Real-Time Robot Localization on the Botball Table Using a USB Camera and CNNs

Tobias Madlberger[1], Leander Klik, Johanna Pucher
HTL St. Pölten, Austria
[1]Corresponding author's email: tobias.madlberger@gmail.com

***Abstract*—Accurate robot localization is essential for autonomous navigation, especially in resource-constrained environments like the Botball competition. This paper presents a vision-based localization system using a USB camera and a lightweight convolutional neural network (CNN) to estimate the robot's position and orientation in real-time. A dataset was collected using synchronized onboard and overhead cameras with ArUco markers for ground-truth labeling. The optimized model runs efficiently on a Raspberry Pi 3 Model B+, achieving an average inference time of 0.0773 seconds per frame with some accuracy loss after INT8 quantization. While real-time performance was achieved, accuracy limitations may restrict direct competition usage without further refinement.**

***Index Terms*—Vision-Based Localization, Visual Odometry, Pose Estimation, USB Camera, Raspberry Pi, Botball**

I. Introduction

Accurate robot localization is vital for autonomous navigation and interaction within a defined environment. In the Botball competition, traditional methods like LiDAR and GPS are either unavailable or impractical, making vision-based localization a compelling alternative [1], [2]. By using a USB camera, robots can capture environmental images and estimate their position and orientation through visual data processing [3].

This paper presents a real-time vision-based localization system using a lightweight convolutional neural network (CNN) to predict the robot's pose (x, y, θ) on the Botball table. The approach focuses on achieving high accuracy and efficiency on resource-constrained hardware, such as the Raspberry Pi 3 Model B+ [4]. Unlike prior classification-based methods [5], this work targets continuous pose regression, enabling more precise navigation. The system is trained on a dataset collected through synchronized camera feeds with ArUco marker-based ground-truth labeling and is validated under varying lighting conditions to ensure robustness.

The primary goal is to demonstrate how a cost-effective, camera-based solution can achieve reliable localization in real-time despite hardware and competition constraints.

II. State of the Art

A. Overview of Robot Localization

Robot localization is a fundamental challenge in autonomous robotics, as it enables precise navigation and task execution. Various methods have been developed, each with distinct advantages and limitations:

- **LiDAR**: Provides highly accurate distance measurements but is costly, computationally intensive, and not included in the Botball kit.
- **GPS**: Performs well outdoors but lacks the precision required for indoor environments and is prohibited in Botball competitions [2].
- **Wheel Odometry**: Estimates movement through encoder readings (or, in the case of Botball, motor tick estimation via back-EMF) [3].
- **IMU (Accelerometer & Gyroscope)**: Estimates position and orientation by integrating acceleration and rotational data. However, due to sensor noise and drift, errors can accumulate over time, making standalone IMU-based localization less reliable.

Given these limitations, vision-based localization has emerged as a promising alternative. By leveraging cameras to capture environmental features, it is possible to estimate the robot's position relative to known landmarks [4].

B. Vision-Based Localization Approaches

Traditional feature-based localization methods depend on detecting visual cues such as edges and corners [10]. However, the Botball playing field poses challenges due to the scarcity of distinct landmarks. While tracking elements like black lines or PVC pipes is possible, the camera may capture featureless sections of the FRP plate, resulting in unreliable localization. This issue is compounded by the presence of non-static game pieces.

To address these challenges, neural network-based approaches offer greater robustness. Deep learning models, when trained on diverse datasets, can develop representations that are more resistant to variations in lighting and texture. Farisi *et al.* demonstrated that a convolutional neural network (CNN) achieved 94.7% accuracy in cluttered indoor environments, surpassing the performance of traditional feature-based methods [5].

To highlight the differences between feature-based and neural network-based localization methods, Table I compares their robustness, computational cost, accuracy, and adaptability.

C. Real-Time Processing for Embedded Systems

Real-time processing is crucial for effective robot navigation, as solutions that process images after a competition do not provide actionable information during gameplay. The robot

TABLE I
COMPARISON OF FEATURE-BASED AND NEURAL NETWORK-BASED LOCALIZATION

Aspect	**Feature-Based**	**Neural Network-Based**
Robustness	Sensitive to noise [11]	Robust [5]
Computational Cost	Low [12]	High [5]
Accuracy	Limited in low-feature areas [11]	High [14]
Adaptability	Manual tuning [12]	Self-learning [5]

must continuously update its position to adapt to a dynamic environment, requiring efficient visual data processing.

This requirement is challenging due to the limited computational resources of the Botball kit. The Wombat Controller, featuring an STM32F4 microcontroller, is primarily designed for motor control and sensor input, lacking the processing power required for complex image analysis [6]. While the attached Raspberry Pi 3 Model B+ offers improved capabilities, it still falls short compared to modern GPUs and specialized edge computing devices [7].

Previous research indicates that achieving real-time image processing on embedded platforms like the Raspberry Pi 3 is demanding. Deep learning models, such as CNNs, often struggle with real-time inference unless optimized through techniques like quantization or hardware acceleration [8], [13]. Furthermore, frame rate limitations can introduce latency, reducing navigation accuracy [9]. Overcoming these constraints remains a significant challenge in the development of real-time localization systems for embedded robotics platforms.

III. CONCEPT

This paper introduces a vision-based localization system designed for robots competing in the Botball competition. The system uses a USB camera and a lightweight deep learning model to determine the robot's position. The process is divided into two main parts: collecting a dataset and training the model.

A. *Camera Placement*

Placing the camera correctly is crucial for accurate localization. The camera should be mounted securely and consistently to prevent shifts that can lead to errors. Its placement should give a clear view of important visual markers in the environment while avoiding obstacles that might block the view. Proper positioning helps the system detect features more reliably and improve overall accuracy.

B. *Dataset Collection*

To train the model, we need images paired with the robot's exact position. This is done using an ArUco marker system:

- Attach an ArUco marker to the robot so it can be easily tracked.
- Use a stationary camera to follow the marker and record the robot's movements.
- Collect images from the robot's perspective, ensuring each image is labeled with its position and orientation.

It is important to gather a dataset that includes a variety of conditions, like different lighting or backgrounds, to make the model adaptable to real-world environments. Distractions like moving objects can be included to improve the model's robustness, at the cost of increased complexity.

C. *Model Training*

The collected images are used to train a compact convolutional neural network (CNN), which is suitable for running on limited hardware like the Wombat controller. The training process includes:

- Preparing the data by cleaning and normalizing the images to reduce noise.
- Using data augmentation techniques (e.g., adjusting brightness) to make the model more resilient.
- Training the model with optimization methods to improve accuracy.
- Testing the model to ensure it performs well on new, unseen images.

To make the model usable on the Wombat controller, it is compressed using INT8 quantization. This reduces the model's size and computation requirements, allowing it to run in real-time without overloading the hardware.

D. *Concept Conclusions*

This system provides a practical solution for locating robots in real-time during Botball matches using common hardware components. By carefully placing the camera, collecting a well-rounded dataset, and optimizing the model, the robot can reliably determine its position even with the hardware limitations present in the competition.

IV. IMPLEMENTATION

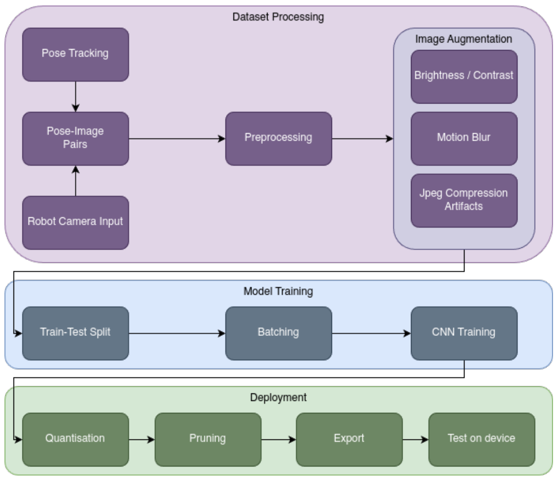

Fig. 1. Overview of the vision-based localization pipeline, illustrating the sequential steps from dataset processing to model training and deployment.

The system architecture consists of three primary stages: **dataset collection**, **model training**, and **deployment** (Figure 1). Data is collected from two cameras: one onboard the robot for environmental perception and another overhead tracking the robot's position via an ArUco marker. Images are synchronized and preprocessed to prepare for model training.

A compact CNN predicts the robot's position (x, y) and orientation (θ) from grayscale images. The network is optimized for deployment on resource-constrained devices, ensuring efficient real-time inference without sacrificing accuracy.

The final stage involves deploying the trained model onto embedded hardware for real-time robot pose estimation.

A. Data Acquisition

Data acquisition ensures high-quality input for pose estimation. The setup employs a Logitech C170 USB camera on the robot and a White Botball camera (2016 model) overhead. The onboard camera captures the robot's perspective (Figure 2), while the overhead camera tracks the ArUco marker for position and orientation.

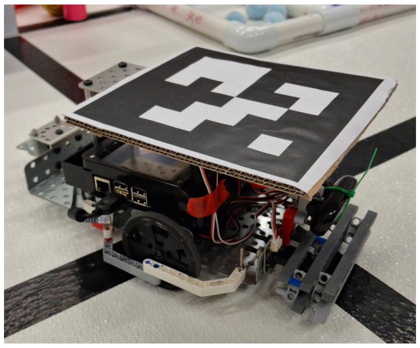

Fig. 2. Robot equipped with a USB camera for environmental perception and an ArUco marker for precise localization. The front-mounted bumper aids in obstacle detection.

To explore the Botball table, the robot follows a cleaning pattern: moving forward until detecting an obstacle, reversing, rotating randomly, and continuing. This approach ensures table coverage without complex path planning but occasionally causes the robot to become stuck due to bumper malfunctions or boundary irregularities.

The overhead camera exhibited skew due to lens distortion and installation misalignment, reducing positional accuracy. A homography transformation using four corner ArUco markers mitigated this issue. Although full calibration was possible, time constraints favored this faster solution.

Images were captured at 160×120 pixels at five frames per second, balancing computational efficiency and detail. Raw data initially showed density variations, with clustered points from navigational behavior and obstacles. To prevent model overfitting, points exceeding a density threshold were removed, reducing the dataset from 12,612 to 9,351 image-pose pairs (Figure 3).

Preprocessing included median-normalizing position values (x, y, θ) to mitigate outliers and converting images to grayscale to reduce complexity. Data augmentation techniques such as brightness variation and motion blur were applied to improve model robustness, a method commonly used in real-world applications [18].

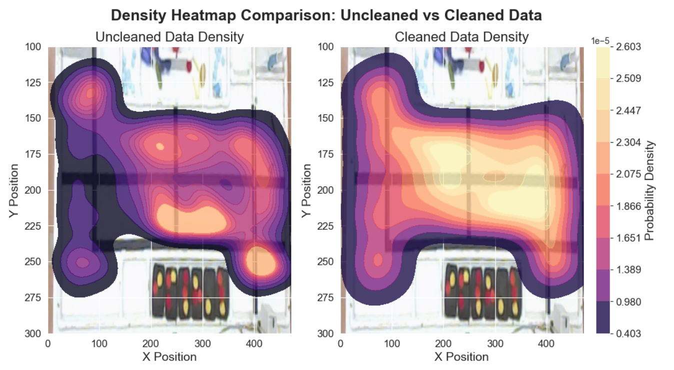

Fig. 3. Density heatmap comparison of localization data before (left) and after cleaning (right). Data balancing reduces overrepresented points, improving model training.

B. Model Training

A CNN is used to predict the robot's pose (x, y, θ) from grayscale images. CNNs are designed to process image data by detecting patterns such as edges, shapes, and textures, making them well-suited for vision-based localization tasks. The dataset was split into training and validation sets. The CNN uses three separable convolutional layers with ReLU activations, max-pooling, and batch normalization, followed by Global Average Pooling (GAP) and a dense regression head (Figure 4). Hyperparameters were selected via a hyperband search.

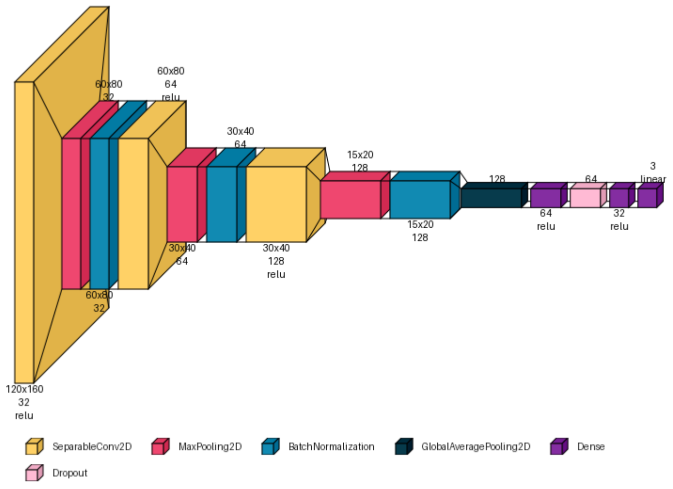

Fig. 4. CNN architecture for pose estimation, featuring separable convolutions, batch normalization, pooling layers, and a dense regression head for predicting (x, y,).

With approximately 22,700 parameters, the model remains under 100 KB, enabling embedded deployment without accuracy loss. Training used the Adam optimizer with an initial learning rate of 0.01, adjusted via a CosineDecayRestarts schedule to improve convergence. Huber loss, chosen for its balance between sensitivity and outlier robustness, outperformed MSE and MAE in regression accuracy.

Experiments were conducted on an NVIDIA GeForce GTX 3060 Ti Laptop GPU using TensorFlow and Keras, with training progress monitored via TensorBoard.

Training ran for up to 1000 epochs with early stopping based on validation loss. Optimal performance was reached at epoch 105, achieving a validation loss of 0.03519. As shown in Figure 5, the model converged within 50 epochs, with the validation loss plateauing thereafter.

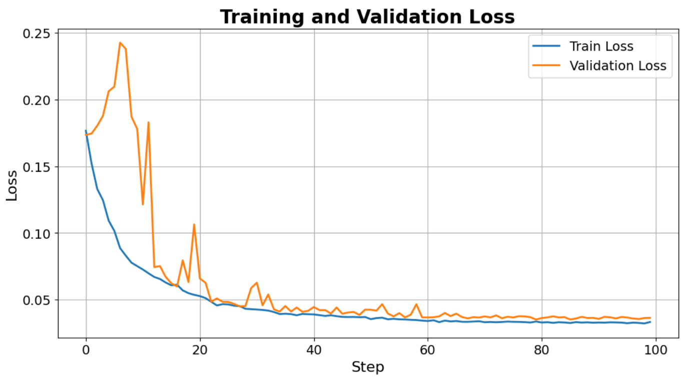

Fig. 5. Training and validation loss over epochs, showing model convergence with steady improvement and minimal overfitting.

V. Results

This section presents the evaluation results of the vision-based localization system. The analysis includes error distribution assessments, kernel density estimation (KDE) visualizations, a comparison between predicted and actual robot positions, and an evaluation of inference results on the Raspberry Pi after quantization. Three figures illustrate the system's localization accuracy and error patterns.

A. Error Distribution Analysis

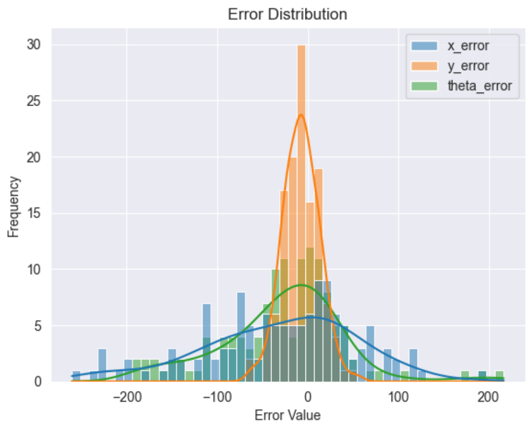

Fig. 6. Error distribution for x, y, and predictions, showing a centered distribution with minimal bias and higher variance in x due to a larger positional range.

Figure 6 shows the histograms and KDE plots of the localization errors in x-position, y-position, and orientation (theta). The distributions are centered around zero, indicating unbiased predictions. However, the x-error shows a wider spread due to the larger range of x-values (0–650) compared to y (0–250) and theta (0–360°). This broader spread reflects the higher variability in horizontal localization. The theta error has a sharp peak, highlighting more precise orientation predictions.

B. KDE of Error Distribution

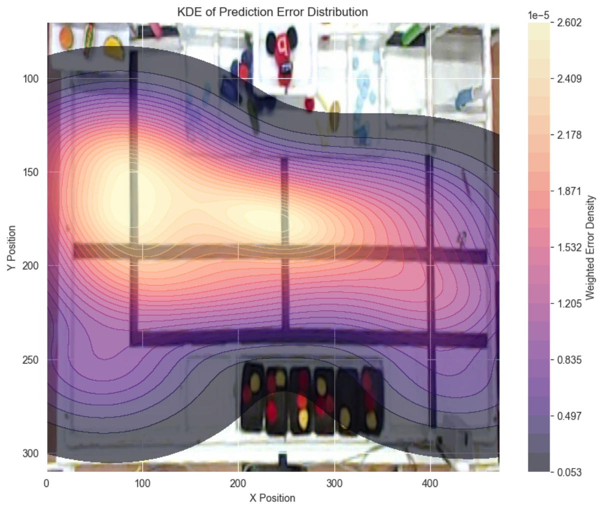

Fig. 7. KDE heatmap of localization errors, highlighting regions with higher prediction inaccuracies, particularly in visually cluttered areas.

Figure 7 displays a KDE heatmap of localization errors overlaid on the Botball table. Warmer colors represent regions with higher error concentrations, most notably in the upper-left area. This pattern likely results from visual clutter or a biased dataset distribution. Conversely, the center and lower-right areas show lower error densities, indicating improved model accuracy in those regions.

C. Predicted vs. Actual Positions

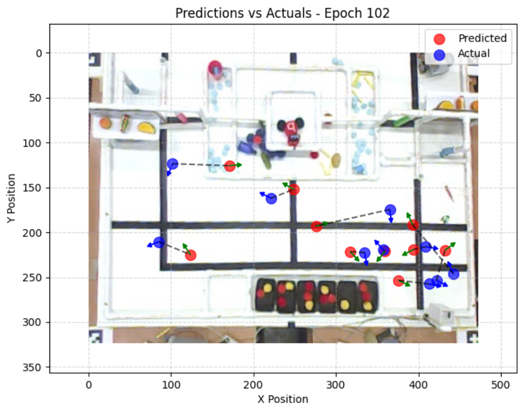

Fig. 8. Comparison of predicted (red) and actual (blue) robot poses, with green arrows indicating orientation. Dashed lines highlight discrepancies, mainly occurring in visually complex areas.

Figure 8 compares predicted and actual robot positions. Red markers show predicted positions, blue markers indicate ground-truth positions, and green arrows represent orientation. Dashed lines connect corresponding predicted and actual points, revealing positional errors. Most predictions closely match actual positions, though larger discrepancies appear in the upper-left quadrant, aligning with the KDE findings. While orientation estimates are often aligned, several instances - particularly in visually complex regions - show deviations exceeding 90°, highlighting areas for improvement. Future work should address discontinuities near the 0°/360° boundary by using a custom loss function that accounts for angular periodicity.

D. Inference Results on Raspberry Pi

To deploy the trained model on the pi, the model has to be quantized, a technique that reduces the precision of model parameters (e.g., converting 32-bit floating-point numbers to 8-bit integers). This significantly decreases memory usage and improves inference speed while introducing a slight accuracy trade-off. To assess the feasibility of deploying the model on an embedded system, inference tests were conducted on a Raspberry Pi using an INT8 quantized version of the model. For quantization, tensorflow lite has been used. Quantization reduced the model size and improved inference speed, albeit with a slight reduction in prediction accuracy.

The original, full-precision model achieved a Root Mean Squared Error (RMSE) of 64.81. After applying INT8 quantization, the RMSE increased to 74.31, an increment by 14.6%, indicating a modest drop in accuracy. However, the quantized model significantly improved inference speed, averaging just 0.0773 seconds per inference on the Raspberry Pi. Given this speed, it is possible to achieve a real-time performance with around 10 frames per second.

These results highlight the trade-off between accuracy and speed. While quantization introduces some performance degradation, the reduced inference time is crucial for real-time applications, making the INT8 quantized model a practical solution for embedded deployment scenarios where computational resources are limited.

VI. Conclusions and Further Work

This work establishes a baseline for real-time vision-based robot localization using a lightweight CNN on resource-constrained hardware. The system achieves real-time pose estimation with an inference time of 0.0773 seconds per frame on a Raspberry Pi 3 Model B+, demonstrating that optimized neural networks can provide reliable localization despite limited computational resources. The results confirm the practicality of deep learning-based localization in competitive robotics environments like Botball.

Despite these achievements, accuracy improvements remain necessary for full competition readiness, particularly in orientation estimation and edge-case angle scenarios. Two promising directions for future work include sensor fusion and temporal models.

A. Sensor Fusion

Combining vision with sensors like gyroscopes improves robustness. Providing the robot's orientation (θ) and focusing on position (x, y) simplifies learning. Training the model to predict uncertainty allows better integration with filters like the Kalman filter, enhancing accuracy and handling visual occlusions.

B. Temporal Models

Using sequential images improves motion consistency. Lightweight models, such as ConvLSTMs or temporal transformers, capture movement patterns efficiently. Extracting features from consecutive frames reduces computation, while integrating temporal data with SLAM yields smoother localization.

References

[1] Scaramuzza, D., Fraundorfer, F. (2011). Visual Odometry [Tutorial]. *IEEE Robot. Automat. Mag.*, 18, 80-92. https://doi.org/10.1109/MRA.2011.943233.

[2] Bisio, I., et al. (2015). Detecting if a Smartphone is Indoors or Outdoors with Ultrasounds. Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. And Serv.. https://doi.org/10.1145/2742647.2745907.

[3] Klauninger, B., et al. (2024). Enhancement of Accuracy in Botball Navigation. HTBLVA Wiener Neustadt, Austria. [Online]. Available: https://www.kipr.org/wp-content/uploads/2024/08/Enhancement_of_Accuracy_in_Botball_Navigation.pdf. [Accessed: Feb. 10, 2025].

[4] Lu, Z., et al. (2022). Vision-based Localization Methods under GPS-Denied Conditions. arXiv preprint arXiv:2211.11988. [Online]. Available: https://arxiv.org/abs/2211.11988. [Accessed: Feb. 10, 2025].

[5] Farisi, Z., et al. (2019). Vision-Based Indoor Localization via CNN. Int. J. Adv. Comput. Sci. Appl.. https://doi.org/10.14569/ijacsa.2019.0100709.

[6] STMicroelectronics. (2022). STM32F4 Series: High-Performance ARM Cortex-M4 MCUs. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html.

[7] Raspberry Pi Foundation. (2018). Raspberry Pi 3 Model B+ Technical Specifications. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[8] Smith, T., et al. (2021). Optimizing CNN Inference on Embedded Systems. IEEE Trans. Embedded Comput., 12(3), 45-59.

[9] Lu, Z., et al. (2022). Real-Time Vision Processing Constraints on Embedded Systems. arXiv preprint arXiv:2211.11988. [Online]. Available: https://arxiv.org/abs/2211.11988.

[10] Wang, J., et al. (2018). A Survey of Corner Detection Methods.

[11] Huang, Q., et al. (2024). A Survey of Feature Matching Methods. IET Image Process., 18, 1385–1410. https://doi.org/10.1049/ipr2.13032.

[12] Azzam, R., et al. (2020). Feature-Based Visual Simultaneous Localization and Mapping: A Survey. SN Appl. Sci., 2, 224. https://doi.org/10.1007/s42452-020-2001-3.

[13] Liu, X. (2021). Model Optimization Techniques for Embedded AI. 2021 2nd Int. Conf. Comput. Data Sci. CDS), 1-6. https://doi.org/10.1109/CDS52072.2021.00008.

[14] Liu, H., et al. (2021). Visual Odometry Algorithm Based on Deep Learning. 2021 6th Int. Conf. Image, Vision, Comput. ICIVC), 322-327. https://doi.org/10.1109/ICIVC52351.2021.9526975.

[15] Howard, A. G., et al. (2017). MobileNets: Efficient CNNs for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: Feb. 10, 2025].

[16] Hugging Face. (2023). Quantization - Optimum. [Online]. Available: https://huggingface.co/docs/optimum/en/concept_guides/quantization. [Accessed: Feb. 10, 2025].

[17] Gavrikov, P. (2020). Visualkeras. GitHub repository. Available at: https://github.com/paulgavrikov/visualkeras.

[18] Viso Suite, "Image Data Augmentation for Computer Vision," Viso.ai, 2023. [Online]. Available: https://viso.ai/computer-vision/image-data-augmentation-for-computer-vision/. [Accessed: Mar. 24, 2025].