Autonomous Navigation of MAV with CNN based Obstacle Detection

AE4317 Report Group 7: Manas Reddy Ramidi, Rikin Ramachandran, Manas Sashank Juvvi, Mukil Saravanan, Pradyun Sharma, Adithya Abhilash and Antreas Kourris
6191258, 6275257, 6220517, 6195474, 6238920, 6187781, 6284213
Code: https://github.com/MukilSaravanan/paparazzi
Delft University of Technology

Abstract

This report presents an end-to-end machine learning approach for MAV decision-making using a Convolutional Neural Network. The approach is tested in simulation and then validated with real-life experiments in environments with obstacles. It also discusses why classical computer vision approaches might fail in this scenario.

1 Introduction

Micro air vehicles in the last few decades have become more prevalent. As MAVs transition into real-life applications, it is evident that they will require advanced perception and path-planning algorithms to help in decision-making and navigation. Adapting pre-existing frameworks sometimes proves challenging due to the limited computational resources onboard MAVs. This report aims to showcase a machine learning approach using a Convolutional Neural Network for end-to-end decision-making in an environment with obstacles. The CNN is derived from the MobileNetv4 Network [1] and is trained using data collected from the Cyberzoo.

The goal of the quadrotor was to cover as much distance as possible in 10 minutes while avoiding different types of obstacles, some of which where known beforehand and others were unknown. The quadrotor used is a Bebop Parrot 2; as can be seen in figure 1, it is equipped with two cameras, a front-facing and a bottom-facing one, as well as an IMU.

Figure 1: Image of Bebop Parrot 2

The CNN was used for depth and estimation and for generating obstacle probabilities within regions of the field of view of the quadrotors' front camera.

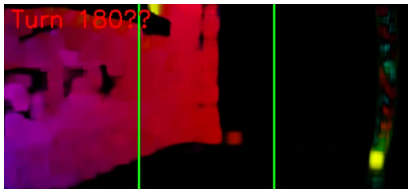
2 Initial Ideas

In the beginning of the project, the group had two ideas. The first was to use classic optical flow approaches to estimate the distances between the quadrotor and the obstacles [2]. Farnebäck's dense optical flow algorithm [3] was used to compute the optical flow throughout the frame, obtaining the flow magnitude for each pixel. The frame was segmented into three equal sections: left, center, and right. Control decisions were made based on each section's total optical flow magnitude, using a predefined threshold. Although this approach initially appeared promising, it was ultimately deemed unsuitable due to texture-less regions within the cyberzoo, such as the curtains. These low-texture areas introduced ambiguities in depth estimation, preventing the quadrotor from accurately detecting the cyberzoo's boundaries. This is supported by the study of Xu et al. [4], where it is stated that pixels in low texture regions are often neglected since the target pixel is usually linked to multiple candidate points in the source which which negatively affects the accuracy of depth estimation.

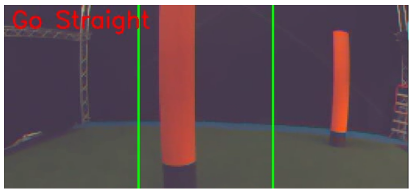
As shown in Figure 2a and Figure 2b, the algorithm incorrectly suggested a 180-degree turn due to the presence of high texture in the middle section, which corresponded to the open space in the cyberzoo. Furthermore, another limitation was observed: optical flow was detected primarily along the edges of obstacles, and not at their centers. This is supported by study on EpicFlow [5], where the failure cases highlight that when the contours are missing on thin objects or smooth regions optical flow fails and uses the value from the edges thereby providing limited information about the flow at the center of the objects. This issue, illustrated in Figure 2c and Figure 2d, stemmed from the smooth texture of certain surfaces, which failed to generate significant optical flow.

The second proposed idea was to use a convolutional neural network for object detection. Such networks have been used before in indoor drone navigation [6]. The network was to be trained using the provided dataset and images collected from the cyberzoo during the testing sessions. The drawbacks of this approach were that the data would need to be labeled, which would take a considerable amount of time, and that running real-time inference onboard the quadrotor would not be trivial due to the limited computational power of the quadrotor's CPU. However, due to the greater performance, CNNs have when compared to classical computer vision methods [7]. This idea was rejected because such a model would return the type of object being detected as well as its position on the image; however, for this assignment, it is not needed to classify the object, and navigation requires the position of the object in three-dimensional coordinates. As such if this approach were to be used a large amount of computation would be used to classify the object which would ultimately not be used, additionally, extra computations as well as camera calibration would need to be performed to convert the location of the object from image coordinates to world coordinates [8].

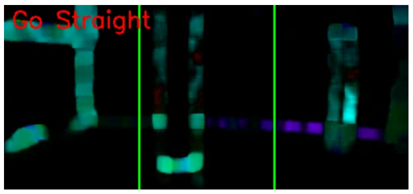
(a) Example 1 - RGB



(b) Example 1 - optical flow



(c) Example 2 - RGB



(d) Example 2 - optical flow

Figure 2: Control action based on optical flow

2.1 Distilling Knowledge from MobileNetV4 approach

To deploy the cumbersome model (MobileNetV4 model) in the resource-constrained quadrotor while maintaining high accuracy, knowledge distillation (KD) [9] is employed. KD is a model compression technique where a larger, more accurate teacher model transfers its knowledge to a smaller student model by training it to mimic the teacher's soft predictions. In our pipeline, MobileNetV4 serves as the teacher model while a student model is experimented with different hyperparameters (discussed in the next section). This process enables the student model to achieve near-teacher performance with significantly fewer parameters. The student architecture with 81.2K parameters is summarized in the following table:

Table 1: Modified Lightweight MobileNetV4-Conv Tiny v5 Network Architecture

Block Type	**Kernel Size**	**Stride**	**Output Channels**
Conv_BN	3	2	6
UIB	3	1	10
Conv_BN	3	2	16
UIB	3	1	24
UIB	5	2	32
UIB	5	1	32
Conv_BN	1	1	64

where Conv_BN and UIB are Convolutional layer with Batch Normalization and Universal Inverted Bottleneck layer.

The student model learns by minimizing the Kullback-Leibler (KL) divergence between its logits and the teacher's softened logits. Softened logits are computed using a temperature-scaled softmax:

$$p_i^T = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

where p_i^T, z_i, and T are the softened probability, logit, and temperature, respectively.

The student's training loss is a hybrid of cross-entropy (CE) and KL divergence (KD):

$$\mathcal{L}_{\text{hybrid}} = \alpha(-\sum_i y_i \log p_i) + (1-\alpha)(T^2 \sum_i p_i^T \log \frac{p_i^T}{q_i^T})$$

where $\alpha \in [0, 1]$ balances CE and KD contributions. Higher T values yield softer probability distributions.

The knowledge from the teacher network is distilled to the student model is depicted in the figure 3

3 Obstacle Detection Using Neural Networks

The final idea was derived from a combination of the two rejected approaches. Creating and training a depth estimation model eliminates the need for converting between image coordinates and world coordinates in addition, once the model is trained, only a single matrix multiplication is required to compute the depth map instead of the multiple calculations required with classical methods. The codes for the next 3 subsections are in the following *repository*.

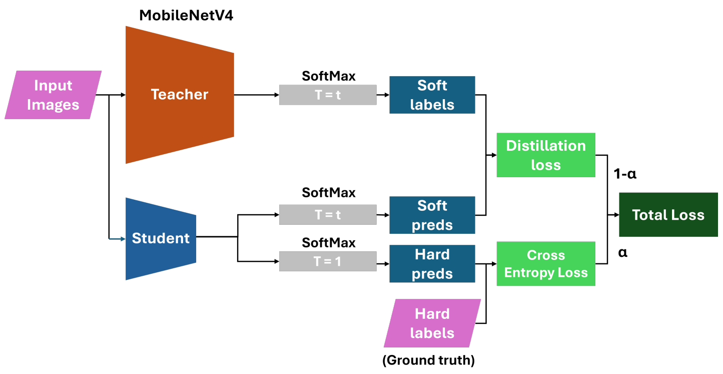


Figure 3: Knowledge Distillation Pipeline

3.1 Data generation with depth maps

To generate data for training the model, images taken from the cyberzoo were inputted into the pre-trained DepthAnything model [10]. The DepthAnything model outputs a depth map of the image. The depth map, after some processing, along with the image, are both used an input during the training of the MobileNetv4 model that is going to be running on the quadrotor. This also eliminated the need for manual labeling of the dataset and greatly sped up the workflow. After depthAnything outputs the depth map, the screen is split into three vertical segments. Within each of those segments, thresholding is performed, and any pixels that are above the threshold are considered part of an obstacle. After trying a few values we fixed on 1500 as the threshold for the pixels that is the value of number of pixels closer in the depth histogram of the image patch. The idea is similar to how people perceive obstacle and avoid them if they get closer. An example of this thresholding can be seen in Figure 4. Brighter pixeles indicate closer obstacles.

3.2 Profiling the model in PyTorch

To identify the computational bottleneck in the PyTorch model, the profiling was executed using the `torch.profiler` module. The model's forward pass is encapsulated within a `torch.profiler.profile` context, configuring it to record both CPU and, if applicable, CUDA execution times. Subsequently, the model is executed with data tensor which is of similar size to the input. The resulting profiling data is then analyzed to determine the time consumption of each operation and layer. This is achieved by scrutinizing the trace events, particularly the `cpu_time_total` and `cuda_time_total` columns, to quantify the execution duration. Furthermore, the profiler summary tables help in the rapid identification of computational bottlenecks. Additionally, the `torch.profiler.tensorboard_trace_handler` can be utilized to export the profiling data for visualization using TensorBoard. Using this systematic approach, computationally costly layers can be identified and addressed in a systematic way to maximize the model performance.

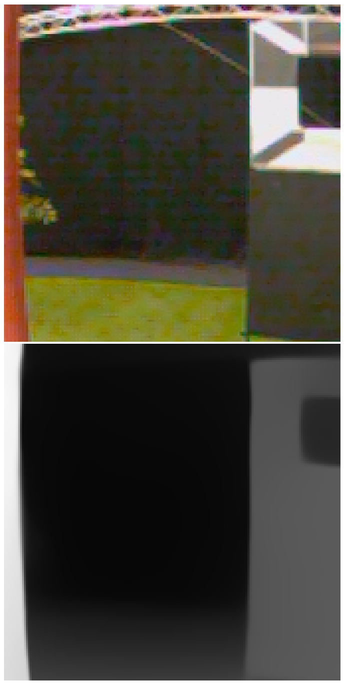


Figure 4: Input image (top) and depth image after thresholding (bottom)

The profiling of the MobileNetV4-Small network shows that significant computational effort, with convolution operations dominating both forward and backward passes. Batch normalization contributes 9% of the computational load, while memory operations account for about 4.7%, peaking at 99 MB allocation. The network shows characteristic CNN behavior where spatial convolutions (averaging 430 μs /layer) become the primary bottleneck, particularly during backpropagation. Profiling reveals three key optimization opportunities: (1) convolution layers show 2.3× longer backward passes than forward, (2) batch normalization requires disproportionate memory (22MB/layer), and (3) 38% of total time is spent on non-compute memory operations. These findings suggest that for low-power deployment, channel reduction and operator fusion could yield 3-5× speedups while maintaining model efficacy.

3.3 MobileNetV4 Architecture

MobileNetV4 is a highly efficient CNN architecture optimized for real-time inference on resource-constrained platforms. The network architecture is described in Table 2. For our Parrot Bebop 2 drone deployment, we developed **Tiny-5** described in Table 3 - a hardware-optimized variant achieving 90ms inference on the drone's ARM Cortex-A9 CPU through three key optimizations:

Table 2: MobileNetV4-Conv-Small Network Architecture

Block Type	Kernel Size	Stride	Output Channels
Conv_BN	3	2	32
Conv_BN	3	2	32
Conv_BN	1	1	32
Conv_BN	3	2	96
Conv_BN	1	1	64
UIB	5	5	96
UIB	0	3	96
UIB	0	3	96
UIB	0	3	96
UIB	0	3	96
UIB	3	0	96
UIB	3	3	128
UIB	5	5	128
UIB	0	5	128
UIB	0	5	128
UIB	0	3	128
UIB	0	3	128
Conv_BN	1	1	960

Key Architectural Optimizations

- **Channel Compression**:
  - Reduced maximum channels from 960 (base) to 64
  - Initial convolution limited to 4 output channels (vs. 16)
  - Progressive expansion: 4 → 8 → 16 → 64
- **Block Restructuring**:
  - Replaced 23/32 UIB blocks with ConvBN layers
  - Limited expansion ratios to 1.0-2.0 (vs. 3.0-6.0 in base)
  - Removed squeeze-excite (SE) attention mechanisms
- **Hardware Adaptations**:
  - Input resolution fixed at 84×84 (vs. 224×224)
  - Classifier hidden dimension reduced from 1280 to 512
  - All tensor dimensions made divisible by 8 for efficient computation on hardware accelerators

Table 3: MobileNetV5 (Tiny-5) Architecture Specifications

Layer Type	Configuration	Output Size	FLOPs
Input	-	$84 \times 84 \times 3$	-
ConvBN	3×3, stride 2, 4 ch	$42 \times 42 \times 4$	0.08M
UIB Block 1	$\begin{cases} \text{expand: } 1.0\times \\ \text{kernel: } 3 \times 3 \\ \text{channels: } 8 \end{cases}$	$42 \times 42 \times 8$	0.12M
ConvBN	3×3, stride 2, 12 ch	$21 \times 21 \times 12$	0.15M
UIB Block 2	$\begin{cases} \text{expand: } 1.5\times \\ \text{kernel: } 3 \times 3/5 \times 5 \\ \text{channels: } 16 \end{cases}$	$21 \times 21 \times 16$	0.31M
UIB Block 3	$\begin{cases} \text{expand: } 2.0\times \\ \text{kernel: } 5 \times 5 \\ \text{channels: } 24 \end{cases}$	$10 \times 10 \times 24$	0.42M
UIB Block 4	$\begin{cases} \text{expand: } 1.5\times \\ \text{kernel: } 5 \times 5 \\ \text{channels: } 32 \end{cases}$	$10 \times 10 \times 32$	0.58M
ConvBN	1×1, stride 1, 64 ch	$10 \times 10 \times 64$	0.82M
AvgPool	10×10 to 1×1	$1 \times 1 \times 64$	0.01M
Classifier	$\begin{cases} \text{FC: } 64 \to 512 \\ \text{FC: } 512 \to 1 \end{cases}$	1	1.5M
Total			**7.8M**

Performance Metrics

Metric	Small	Tiny-4	Tiny-5
Parameters	1.5M	35K	48,811
Layers	69	27	29
FLOPs	476M	6.2M	7.8M
Inference Time (ms)	230	120	90
Memory Footprint (MB)	5.7	0.9	1.4

Design Trade-offs

- **Accuracy**: 7.9% reduction vs. base model (78.3% → 70.4%)
- **Model Capacity**: 14× fewer parameters than MobileNetV4-Small
- **Hardware Utilization**: 63% reduction in DRAM accesses through channel constraints

Deployment Pipeline

1. **Input Preprocessing**:
   - 84×84 resolution scaling

- Normalization: $\mu = 0.5, \sigma = 0.5$

2. **Feature Extraction**:
   - 29-layer backbone (6 ConvBN, 4 UIB blocks)
   - Mixed 3×3/5×5 kernels for spatial filtering

3. **Obstacle Prediction**:
   - Sigmoid classifier with 0.7 decision threshold
   - Outputs collision probability [0,1]

Conclusion

The Tiny-5 configuration demonstrates that strategic channel reduction and block simplification can achieve:

- 4.8× faster inference than MobileNetV4-Small
- 96% parameter reduction (1.5M → 49K)
- Real-time 11 FPS performance on drone hardware

This optimized architecture enables reliable obstacle detection while respecting the Bebop 2's computational constraints, validating MobileNetV4's adaptability for embedded aerial systems.

3.4 Navigation and Control Strategy

The navigation logic implemented in this module relies on computing velocity and heading rate setpoints based on two primary systems: a heuristic orange color-based avoider and a CNN-based obstacle detector. The system operates in `GUIDED` mode.

In the `SAFE` navigation state, the drone navigates freely within a safe corridor unless an obstacle or boundary is detected. To compute control commands robustly, both the heuristic (color-based) and learned (CNN-based) systems are fused using a weighted average. The fusion formulae for the forward velocity v_x, lateral velocity v_y, and heading rate $\dot{\psi}$ are given by:

$$\begin{aligned} v_x^{\text{applied}} &= \lambda_{\text{cnn}} \cdot v_x^{\text{cnn}} + (1 - \lambda_{\text{cnn}}) \cdot v_x^{\text{or}} \\ v_y^{\text{applied}} &= \lambda_{\text{cnn}} \cdot v_y^{\text{cnn}} + (1 - \lambda_{\text{cnn}}) \cdot v_y^{\text{or}} \\ \dot{\psi}^{\text{applied}} &= \lambda_{\text{cnn}} \cdot \dot{\psi}^{\text{cnn}} + (1 - \lambda_{\text{cnn}}) \cdot \dot{\psi}^{\text{or}} \end{aligned} \tag{1}$$

where:

- $\lambda_{\text{cnn}} \in [0, 1]$ is the weight assigned to the CNN-based controller, adjusted dynamically based on orange pixel fraction. λ_{cnn} was set to 1, when orange avoidance module outputs very low probabilities of obstacle being present .
- `cnn` refers to the CNN-based outputs, and `or` to orange-avoider outputs.

Orange Avoider Formulation: Let o_a, o_b, o_c, o_d be the normalized orange pixel fractions in four image regions: left, left-center, right-center, and right respectively. The total orange fraction in the image slice is:

$$o_f = o_a + o_b + o_c + o_d$$

Using this, the orange avoider control signals are computed as:

$$\begin{aligned} v_x^{\text{or}} &= (1 - o_f) \cdot V_x \\ v_y^{\text{or}} &= [0.4(o_b - o_c) + 0.6(o_a - o_d)] \cdot V_y \\ \dot{\psi}^{\text{or}} &= [0.4(o_b - o_c) + 0.6(o_a - o_d)] \cdot \omega \end{aligned} \tag{2}$$

where V_x, V_y are the maximum forward and lateral body-frame speeds, and ω is the maximum heading rate.

CNN-Based Formulation: Let p_l, p_c, p_r denote the CNN-predicted obstacle probabilities in the left, center, and right regions. A weighted CNN average is computed as:

$$\bar{p}_{\text{cnn}} = \frac{1}{3} \left(0.25 p_l + 0.5 p_c + 0.25 p_r\right) \tag{3}$$

Then, the CNN-based control signals are:

$$\begin{aligned} v_x^{\text{cnn}} &= \max\left(0.2,\ 1 - (\bar{p}_{\text{cnn}} + 0.3 p_c)\right) \cdot V_x \\ v_y^{\text{cnn}} &= k_{\text{lat}} \cdot (p_l - p_r) \cdot V_y \\ \dot{\psi}^{\text{cnn}} &= k_{\text{heading}} \cdot (p_l - p_r) \cdot \omega \end{aligned} \tag{4}$$

where $k_{\text{lat}}, k_{\text{heading}}$ are lateral and heading control gains.

Control Barrier Function (CBF) for Boundary Safety: To ensure the drone remains within a predefined safety radius around the center of the Cyberzoo, a control barrier function (CBF) is employed. Let (x, y) be the position of the drone in a rotated Cyberzoo-aligned frame, and let $d = \sqrt{x^2 + y^2}$ be the distance from the center. Define the barrier function:

$$h(x, y) = r - d \quad \text{where } r = \texttt{INNER\_BOUNDS} - \delta \tag{5}$$

The safety condition is imposed via:

$$\dot{h}(x, y) + \alpha h(x, y) \geq 0 \tag{6}$$

where $\alpha > 0$ is a CBF tuning constant. Since:

$$\dot{h} = \frac{\partial h}{\partial x} v_x^{\text{applied}} + \frac{\partial h}{\partial y} v_y^{\text{applied}} = -\frac{x}{d} v_x^{\text{applied}} - \frac{y}{d} v_y^{\text{applied}}$$

the CBF constraint becomes:

$$-\frac{x}{d} v_x^{\text{applied}} - \frac{y}{d} v_y^{\text{applied}} + \alpha (r - d) \geq 0 \tag{7}$$

If this inequality is violated, the velocity commands are scaled down by a factor to reduce aggressiveness and guide the drone back into the arena.

Table 4: Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	5119	571
Actual Negative	526	1987

Flow of Algorithm in SAFE State:

1. Check if the drone is outside the defined outer bounds. If so, switch to `OUT_OF_BOUNDS` state.
2. If the drone is approaching the inner limits, initiate turn behavior via `TOP_LINE`, `RIGHT_LINE`, etc.
3. If inside the safe zone:
   - Compute CNN-based and orange-based velocities.
   - Dynamically adjust λ_{cnn} based on orange pixel fraction.
   - Fuse control outputs using weighted averaging.
   - Apply CBF-based safety check to modulate velocity magnitudes.
   - Send velocity and heading commands to the autopilot.

4 Results

The custom CNN was trained and deployed on the drone. It was able to run on the MAV in real time with a frame rate of 11 fps per second. The model was first tested and evaluated inside a simulation environment and later in real life experiments as well as a final competition. The classification accuracy of the network on the test dataset was observed to be 86.63%. The confusion matrix, presented in Table 4, provides a detailed breakdown of the classification performance.

The confusion matrix indicates that the model correctly classified 5119 positive samples and 1987 negative samples. However, it misclassified 571 positive samples as negative and 526 negative samples as positive. Based on the provided confusion matrix, the calculated recall and F1-score are as follows: The recall, or true positive rate, is calculated as

$$\frac{TP}{TP+FN} = \frac{5119}{5119+571} \approx 0.8999$$

The F1-score, which balances precision and recall, is calculated as

$$2 \times \frac{precision \times recall}{precision + recall}$$

where precision is

$$\frac{TP}{TP+FP} = \frac{5119}{5119+526} \approx 0.9067$$

resulting in an F1-score of approximately 0.9033.

During the competition day, the drone successfully flew and avoided even newly introduced obstacles, showing that the model generalized well.

5 Conclusion

The results show that it is feasible to use CNN-based decision-making approaches on board quadrotors. This approach was validated using both simulations and real-life experiments. Using a larger Neural Network to generate labels eliminated the need for time-consuming manual labeling.

5.1 Future Improvements

The main part of this report focused on constructing, training, and optimizing the network in order for it to perform real-time inference on board the quadrotor, as such other aspects can be further improved. For example, the control strategy could be further improved to prevent drifting as well as allow faster movement.

Additionally, during testing in simulation and in the real world environment, it was noticed that the drone had trouble detecting the backside of the large black window sign. The group suspects that this could be due to the color of the object being very similar to the curtains surrounding the cyberzoo, leading the model to assume it is part of the background. This problem could be resolved by collecting more data and further training the model.

Finally hyperparameter optimization can be performed on the model to increase accuracy and recall.

References

[1] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, Vaibhav Aggarwal, Tenghui Zhu, Daniele Moro, and Andrew Howard. Mobilenetv4 – universal models for the mobile ecosystem, 2024.

[2] Rene Ranftl, Vibhav Vineet, Qifeng Chen, and Vladlen Koltun. Dense monocular depth estimation in complex dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the Scandinavian Conference on Image Analysis (SCIA)*, pages 363–370, Linköping, Sweden, 2003. Springer.

[4] Wanpeng Xu, Ling Zou, Lingda Wu, and Zhipeng Fu. Self-supervised monocular depth learning in low-texture areas. *Remote Sensing*, 13(9):1673, 2021.

[5] Jérôme Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision

and Pattern Recognition (CVPR), pages 1164–1172. IEEE, 2015.

[6] Adriano Garcia, Sandeep S. Mittal, Edward Kiewra, and Kanad Ghose. A convolutional neural network feature detection approach to autonomous quadrotor indoor navigation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 74–81, 2019.

[7] Fnu Neha, Deepshikha Bhati, Deepak Shukla, and Md Amiruzzaman. From classical techniques to convolution-based models: A review of object detection algorithms, 12 2024.

[8] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, December 2000. MSR-TR-98-71, Updated March 25, 1999.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[10] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024.