occasional mismatched predictions could significantly affect the blink-frequency calculation. Future improvements should therefore focus on integrating bilateral eye information to enhance robustness and accuracy.

7 Discussion

The ResNet50 model initialized with ImageNet pre-trained weights achieved the highest performance according to conventional metrics, such as accuracy. However, due to the highly imbalanced nature of the experimental dataset, these metrics require careful interpretation. A primary limitation of traditional metrics like accuracy is their vulnerability to inflation in imbalanced scenarios, as classifiers heavily biased toward the majority class can still exhibit misleadingly high accuracy. Thus, relying solely on accuracy can obscure substantial classification errors, especially for the minority class.
An essential consideration is the approach used for blink detection. The naive method employed in this study—identifying blinks based on a fixed threshold of three consecutive frames classified as "closed"—is straightforward but has clear limitations. Such a rigid threshold does not accommodate natural variability in blink duration and frequency, potentially missing blinks shorter or longer than the predetermined

Figure 13: Prediction samples

threshold. This rigidity can result in underestimating true blink events, which may have significant implications in practical scenarios, such as driver fatigue monitoring and attention tracking tasks, where accurate blink detection is crucial.
Moreover, the use of purely CNNs, despite their effectiveness in spatial feature extraction, does not inherently capture temporal dynamics. Thus, CNN-based methods might overlook critical temporal information essential for accurately characterizing blinks. Future methodologies should incorporate temporal models such as Long Short-Term Memory networks or other RNN architectures. These temporal models excel at capturing sequential dependencies and subtle temporal variations in blink patterns, thereby enhancing the accuracy and robustness of blink detection.

In conclusion, while the ResNet50 model demonstrates promising performance, future studies should critically consider the limitations identified here. Even though some criticisms can be made regarding

model selection and the blink identification methodology, this approach serves as a valuable proof-of-concept demonstrating feasibility in utilizing deep learning techniques for blink detection using webcam-based data. Future improvements should focus on more flexible blink detection criteria, diversified data collection to increase generalizability, and incorporating temporal modeling techniques to further enhance predictive performance and reliability.

List of References

Alsaeedi, N., & Wloka, D. (2019). Real-time eyeblink detector and eye state classifier for virtual reality (vr) headsets (head-mounted displays, hmds). *Sensors*, *19*(5), 1121. https://doi.org/10.3390/s19051121

Anas, E. R., Henriquez, P., & Matuszewski, B. J. (2017). Online eye status detection in the wild with convolutional neural networks. *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 88–95. https://doi.org/10.5220/0006172700880095

Ata, F., Ayturan, K., Hardalaç, F., & Kutbay, U. (2024). Deep learning-based eye blink detection in video frames: Performance analysis of various models. *Preprints*. https://doi.org/10.20944/preprints202408.0597.v1

Cortacero, K., Fischer, T., & Demiris, Y. (2019). Rt-bene: A dataset and baselines for real-time blink estimation in natural environments. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 1159–1168. https://doi.org/10.1109/iccvw.2019.00147

Culemann, W., Neuber, L., & Heine, A. (2023). Pupil vs. eyelid: Evaluating the accuracy of blink detection in pupil-based eye tracking devices [27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023)]. *Procedia Computer Science*, *225*, 2008–2017. https://doi.org/https://doi.org/10.1016/j.procs.2023.10.191

de la Cruz, G., Lira, M., Luaces, O., & Remeseiro, B. (2024). Eye-lrcn: A long-term recurrent convolutional network for eye blink completeness detection. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(4), 5130–5140. https://doi.org/10.1109/TNNLS.2022.3202643

Drew, P. J., Winder, A. T., & Zhang, Q. (2019). Twitches, blinks, and fidgets: Important generators of ongoing neural activity [PMID: 30311838]. *The Neuroscientist*, *25*(4), 298–313. https://doi.org/10.1177/1073858418805427

Elleuch, H., & Wali, A. (2019). Unwearable multi-modal gestures recognition system for interaction with mobile devices in unexpected situations. *IIUM Engineering Journal*, *20*(2), 142–162. https://doi.org/10.31436/iiumej.v20i2.1000

Frank, G. K. W., Kalina, C., DeGuzman, M. C., & Shott, M. E. (2020). Eye blink and reward prediction error response in anorexia nervosa. *International Journal of Eating Disorders*, *53*(9), 1544–1549. https://doi.org/10.1002/eat.23332

Fusek, R. (2018). Pupil localization using geodesic distance. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11241 LNCS*, 433–444. https://doi.org/10.1007/978-3-030-03801-4_38

Glosser.ca. (2013). Artificial neural network with layer coloring [Illustration of an artificial neural network. Licensed under CC BY-SA 3.0. Retrieved from https://upload.wikimedia.org/wikipedia/commons/4/46/Colored_neural_network.svg].

Godoy, D. V. (2023a). Dl-visuals [Licensed under CC BY 4.0. Retrieved from https://commons.wikimedia.org/w/index.php?curid=150823503].

Godoy, D. V. (2023b). Dl-visuals [Licensed under CC BY 4.0. Retrieved from https://commons.wikimedia.org/w/index.php?curid=150823502].

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988. https://doi.org/10.1109/ICCV.2017.322

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. https://doi.org/10.48550/ARXIV.1512.03385

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. https://doi.org/10.48550/ARXIV.1603.05027

Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1867–1874. https://api.semanticscholar.org/CorpusID:2031947

Kim, J., Sunil Kumar, Y., Yoo, J., & Kwon, S. (2018). Change of blink rate in viewing virtual reality with hmd. *Symmetry*, *10*(9). https://doi.org/10.3390/sym10090400

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Lalonde, M., Byrns, D., Gagnon, L., Teasdale, N., & Laurendeau, D. (2007). Real-time eye blink detection with gpu-based sift tracking. *Fourth Canadian Conference on Computer and Robot Vision (CRV '07)*, 481–487. https://doi.org/10.1109/CRV.2007.54

Mazzone, L., Yu, S., Blair, C., Gunter, B. C., Wang, Z., Marsh, R., & Peterson, B. S. (2010). An fmri study of frontostriatal circuits during the inhibition of eye blinking in persons with tourette syndrome. *American Journal of Psychiatry*, *167*(3), 341–349. https://doi.org/10.1176/appi.ajp.2009.08121831

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/bf02478259

Minsky, M., & Papert, S. (1969). *Perceptrons; an introduction to computational geometry*. MIT Press. https://books.google.de/books?id=Ow1OAQAAIAAJ

Pahariya, S., Vats, P., & Suchitra, S. (2024). Driver drowsiness detection using mobilenetv2 with transfer learning approach. *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 1–6. https://doi.org/10.1109/ADICS58448.2024.10533606

Purves, D. (2001, January). *Neuroscience* (D. Purves, G. J. Augustine, D. Fitzpatrick, C. Katz Lawrence, A.-S. Lamantia, J. O. McNamara, & S. M. Williams, Eds.; 2nd ed.). Sinauer Associates.

Rhcastilhos & Jmarchn. (2007). Schematic diagram of the human eye [Accessed: 2025-03-15].

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Sanyal, R., & Chakrabarty, K. (2019). Two stream deep convolutional neural network for eye state recognition and blink detection. *2019 3rd International Conference on Electronics, Materials Engineering Nano-Technology (IEMENTech)*, 1–8. https://doi.org/10.1109/IEMENTech48150.2019.8981102

Schillingmann, L., & Nagai, Y. (2015). Yet another gaze detector: An embodied calibration free system for the icub robot. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 8–13. https://doi.org/10.1109/HUMANOIDS.2015.7363515

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. https://doi.org/10.48550/ARXIV.1412.6806

Xiong, J., Dai, W., Wang, Q., Dong, X., Ye, B., & Yang, J. (2025). A review of deep learning in blink detection. *PeerJ Computer Science*, *11*, e2594. https://doi.org/10.7717/peerj-cs.2594

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into deep learning* [https://D2L.ai]. Cambridge University Press.

Zhao, L., Wang, Z., Zhang, G., Qi, Y., & Wang, X. (2017). Eye state recognition based on deep integrated neural network and transfer learning. *Multimedia Tools and Applications*, *77*(15), 19415–19438. https://doi.org/10.1007/s11042-017-5380-8

Declaration of Authorship

Webcam-based Eyetracking using Deep Learning Approaches

I hereby declare that the present paper is entirely my own work and without the use of any unauthorised assistance. Any content which has been taken verbatim or paraphrased from other sources has been identified as such. This paper has not been submitted in any form whatsoever to an examining body. Previously published work has been cited as such.

Göttingen, March 15, 2025 Henning Siept Dinkela

Erklärung zur Nutzung von ChatGPT und vergleichbaren Werkzeugen im Rahmen von Prüfungen

In der hier vorliegenden Arbeit habe ich ChatGPT oder eine andere KI wie folgt genutzt:

× gar nicht

✓ bei der Ideenfindung

✓ bei der Erstellung der Gliederung

× zum Erstellen einzelner Passagen, insgesamt im Umfang von . . . % am gesamten Text

✓ zur Entwicklung von Software-Quelltexten

✓ zur Optimierung oder Umstrukturierung von Software-Quelltexten

✓ zum Korrekturlesen oder Optimieren

× Weiteres, nämlich: . . .

Ich versichere, alle Nutzungen vollständig angegeben zu haben. Fehlende oder fehlerhafte Angaben werden als Täuschungsversuch gewertet.