

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

BACHELOR THESIS IN COMPUTER ENGINEERING

Multi-View CNNs for Industrial Object Classification: From Synthetic Dataset Design to Transfer Learning and Fusion Strategies

CANDIDATE

Gianmaria Frigo

Student ID 2066598

SUPERVISOR

Prof. Stefano Ghidoni

University of Padova

CO-SUPERVISOR

Fabio Scarpa

M31 S.r.l.

ACADEMIC YEAR
2024/2025

*To those who
stood beside me
throughout.*

Abstract

Industrial assembly lines increasingly rely on automated vision systems to sort thousands of visually similar components. However, collecting large, labeled, multi-view image sets for every part is impractical, since parts often arrive directly from suppliers without prior access for training. This thesis investigates a synthetic-to-real pipeline for multi-view convolutional neural networks (MVCNNs). This approach enables classifiers to be trained from CAD models and then applied to real images in a five-camera imaging box.

We developed a Blender-based renderer to generate a synthetic dataset of 80 parts with randomized poses, materials, lighting, and optics, which provides diverse five-view samples for training. Using this dataset, we evaluate transfer learning with ImageNet-pretrained backbones, freezing strategies, fusion mechanisms, weight sharing, and several backbone families. Freezing the first three ResNet-50 stages matches the accuracy of full fine-tuning while improving stability. Among fusion mechanisms, score-sum and deep early fusion achieve the most reliable transfer to real data. Full weight sharing across view branches improves robustness while reducing parameters. Backbone comparison shows that compact modern CNNs, such as ConvNeXt-Small, generalize best.

Overall, the results demonstrate that synthetic training combined with judicious transfer learning, deep fusion, and full weight sharing yields near-perfect real-world accuracy with modest model footprints. However, the study is limited to 80 synthetic classes and a real sample evaluation set with a single physical object, while the target application involves up to 10,000 categories. The findings should therefore be regarded as preliminary, establishing a baseline and outlining a scalable route toward industrial deployment with reduced data-collection overhead.

Sommario

Le linee di montaggio industriali si affidano sempre più spesso a sistemi di visione automatica per smistare migliaia di componenti tra loro simili. Raccogliere grandi insiemi di immagini multi-vista etichettate è spesso impraticabile, poiché i componenti arrivano dai fornitori senza che vi sia la possibilità di utilizzarli in fase di addestramento. Questa tesi esplora una pipeline *synthetic-to-real* per reti neurali convoluzionali multi-vista (MVCNN), addestrando i classificatori sui modelli CAD e applicandoli poi alle immagini reali acquisite da un box industriale dotato di cinque telecamere.

Abbiamo sviluppato in Blender un renderer con cui è stato generato un dataset sintetico di 80 componenti con pose, materiali, illuminazione e parametri ottici randomizzati, producendo campioni a cinque viste per l'addestramento. Con questo dataset valutiamo il transfer learning con backbone pre-addestrati su ImageNet, strategie di congelamento dei layer, meccanismi di fusione, condivisione dei pesi e diverse famiglie di backbone. Congelare i primi tre stadi di ResNet-50 eguaglia l'accuratezza del fine-tuning completo, con maggiore stabilità. Tra i metodi di fusione, la somma dei punteggi (score-sum) e la fusione a livello di feature (early fusion) in stadi profondi trasferiscono meglio ai dati reali. La condivisione totale dei pesi tra le cinque viste ha migliorato la robustezza riducendo i parametri. Il confronto tra backbone indica come le CNN moderne e compatte, come ConvNeXt-Small, generalizzano meglio.

Complessivamente, l'addestramento su dati sintetici, combinato con transfer learning mirato, fusione profonda e condivisione dei pesi completa, permette di raggiungere un'accuratezza quasi perfetta su immagini reali con modelli dalle dimensioni contenute. Rimangono però alcuni limiti: il dataset sintetico comprende solo 80 classi e la valutazione su sample reali è stata condotta su un unico oggetto fisico, mentre l'applicazione finale prevede fino a 10.000 categorie. I risultati sono quindi preliminari, ma forniscono una base e delineano un percorso scalabile verso l'impiego industriale, riducendo i costi di acquisizione dei dati.

Contents

List of Figures	xi
List of Tables	xiii
List of Acronyms	xix
1 Introduction	1
1.1 Context and Motivation	1
1.2 Problem Setting	2
1.3 Research Objectives	2
1.4 Contributions	3
1.5 Thesis Structure	3
2 Related Work	4
2.1 Convolutional Neural Networks	4
2.2 Multi-View Recognition	5
2.3 Synthetic-to-Real Transfer	7
2.4 Transfer Learning and Pre-Training	8
3 Methods	9
3.1 Synthetic Dataset Generation	9
3.1.1 Imaging Box and Camera Layout	9
3.1.2 Domain Randomization	11
3.1.3 Dataset Composition	12
3.2 Model Architectures and Design Choices	14
3.2.1 Backbone Encoders and Input Handling	14
3.2.2 Fusion Strategies	15
3.2.3 Weight-Sharing Schemes	16
3.2.4 Freezing Policies	17

3.2.5	Classifier Head	17
3.3	Evaluation Protocol	18
3.4	Ablation Studies	19
4	Results	23
4.1	Reporting Conventions	23
4.2	Freezing Policies	24
4.3	Fusion Strategies	25
4.4	Weight-Sharing Schemes	26
4.5	Backbone Families	27
4.6	Label Smoothing	28
4.7	Temperature Scaling	30
4.8	Consistency Across Seeds	32
5	Discussion	34
5.1	Freezing Policies and Transferability	34
5.2	Fusion Strategies	35
5.3	Weight Sharing Schemes	36
5.4	Backbone Families	37
5.5	Calibration	38
5.6	Robustness & Reliability	39
5.7	Limitations	39
5.8	Future Work	40
6	Conclusions	41
6.1	Contributions	41
6.2	Outlook	42
	References	43
	Acknowledgments	46

List of Figures

2.1	Typical architecture of a CNN. <i>Reproduced from</i> Alzubaidi et al. [2], licensed under CC BY 4.0.	5
2.2	MVCNN pipeline: multiple views are processed by a shared backbone, fused, and classified. <i>Reproduced from</i> Alzahrani et al. [1], licensed under CC BY 4.0.	6
2.3	Top: early, late, and score fusion strategies. Bottom: fusion strategies mapped onto the ResNet-50 architecture. <i>Adapted from</i> Seeland and Mäder [18], licensed under CC BY 4.0.	7
3.1	Imaging box used for testing.	10
3.2	Domain randomization examples: the same object (used in the real captures), <i>back_left</i> view, different randomized poses and render settings (five samples).	12
3.3	Three example synthetic samples showing the five views produced by the pipeline; top row: sample of the 4th CAD object provided by the client; middle and bottom rows: samples from the selected Thing10K set.	13
3.4	Sample five-view capture from the <i>Good Conditions</i> subset.	18
3.5	Sample five-view capture from the <i>Edge cases</i> subset.	18
3.6	Sample five-view capture from the <i>Objects interference</i> subset.	19
3.7	Sample five-view capture from the <i>View occlusion</i> subset.	19
4.1	Validation accuracy evolution during training for the different freezing policies.	24
4.2	Validation accuracy evolution for representative fusion strategies over 10 epochs.	26
4.3	Validation accuracy over 10 epochs for the three weight-sharing policies.	27

4.4	Validation accuracy over 10 epochs for backbone families under <i>score-sum</i> fusion with <i>all-shared</i> encoders.	28
4.5	Validation accuracy over epochs for $\varepsilon=0.0$ vs $\varepsilon=0.2$ (synthetic validation).	29
4.6	ECE across real subsets for $\varepsilon=0.0$ vs $\varepsilon=0.2$	30
4.7	NLL across real subsets for $\varepsilon=0.0$ vs $\varepsilon=0.2$	30
4.8	ECE before/after Temperature Scaling across real subsets.	31
4.9	NLL before/after Temperature Scaling across real subsets.	32
4.10	Validation accuracy over 10 epochs for three independent runs per configuration.	33

List of Tables

3.1	Domain randomization strategies applied during synthetic data generation.	11
3.2	Default training configuration used across studies. Exact alternatives are described with each ablation.	20
4.1	Freezing policies: accuracy on real test subsets and on synthetic validation (§3.3). Higher is better.	24
4.2	Fusion strategies: synthetic validation accuracy and accuracy on real test subsets (§3.3). Notation: S_k = insertion after Stage k of the backbone.	25
4.3	Weight-sharing policies: accuracy on real test subsets and on synthetic validation (§3.3). Notation: <i>All-shared</i> = same encoder for all 5 views; <i>First-4-shared</i> = side views share weights, top view separate; <i>Unshared</i> = independent encoders per view.	26
4.4	Backbone families under <i>score-sum</i> fusion with <i>all-shared</i> encoders: accuracy on real test subsets and synthetic validation accuracy (§3.3). 28	
4.5	Label smoothing ablation: real-set Accuracy, ECE, and NLL for $\epsilon=0.0$ vs $\epsilon=0.2$ (§3.3). Lower is better for ECE and NLL.	29
4.6	Temperature Scaling on real test subsets (§3.3). ECE and NLL reflect calibration changes with temperature $T = 0.474$ (fitted on synthetic validation). Lower is better for ECE and NLL.	31
4.7	Seed consistency (three runs per configuration): accuracy on real test subsets and best synthetic validation accuracy (mean \pm std). Higher is better.	32

List of Acronyms

PIB Physical Imaging Box

CNN Convolutional Neural Network

MVCNN Multi-View Convolutional Neural Network



Introduction

1.1 CONTEXT AND MOTIVATION

Computer vision is now a core technology in industrial settings. Across manufacturing and logistics, automated perception supports tasks such as visual inspection, defect detection, part tracking, and inventory control. The benefits are well known: higher throughput, fewer errors, and more consistent quality. At the same time, these systems must operate under practical constraints such as limited labeled data, changing product lines, strict latency and reliability requirements.

Among the many application domains, a critical one concerns the handling and preparation of components before assembly. In particular, the *kitting* process groups all the parts required for an operation into a single container, so that the assembly line can work without searching for individual items. In our case this is performed as *pre-kitting*, meaning that the containers are prepared in advance and staged upstream in the warehouse before the actual production run. While this reduces line interruptions later, it transfers the burden of recognition and sorting to warehouse operations. If parts are misidentified or misplaced, the downstream cost is high: delays, rework, or even temporary line stops.

During my curricular internship at M31, an engineering company active in industrial applications of computer vision and automation, I worked on the development of an *automated object classification system* to support operators during pre-kitting, with the goal of reducing mis-sorts and increasing throughput. The work carried out in that setting forms the basis of this thesis.

1.2 PROBLEM SETTING

The proposed system is built around a specialized imaging box equipped with five cameras. When an operator places an object inside, the box captures five synchronized views from fixed angles. A machine learning model then processes these images to classify the object and indicate the correct container. The operator follows the suggestion, closing the loop of an assisted yet automated workflow.

Developing such a system presents unique constraints. A critical one is that the classifier must be operational *as soon as new parts enter production*. Because physical samples of these parts only exist once manufacturing has started, collecting and annotating real images in advance is infeasible. As a result, the initial model must be trained entirely on *synthetic images* rendered from CAD models of the target objects. Moreover, the final deployment must scale to a label space of up to $\sim 10,000$ distinct categories.

1.3 RESEARCH OBJECTIVES

The goal is to design a *multi-view CNN* pipeline that can be trained on synthetic renders yet remain reliable on real images from the imaging box. To this end, the thesis explores the following directions:

- **Adapting pretrained CNN backbones.** How to leverage ImageNet-pretrained encoders for this setting, and whether partial freezing improves transferability while limiting overfitting.
- **Multi-view fusion in CNNs.** How early, late, and score-level fusion mechanisms affect accuracy under synthetic-to-real shift, and which insertion depth is most effective.
- **Parameter sharing across views.** Whether sharing encoder parameters across the five branches improves generalization compared to partially shared or unshared designs.
- **Evaluating backbone families.** The study focuses on established CNN architectures such as ResNet, EfficientNet, and ConvNeXt, while also including a small set of transformer-based models (e.g., Swin, ViT) for comparison within the same pipeline.
- **Assessing calibration and reliability.** How training-time regularization (e.g., label smoothing), post-hoc calibration (temperature scaling), and repeated runs across random seeds influence the stability of the resulting system.

1.4 CONTRIBUTIONS

This thesis makes the following contributions:

- Construction of a **synthetic multi-view dataset** in Blender that mirrors the five-camera imaging box, with controlled randomization of camera, lighting, and materials.
- Implementation of a modular **multi-view CNN pipeline** based on ImageNet-pretrained encoders and fusion modules designed for five synchronized views.
- **Ablation studies** on freezing policies, fusion mechanisms, weight-sharing schemes, backbone families and auxiliary analyses of calibration and reliability.
- **Practical guidelines** for deploying a CNN-based multi-view classifier trained on synthetic data.

1.5 THESIS STRUCTURE

The remainder of this document is organized as follows:

- **Chapter 2** reviews the background on convolutional networks, multi-view recognition, synthetic-to-real transfer, and transfer learning.
- **Chapter 3** describes dataset generation, architectures, and evaluation design, including the pre-registered ablations.
- **Chapter 4** presents the outcomes for freezing, fusion, weight sharing, and backbone comparisons, as well as calibration and reliability analyses.
- **Chapter 5** interprets these findings, highlights implications for deployment, and discusses both limitations and future work.
- **Chapter 6** concludes with a summary of the study.



Related Work

2.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are the standard choice for visual recognition tasks. Their key innovation is the use of local filters applied across the image, which makes them efficient and sensitive to recurring patterns such as edges or textures. Early networks such as LeNet-5 demonstrated this principle for handwritten digits [14], while AlexNet's success on ImageNet in 2012 popularized deep CNNs by combining convolutions, ReLU activations, data augmentation, and GPU training [12]. Since then, CNNs have remained a strong default due to their accuracy, efficiency, and ease of deployment.

Core architecture. A typical CNN processes an image through a sequence of *convolutional layers* that detect local features, followed by *non-linear activations* (commonly ReLU) that allow more complex representations. Spatial resolution is gradually reduced, either by convolutions or pooling operations, until global average pooling (GAP) produces a compact feature vector. Finally, a *fully connected classifier* maps this representation to class logits, which are typically converted to probabilities with a softmax function. This hierarchy gives CNNs their characteristic inductive bias: translation invariance and increasing abstraction from pixels to high-level patterns.

Regularization and stabilization. Modern CNNs combine several techniques to improve generalization and training stability: *data augmentation* (random crops,

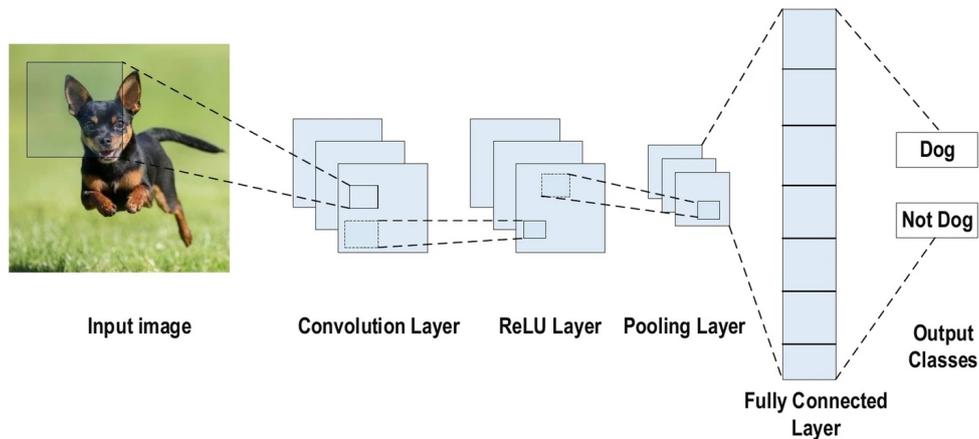


Figure 2.1: Typical architecture of a CNN. *Reproduced from* Alzubaidi et al. [2], licensed under CC BY 4.0.

flips, color jitter) exposes the model to diverse views of the same class; *weight decay* penalizes large parameters [13]; *dropout* prevents reliance on single activations by randomly masking neurons during training [19]; and *batch normalization* stabilizes optimization by normalizing activations [10]. Other methods include *label smoothing*, which softens one-hot targets to reduce overconfidence during training [21], and *temperature scaling*, a post-hoc calibration technique that rescales logits to produce more reliable probability estimates [8]. These techniques are also applied in this thesis, for example when constructing the classifier head on top of ImageNet-pretrained backbones.

In our multi-view classification system, CNNs serve as per-view feature extractors. They are suited to capture shape and texture cues of industrial parts, providing the foundation for the studies reported in later chapters.

2.2 MULTI-VIEW RECOGNITION

A single 2D image often misses important details of a 3D object, especially if parts are hidden. Multi-view recognition solves this by combining information from several viewpoints taken around the object. This strategy has been commonly adopted in shape benchmarks such as ModelNet40 and ShapeNet, and is directly relevant to our industrial setup with a fixed rig of five cameras.

General pipeline. A typical Multi-View Convolutional Neural Network (MVCNN) follows a standard pipeline [1]: (i) render or capture multiple 2D views; (ii) extract features for each view with a backbone; (iii) fuse these features using a chosen strategy; (iv) feed the aggregated representation to a classifier. The design of the fusion mechanism is the most critical factor influencing performance, efficiency, and robustness.

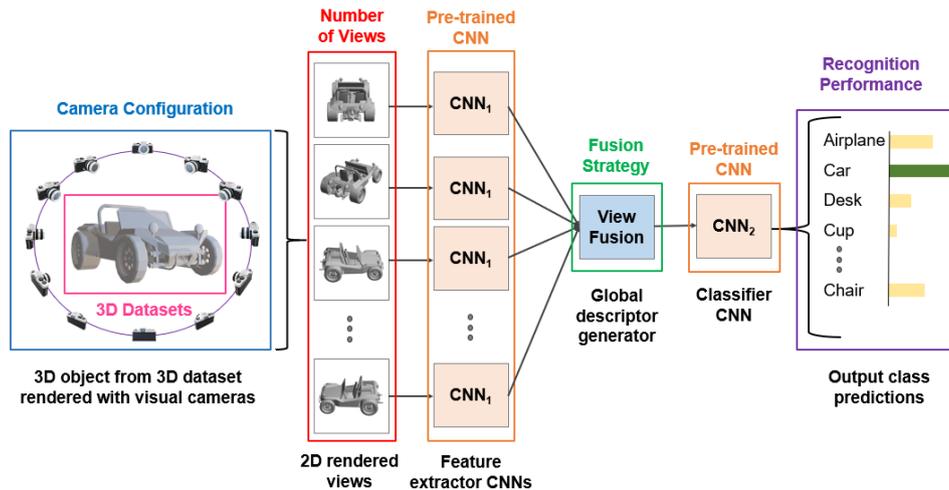


Figure 2.2: MVCNN pipeline: multiple views are processed by a shared backbone, fused, and classified. *Reproduced from Alzahrani et al. [1], licensed under CC BY 4.0.*

Fusion strategies. In a multi-view CNN, the difference between fusion mechanisms is where the views are combined across the pipeline [18, 1]:

- **Early fusion.** Views are merged *inside the backbone*, at the level of convolutional feature maps. For example, feature maps from different views can be pooled or convoluted and then processed by the remaining backbone layers.
- **Late fusion.** Each view is encoded with the backbone, then reduced to a feature vector via global average pooling, producing a compact embedding vector. These per-view embeddings are then fused either by pooling (e.g. max) or by concatenation followed by an additional fully connected layer, *before entering the final classifier*.
- **Score fusion.** Each branch outputs logits or probabilities, which are aggregated *after the classifier* using rules such as summation, maximization, or multiplication in log-space. This strategy is extremely simple as it requires no modification to the backbone.

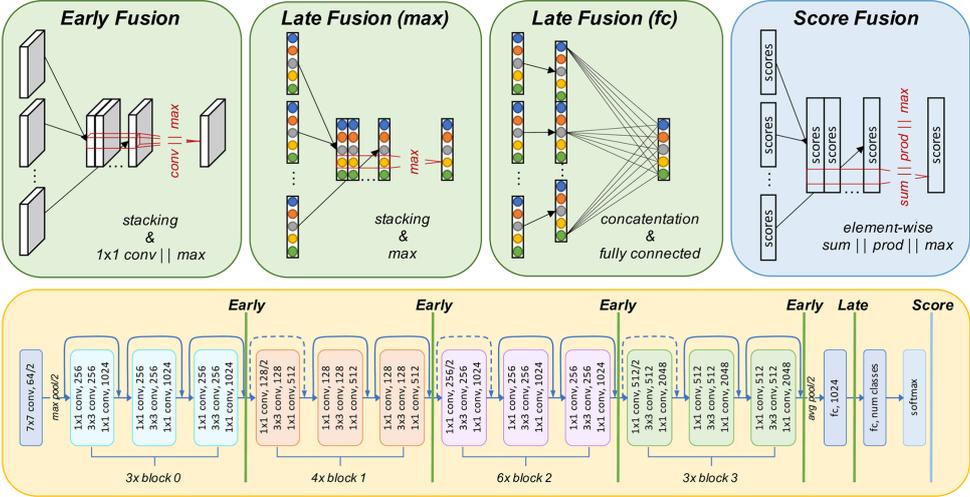


Figure 2.3: Top: early, late, and score fusion strategies. Bottom: fusion strategies mapped onto the ResNet-50 architecture. Adapted from Seeland and Mäder [18], licensed under CC BY 4.0.

Our ablation studies systematically evaluate these three families of fusion in our fixed five camera scenario.

2.3 SYNTHETIC-TO-REAL TRANSFER

Training deep visual models typically requires a substantial number of labeled examples per class, which is often impractical to collect in real-world settings. A common alternative is to generate synthetic training data from 3D models, allowing large-scale datasets with perfect labels to be produced efficiently and at low cost.

Advantages and limitations. Synthetic training offers major advantages: labels come for free, the dataset can be expanded arbitrarily, and rendering parameters can be controlled to simulate deployment conditions. At the same time, synthetic images often lack the imperfections and variability of real data, producing a *domain gap*. Models trained only on synthetic renders may transfer poorly if textures, noise, or occlusions differ significantly in the real environment [23].

Domain randomization. To narrow the gap between synthetic and real data, recent work proposes *domain randomization*, i.e. deliberately varying rendering parameters such as camera pose, focal length, lighting, background, and material

properties [23]. The idea is that if a model is trained on enough randomized variation, the real world will appear as just another variation, improving transfer. This approach has been successfully applied in robotics and 3D object recognition benchmarks.

In our setting, the only option was to generate training data synthetically from CAD models. To mitigate overfitting to synthetic statistics, we adopted domain randomization. This made synthetic-to-real transfer a central challenge of our thesis.

2.4 TRANSFER LEARNING AND PRE-TRAINING

Modern visual recognition models are rarely trained from scratch. Instead, *transfer learning* leverages models pre-trained on large datasets such as ImageNet [6], then adapts them to a new task. Pre-training provides robust low- and mid-level features (edges, textures, shapes) that generalize well across domains, reducing the amount of task-specific data required.

Freezing vs. fine-tuning. When adapting a pre-trained backbone, two common strategies are used: (i) *freezing* some early layers while training only the later layers or classifier head, which preserves generic filters and reduces overfitting; or (ii) *fine-tuning* the entire network, which allows the model to adapt more deeply to the new domain but risks overfitting when little data is available [24]. In practice, freezing early blocks is often effective because low-level features such as edges or textures transfer broadly across datasets.

In our experiments, all backbones were initialized with ImageNet-pretrained weights. We compared different freezing strategies to study their effect on synthetic-to-real transfer.

3

Methods

This chapter describes the methodology for training multi-view object classifiers using synthetic Blender [5] renders and evaluating their transfer to real multi-camera captures. The following sections cover dataset generation and randomization (§3.1), model and fusion designs (§3.2), evaluation protocols (§3.3), and ablation studies (§3.4).

3.1 SYNTHETIC DATASET GENERATION

Due to the lack of physical training samples, the system was trained using synthetic data generated from our 3D Model Corpus, a curated collection of 3D object meshes drawn from public repositories and proprietary CAD models. This section describes the process of creating a Synthetic Multi-View Image Dataset using Blender, a powerful open-source 3D creation suite. The aim was to replicate the Physical Imaging Box (PIB) and inject *domain randomization* via variability in optics, lighting, and enclosure materials. Importantly, generating data in Blender allows for automatic annotation of all samples, eliminating the need for manual labeling and ensuring perfect ground truth for each image.

3.1.1 IMAGING BOX AND CAMERA LAYOUT

The PIB available during development was a repurposed commercial photo tent (Figure 3.1). Although not designed for multi-view acquisition, it provided a temporary testbed while the company’s dedicated five-camera box was under

construction. The synthetic dataset was therefore designed to replicate the same five fixed viewpoints (four lateral corners and one top view) as the final box would provide.



Figure 3.1: Imaging box used for testing.

Static box. The enclosure in Blender was modeled to match the dimensions of the PIB (70 cm \times 70 cm \times 70 cm). Details such as the reflective fabric, PVC background, and other subtleties were omitted; instead, it was represented as a simple white box with flipped normals to ensure interior visibility from the camera viewpoints. The box was configured as a passive rigid body to enable realistic physics interactions with the objects placed inside.

Lighting setup. The PIB employed four LED strips attached to the ceiling in a square configuration. In Blender, this arrangement was approximated using rectangular area lights placed at the center of each edge of the box's top face. Each light measured approximately 56 cm in length and 3.5 cm in width, matching the proportions of the physical setup.

Camera configuration. The five-viewpoint layout of the intended PIB was replicated, with four cameras positioned at the lateral corners and one top-down camera.

3.1.2 DOMAIN RANDOMIZATION

To enhance the diversity of the synthetic dataset and improve model generalization, several *domain randomization* [23] strategies are applied to the scene appearance, camera configuration, and object states during rendering. Table 3.1 summarizes all randomization factors and their ranges.

Table 3.1: Domain randomization strategies applied during synthetic data generation.

Category	Parameter	Randomization Range
Box Material	HSV Hue	$[-0.1, +0.1]$ around white ($H=0$)
	HSV Saturation	$[0.0, 0.2]$
	HSV Value (brightness)	$[0.1, 1.0]$
	Surface properties	Roughness $[0.5, 0.95]$; Specular $[0.0, 0.5]$; Metallicity $[0.0, 0.4]$
Lighting	Light energy	$[7, 50]$ W per light
	Light color	RGB offsets $[-0.7, +0.7]$ per channel
Camera	Focal length	$[4.0, 10.0]$ mm
	Look-at target	Offset (width/depth): $[-5, +5]$ cm
Object Pose	Initial position	Offset (width/depth): $[-7, 7]$ cm
	Initial orientation	Uniform Euler angles $[0, 2\pi]$ on all axes
	Physics properties	Friction 0.7; Restitution 0.2; Damping: linear 0.4, angular 0.6
	Simulation duration	120 frames (settling under gravity)

Box material randomization. The enclosure’s surface properties are varied in HSV color space, starting from a white base and perturbing hue, saturation, and brightness to create diverse color and lighting interactions. Material properties (roughness, specular reflection, metallicity) are also randomized to produce surfaces ranging from matte to slightly reflective.



Figure 3.2: Domain randomization examples: the same object (used in the real captures), *back_left* view, different randomized poses and render settings (five samples).

Lighting randomization. Both the intensity and the color (RGB channel perturbations) of the four area lights are randomized, producing diverse illumination conditions that increase dataset variability.

Camera randomization. Geometric variability is introduced by jittering camera positions, randomizing focal lengths, and offsetting look-at targets. This simulates slight misalignments and ensures each sample is captured from a subtly different viewpoint.

Object pose randomization. Blender’s physics engine is used to generate natural object placements. Objects start with random positions and orientations above the box center, then fall and settle under gravity with realistic material properties (friction, restitution, damping). This physics-driven approach produces diverse, stable poses that would naturally occur in the real industrial setting.

3.1.3 DATASET COMPOSITION

At the time of testing, the client provided only 5 CAD models and one physical object. To broaden the asset pool, 75 additional meshes were selected from Thingi10K [25], a large-scale repository of 3D-printable models. These 80 meshes formed the basis for the synthetic dataset.

For each class, 750 randomized renderings were generated using the strategies described in §3.1.2. Each rendering corresponds to a randomized object placement captured simultaneously from the five fixed viewpoints of the imaging box, resulting in five synchronized images with a shared class label. In total, the dataset comprises 60,000 multi-view samples (300,000 individual images).

Rendering was performed with Blender’s Eevee [4], chosen for its speed compared to the more photorealistic Cycles [3], allowing large-scale generation at manageable cost. Images were rendered at 256×256 resolution, balancing efficiency with sufficient detail for feature extraction.

The dataset was split 80/20 at the sample level, yielding 48,000 training samples (240,000 images) and 12,000 validation samples (60,000 images). No separate synthetic test set was created, since evaluation focused on transfer to real-world data.

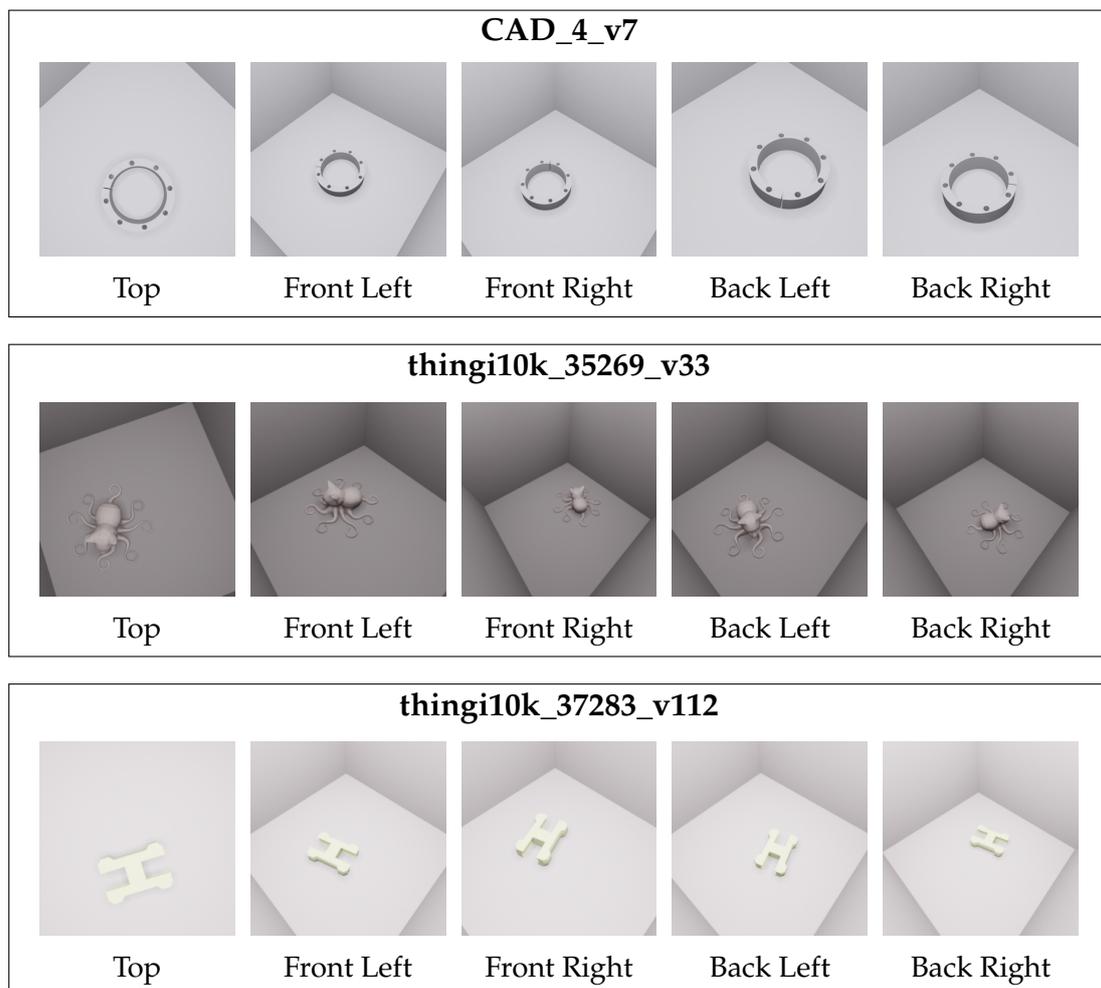


Figure 3.3: Three example synthetic samples showing the five views produced by the pipeline; top row: sample of the 4th CAD object provided by the client; middle and bottom rows: samples from the selected Thingi10K set.

3.2 MODEL ARCHITECTURES AND DESIGN CHOICES

This section details the design of the multi-view classification pipeline.

We describe the backbone encoders used to process each view, the fusion mechanisms to combine them, schemes for weight sharing across branches, adaptation policies for pretrained encoders (freezing), and the structure of the classifier head. Together, these components define the model families compared in the ablation studies of Chapter 4. All models were implemented and trained using *TensorFlow/Keras* [17].

3.2.1 BACKBONE ENCODERS AND INPUT HANDLING

Each view in a sample of the synthetic dataset is processed by an ImageNet-pretrained encoder (*backbone*) to extract a per-view representation. Backbones span both convolutional and transformer families, covering a spectrum of inductive biases and capacities.

Backbone families. We evaluate residual CNNs (**ResNet-50/152** [9]), compound-scaled CNNs (**EfficientNet-B0** [22]), modern convolutional networks (**ConvNeXt-Small** [16]), a hierarchical vision transformer (**Swin-Tiny** [15]), and a pure transformer (**ViT-B/16** [7]). All are used as feature extractors; classification heads are defined separately.

Input and preprocessing. Rendered images are 256×256 (§ 3.1), resized to the encoder’s expected input size (224×224), and normalized according to each backbone’s native preprocessing.

Exported features. Depending on the fusion family used (§3.2.2), backbones export different tensor forms. For *early* fusion, intermediate feature maps $F_v \in \mathbb{R}^{H \times W \times C}$ are taken at a chosen insertion point (e.g., after `conv2_block3_out`, `conv3_block4_out`, `conv4_block6_out`, or `conv5_block3_out` for ResNet-50). For *late* fusion, each view is reduced to a pooled feature vector $z_v \in \mathbb{R}^C$. For *score* fusion, per-view classifier heads output probability vectors $p_v \in \mathbb{R}^K$. These exported representations are the inputs to the fusion operations described in the next section. Encoder weight-sharing schemes are detailed in §3.2.3.

3.2.2 FUSION STRATEGIES

In the following, we present the exact fusion variants as implemented in our experiments. We adopt the common taxonomy of multi-view fusion into *early feature-level*, *late feature-vector*, and *score-level* fusion [18].

Early feature-level fusion. Each view is encoded up to an insertion point within the backbone, producing intermediate feature maps $F_v \in \mathbb{R}^{H \times W \times C}$. We evaluate two operators that reduce the stacked tensor $\tilde{F} = \text{Concat}(F_1, \dots, F_5) \in \mathbb{R}^{H \times W \times 5C}$ to a single fused map $F_{\text{fused}} \in \mathbb{R}^{H \times W \times C}$:

$$(i) \text{ Max over views: } F_{\text{fused}}(i, j, c) = \max_{v \in \{1, \dots, 5\}} F_v(i, j, c),$$

$$(ii) 1 \times 1 \text{ adapter: } F_{\text{fused}} = \text{ReLU}(W * \tilde{F}), \quad W \in \mathbb{R}^{1 \times 1 \times 5C \times C},$$

where $*$ denotes convolution, ReLU is the rectified linear unit activation, and W are the learnable weights of the 1×1 convolution that determine how features from all views are linearly combined at each spatial location. The adapter outputs C channels to remain compatible with the resumed backbone stages. The fused map is then forwarded through the *remaining* backbone stages and into the classifier head. In our implementation, early fusion is instantiated on the ResNet family by splitting the network at stage boundaries, which we abbreviate as Sk (e.g., $S2 =$ insertion after Stage 2).

Late feature-vector fusion. Each view is encoded into a 1024-dimensional feature vector h_v using global average pooling on the final convolutional map, followed by a fully connected layer, batch normalization, and dropout. These vectors are then fused in two alternative ways:

$$(i) \text{ Max over views: } z_{\text{fused}}(c) = \max_v h_v(c),$$

$$(ii) \text{ Concatenation + FC: } z_{\text{fused}} = \phi([h_1; \dots; h_5]),$$

where ϕ is a fully connected layer with 1024 units.

Score-level fusion. Each per-view branch outputs a class probability vector $p_v \in \mathbb{R}^K$ via a softmax layer. We aggregate the five probability vectors into a fused

score $s \in \mathbb{R}^K$ by one of three element-wise rules:

$$(i) \text{ Sum: } s(k) = \sum_{v=1}^5 p_v(k),$$

$$(ii) \text{ Max: } s(k) = \max_v p_v(k),$$

$$(iii) \text{ Geometric mean (log-product): } s(k) = \exp\left(\frac{1}{5} \sum_{v=1}^5 \log(p_v(k) + \varepsilon)\right),$$

where all operations are performed element-wise over the class index k , and $\varepsilon = 10^{-7}$.

The fused score s is then normalized via softmax to yield the final prediction probability $p_{\text{fused}} = \text{softmax}(s) \in \mathbb{R}^K$, ensuring $\sum_{k=1}^K p_{\text{fused}}(k) = 1$.

Implementation scope across backbones. Early fusion was implemented only for ResNet-style encoders, where stage boundaries (Stage 2–5) allow a clean split–fuse–resume design. Late and score fusion are implemented uniformly across all backbones considered. Extending early fusion to other CNNs with stage-structured architectures would follow the same pattern. Extending it to transformer encoders would require non-trivial design modifications and was not explored.

3.2.3 WEIGHT-SHARING SCHEMES

We explore three regimes for weight sharing across the five per-view encoders, while keeping the per-view classifier heads unshared to allow view-specific calibration.

We denote by $E(x; \theta)$ the output of an encoder E applied to input x with parameters θ . When encoders differ across views we write $E_v(\cdot)$, where $v \in \{1, \dots, 5\}$ indexes the camera view.

All-shared. A single encoder $E(\cdot; \theta)$ is reused for all views, encouraging view-agnostic representations and minimizing parameters.

Partially-shared. The four lateral views share one encoder $E(\cdot; \theta_{\text{lat}})$, while the top view uses a separate instance $E(\cdot; \theta_{\text{top}})$. This is motivated by the geometric

similarity of the four lateral cameras, which can benefit from lateral regularization, while allowing the top-down view to specialize separately.

Unshared. Each view has its own encoder $E_v(\cdot; \theta_v)$, maximizing capacity but increasing encoder parameters by roughly 5×.

3.2.4 FREEZING POLICIES

We define four regimes for adapting ImageNet-pretrained encoders, differing in how many early stages are kept fixed during training.

Fully trainable. All stages of the encoder are updated during training, providing maximum flexibility to adapt to the synthetic domain.

Stages 1–3 frozen. The first three convolutional stages are kept fixed, while later stages are fine-tuned. This preserves generic low-level filters while allowing adaptation in deeper layers.

Stages 1–4 frozen. Both early and mid-level features are preserved by freezing the first four stages. Only the final block and classifier are updated, reducing the number of trainable parameters.

Fully frozen. The entire backbone is kept fixed, and only the fusion layers and classifier head are trained. This minimizes trainable capacity and serves as a lower-bound reference.

3.2.5 CLASSIFIER HEAD

The classifier head maps either fused features or per-view features into class scores over K classes. Unless otherwise specified, heads follow a simple design: global average pooling (GAP) of the encoder output, a fully connected layer of width 1024 with ReLU activation, batch normalization and dropout, and a final softmax layer.

For *early fusion* (§3.2.2), the head is applied once to the single fused stream. For *late fusion*, each view first produces a 1024-D vector; these are combined by the chosen operator (max or concatenation+FC) before classification. For *score fusion*,

each view has its own head producing per-view class probabilities, which are then aggregated by the fusion rule (sum, max, or log-product) and normalized by a final softmax.

3.3 EVALUATION PROTOCOL

EVALUATION SETS

Model selection is based on the synthetic validation split defined in §3.1. Final evaluation targets real data acquired with the PIB, using the same view configuration as in the synthetic pipeline. To assess robustness, real samples are grouped into four subsets:

- **Good conditions:** clean captures under intended setup. This set contains 10 samples (50 images in total, five views per sample).

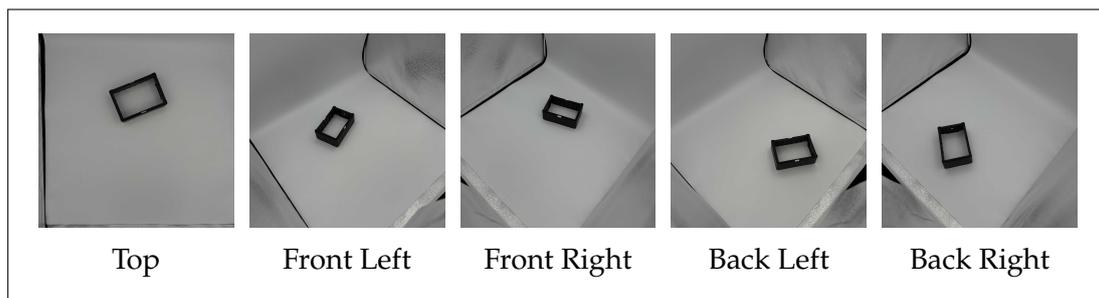


Figure 3.4: Sample five-view capture from the *Good Conditions* subset.

- **Edge cases:** objects placed near walls or corners of the box. This set contains 5 samples (25 images).

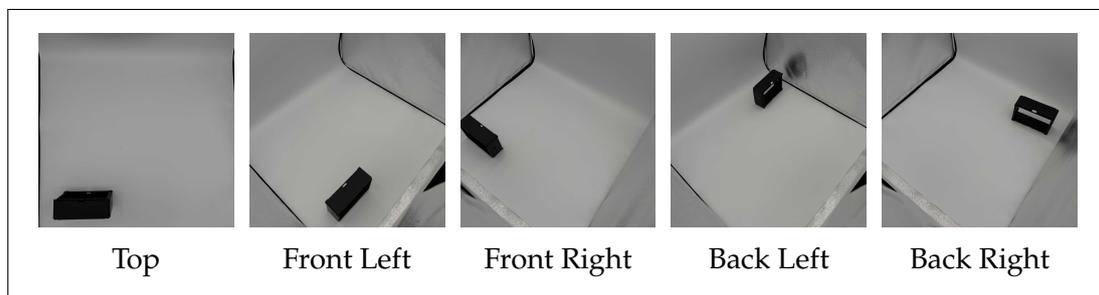


Figure 3.5: Sample five-view capture from the *Edge cases* subset.

- **Objects interference:** additional foreign items present in the scene. This set contains 5 samples (25 images).

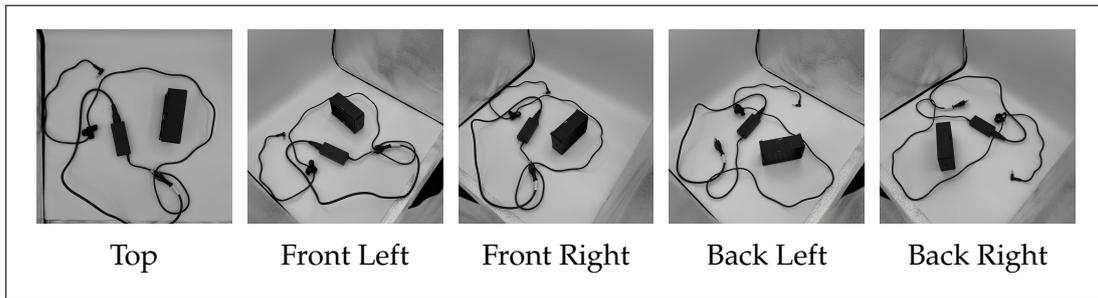


Figure 3.6: Sample five-view capture from the *Objects interference* subset.

- **View occlusion:** one or more cameras deliberately blocked, simulating sensor failure. This set contains 5 samples (25 images).

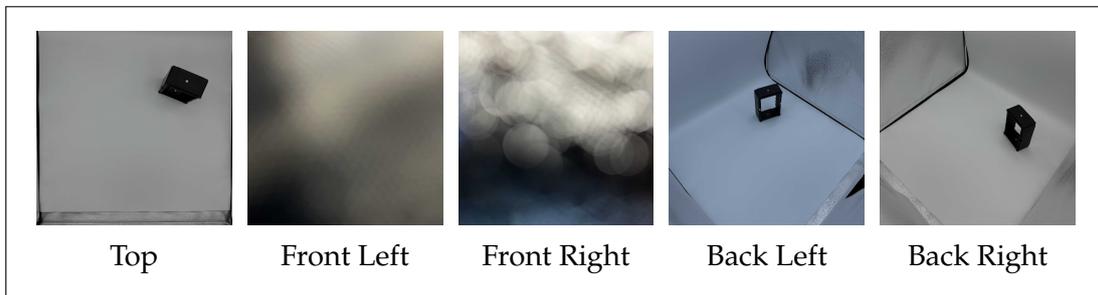


Figure 3.7: Sample five-view capture from the *View occlusion* subset.

All real subsets were captured using the *same physical object*, the only physical part available at the time of testing. Consequently, evaluation covers only a single class in real conditions.

METRICS AND MODEL SELECTION

The **primary endpoint** is the accuracy on the *Real / Good conditions* subset. **Secondary endpoints** include accuracy on the other real subsets (Edge, Objects interference, View occlusion) and on the synthetic test set split. In addition, we report *calibration metrics* as Expected Calibration Error (ECE) and Negative Log-Likelihood (NLL) to assess the quality of predictive probabilities.

3.4 ABLATION STUDIES

This section pre-registers all experimental comparisons reported in Chapter 4. Unless otherwise stated, all runs use the baseline configuration in Table 3.2, with train/validation splits defined in §3.1.3. The primary and secondary endpoints are

defined in §3.3. Models are selected at the epoch achieving the highest accuracy on the synthetic validation split, and the chosen checkpoint is then evaluated on the real evaluation subsets (§3.3).

Table 3.2: Default training configuration used across studies. Exact alternatives are described with each ablation.

Component	Configuration
Loss	Categorical cross-entropy
Optimizer	AdamW with weight decay 0.004; TensorFlow defaults: $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-7}$
Learning rate	Cosine decay, $\eta_0 = 10^{-5}$, floor $\alpha = 10^{-7}$
Batch size	16
Preprocessing	Resize to encoder input (224×224); backbone-specific normalization
Augmentation	None during training; variability from Blender domain randomization (§3.1)
Fusion strategy	Score-sum
Freezing policy	Stages 1–3 frozen
Weight sharing	All-shared encoders
Label smoothing	$\varepsilon = 0.1$
Head regularization	Dropout 0.25 on the penultimate dense layer (§3.2.5)

FREEZING POLICIES

RQ: Does freezing early backbone layers improve stability without reducing transfer performance?

Factors: Four policies: None (fully trainable); Stages 1–3 frozen; Stages 1–4 frozen; Full backbone frozen.

Controls: ResNet-50 backbone; early-max fusion with insertion after Stage 2; unshared encoders; label smoothing $\varepsilon = 0.0$.

Epochs/Runs: 20 epochs; 1 run/configuration.

FUSION STRATEGIES

RQ: Which fusion strategy (early, late, score) and insertion depth (Stages 2–5) yields higher real-data accuracy?

Factors: Early fusion after Stages 2–5 with (a) reduce-max, (b) 1×1 adapter; Late fusion with (a) reduce-max, (b) concatenation+FC; Score fusion with (a) sum, (b) max, (c) log-product. (§3.2.2)

Controls: ResNet-50; unshared encoders; Stages 1–3 frozen; label smoothing $\varepsilon = 0.1$.

Epochs/Runs: 10 epochs; 1 run/configuration.

WEIGHT-SHARING SCHEMES

RQ: Does sharing encoder parameters across views improve transfer?

Factors: (i) all-shared, (ii) partially shared, (iii) unshared.

Controls: ResNet-50; score-sum fusion; Stages 1–3 frozen; label smoothing $\varepsilon = 0.1$.

Epochs/Runs: 10 epochs; 1 run/configuration.

BACKBONE FAMILIES

RQ: Can newer architectures such as EfficientNet, ConvNeXt, and transformers achieve better synthetic-to-real transfer than traditional residual CNNs?

Factors: ResNet-50/152, EfficientNet-B0, ConvNeXt-Tiny/Small, Swin-Tiny, ViT-B/16.

Controls: Score-sum fusion; all-shared; Stages 1–3 frozen; label smoothing $\varepsilon = 0.1$.

Epochs/Runs: 10 epochs; 1 run/configuration.

LABEL SMOOTHING

RQ: How does label smoothing affect transfer and calibration?

Factors: $\varepsilon \in \{0, 0.2\}$.

Controls: ResNet-50; score-sum fusion; unshared encoders; Stages 1–3 frozen.

Epochs/Runs: 5 epochs; 1 run/configuration.

TEMPERATURE SCALING

RQ: Can temperature scaling improve calibration on real data?

Procedure/Factors: Compare uncalibrated logits vs logits scaled by a temperature T .

Controls: Applied post-hoc to a representative trained model (ResNet-50, score-sum, all-shared, Stages 1–3 frozen). T is fitted on the synthetic validation set by minimizing NLL.

Epochs/Runs: 1 calibration/model.

CONSISTENCY ACROSS SEEDS

RQ: How variable are outcomes across random seeds?

Design: Retrain two contrasting configurations three times each with different seeds: (i) ResNet-50 with score-level fusion (sum); (ii) ResNet-50 with early fusion after Stage 3 (convolutional variant).

Controls: Same optimizer, schedule, and epochs as the base configurations.

Epochs/Runs: 10 epochs; 3 runs/configuration.

4

Results

This chapter presents the outcomes of the ablation studies and evaluation experiments. We begin by outlining the reporting conventions. Subsequent sections report results for each study family, organized around the experimental factors defined in § 3.4.

4.1 REPORTING CONVENTIONS

To ensure consistency and clarity, we adopt the following conventions throughout this chapter:

- **Model selection.** Results are reported from the checkpoint achieving the highest accuracy on the synthetic validation set (§ 3.4).
- **Primary endpoint.** The main evaluation metric is the accuracy on the *Good Conditions* test set (§ 3.3).
- **Secondary endpoints.** We also report accuracy on the synthetic test set and on additional real subsets (*Edges, Interference, Occlusion*). Calibration quality is assessed via *ECE* and *NLL*.
- **Best results.** The best-performing configuration in each block is highlighted in **bold**.

4.2 FREEZING POLICIES

Setup. As defined in §3.4, ResNet-50 with early-max insertion after Stage-2. We compare four freezing policies while keeping all other settings fixed.

Outcomes. Table 4.1 reports results on the primary and secondary endpoints. All policies except the fully frozen backbone reach 1.00 on Real / Good. The *Stage 1–4 frozen* variant shows lower accuracy on the Interference and Occlusion subsets, while the fully frozen backbone records the lowest scores across all real subsets. Figure 4.1 shows the corresponding validation accuracy curves.

Freezing Policy	Accuracy (Good)	Accuracy (Edges)	Accuracy (Interf.)	Accuracy (Occ.)	Accuracy (Synthetic)
Fully trainable	1.00	1.00	1.00	1.00	0.988
Freeze Stage 1–3	1.00	1.00	1.00	1.00	0.986
Freeze Stage 1–4	1.00	1.00	0.60	0.60	0.976
Fully frozen backbone	0.70	0.80	0.00	0.40	0.918

Table 4.1: Freezing policies: accuracy on real test subsets and on synthetic validation (§3.3). Higher is better.

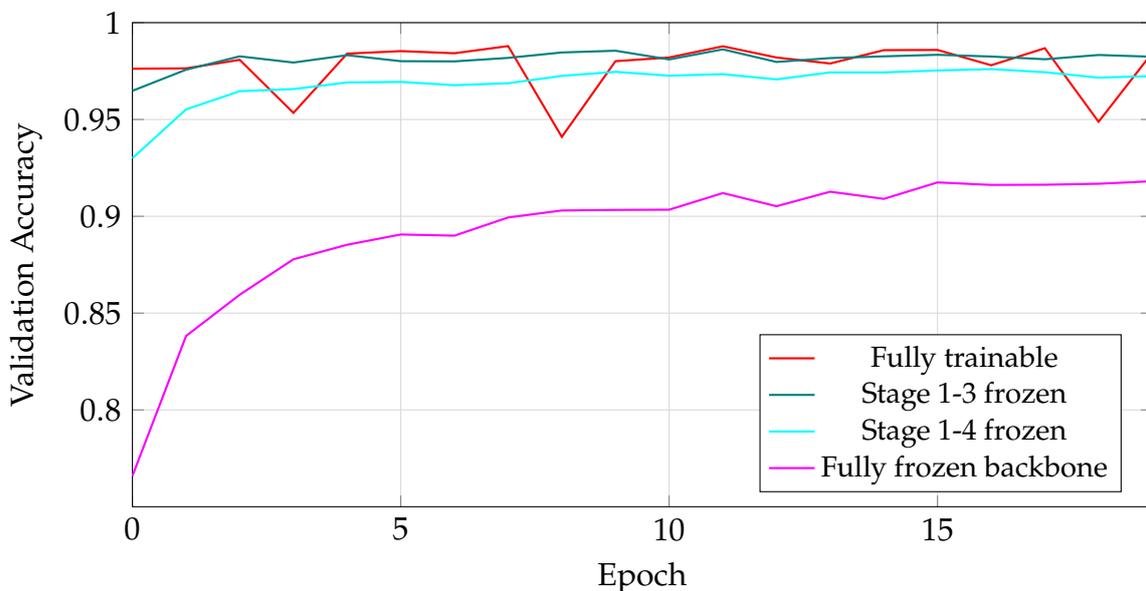


Figure 4.1: Validation accuracy evolution during training for the different freezing policies.

4.3 FUSION STRATEGIES

Setup. As defined in §3.4, we compare three fusion families with *ResNet-50* backbones while keeping all other settings fixed.

Outcomes. Table 4.2 and Figure 4.2 summarize the results. On the primary endpoint (Real/Good), several configurations reached perfect accuracy, including all the *Score* variants and multiple early-fusion variants (S5 conv, S4 max, S3 max, S2 max). Differences appeared more clearly on the stress subsets: *Early (S5 conv)* and *Early (S4 max)* achieved the best results on Edges and Occlusion, whereas *Score (sum)* remained lower on these subsets (0.20–0.60). *Late (FC)* showed a more balanced profile, with accuracy of 0.80 on Interference and 0.90 on Good, though not the top performer on other subsets. On synthetic validation, most strategies stabilized above 0.985, with *Score (sum)* slightly ahead (0.9885).

Fusion configuration	Accuracy (Good)	Accuracy (Edges)	Accuracy (Interf.)	Accuracy (Occ.)	Accuracy (Synthetic)	Params (MB)
Early (S5, conv)	1.00	0.80	0.80	0.80	0.9879	178.03
Early (S5, max)	0.90	0.80	0.40	0.40	0.9881	98.02
Early (S4, conv)	0.70	0.00	0.00	0.40	0.9867	118.02
Early (S4, max)	1.00	1.00	0.60	0.80	0.9880	98.02
Early (S3, conv)	0.00	0.00	0.00	0.00	0.9644	103.02
Early (S3, max)	1.00	1.00	0.60	0.80	0.9871	98.02
Early (S2, conv)	0.00	0.00	0.00	0.00	0.9133	99.27
Early (S2, max)	1.00	0.80	0.60	0.60	0.9829	98.02
Score (sum)	1.00	0.60	0.20	0.60	0.9885	490.10
Score (max)	1.00	0.60	0.00	0.40	0.9816	490.10
Score (product)	1.00	0.20	0.00	0.40	0.9854	490.10
Late (FC)	0.90	0.40	0.80	0.60	0.9877	510.04
Late (max)	0.90	0.40	0.40	0.40	0.9853	490.02

Table 4.2: Fusion strategies: synthetic validation accuracy and accuracy on real test subsets (§3.3). Notation: S_k = insertion after Stage k of the backbone.

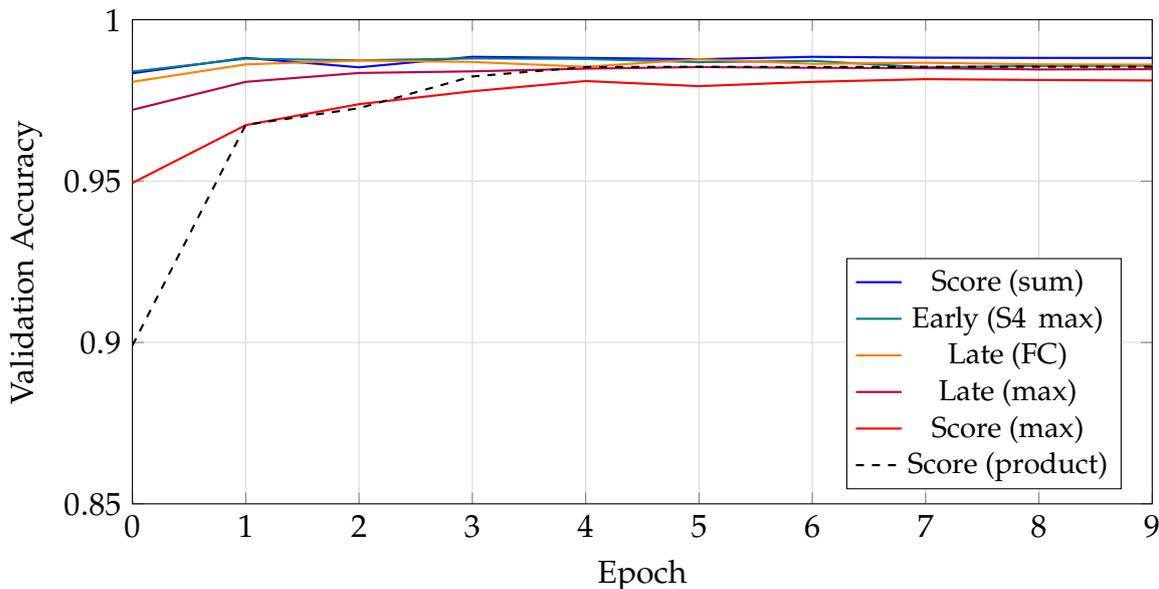


Figure 4.2: Validation accuracy evolution for representative fusion strategies over 10 epochs.

4.4 WEIGHT-SHARING SCHEMES

Setup. As defined in §3.4, we compare three weight-sharing policies with *ResNet-50* backbones and *score-sum* fusion. All other settings are kept fixed.

Outcomes. Table 4.3 shows that all three policies reach similar synthetic accuracy (≈ 0.989). On real subsets, *All-shared* achieves the highest accuracy across conditions, followed by *First-4-shared* and *Unshared*. Validation curves in Figure 4.3 overlap closely.

Weight sharing	Accuracy (Good)	Accuracy (Edges)	Accuracy (Interf.)	Accuracy (Occ.)	Accuracy (Synthetic)	Params (MB)
All-shared	1.00	1.00	0.60	1.00	0.9890	130.18
First-4-shared	1.00	0.80	0.20	0.80	0.9888	220.16
Unshared	0.80	0.80	0.00	0.20	0.9888	490.10

Table 4.3: Weight-sharing policies: accuracy on real test subsets and on synthetic validation (§3.3). Notation: *All-shared* = same encoder for all 5 views; *First-4-shared* = side views share weights, top view separate; *Unshared* = independent encoders per view.

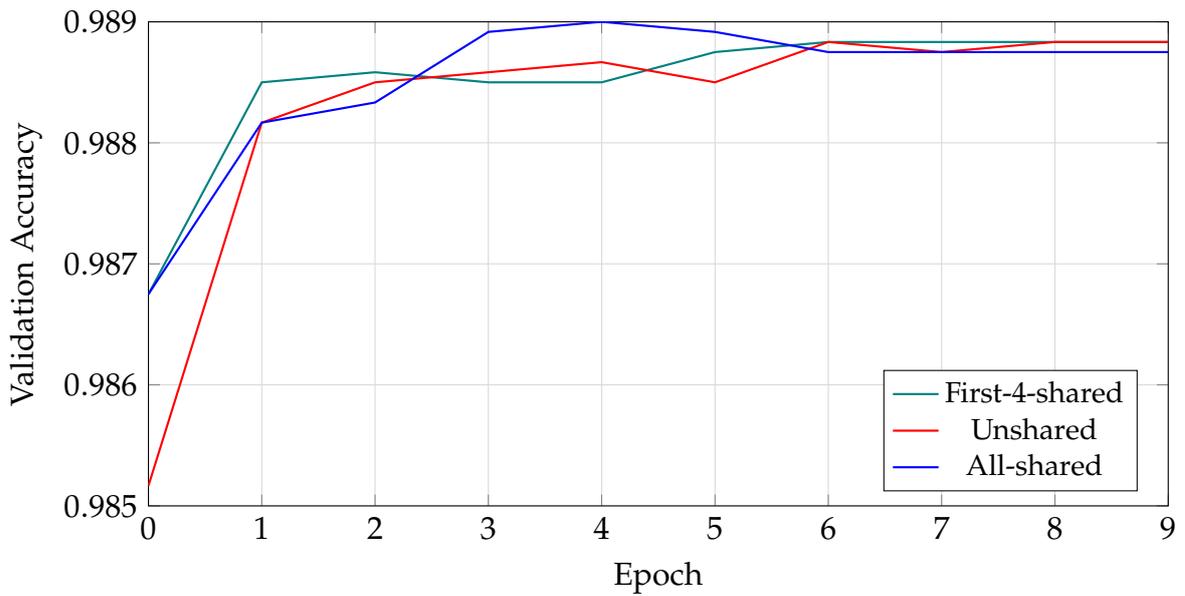


Figure 4.3: Validation accuracy over 10 epochs for the three weight-sharing policies.

4.5 BACKBONE FAMILIES

Setup. As defined in §3.4, we evaluate multiple backbones under *score-sum* fusion with *all-shared* encoders.

Outcomes. Table 4.4 reports accuracy on the real subsets and on synthetic validation. *ConvNeXtSmall* and *SwinTiny* achieve 1.00 across all real subsets; *EfficientNetB0* is competitive on real sets with the smallest parameter footprint. *ResNet152* and *ViT-B/16* show lower accuracy on real subsets. Figure 4.4 shows validation accuracy trajectories over 10 epochs.

Backbone	Accuracy (Good)	Accuracy (Edges)	Accuracy (Interf.)	Accuracy (Occ.)	Accuracy (Synthetic)	Params (MB)
ResNet-50	1.00	1.00	0.60	1.00	0.9890	130.18
Swin-Tiny	1.00	1.00	1.00	1.00	0.9887	120.17
ResNet-152	0.60	0.60	0.00	0.40	0.9887	262.86
ConvNeXt-Tiny	1.00	1.00	0.80	1.00	0.9887	121.32
EfficientNet-B0	1.00	1.00	0.60	1.00	0.9884	40.64
ViT-B/16	0.20	0.20	0.40	0.20	0.9887	342.49
ConvNeXt-Small	1.00	1.00	1.00	1.00	0.9889	203.85

Table 4.4: Backbone families under *score-sum* fusion with *all-shared* encoders: accuracy on real test subsets and synthetic validation accuracy (§3.3).

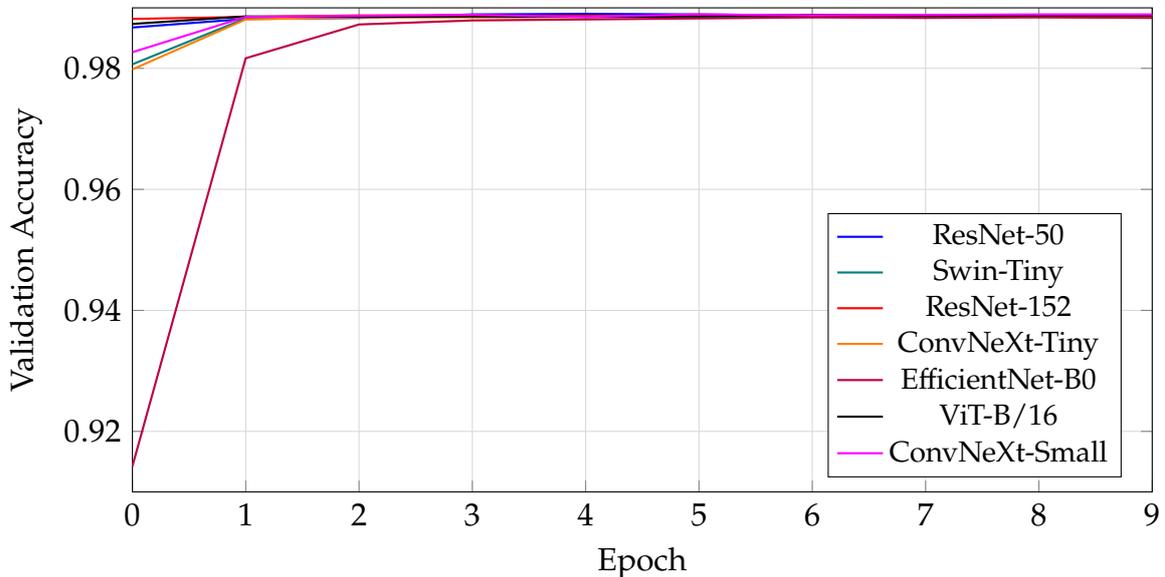


Figure 4.4: Validation accuracy over 10 epochs for backbone families under *score-sum* fusion with *all-shared* encoders.

4.6 LABEL SMOOTHING

Setup. As defined in §3.4, we evaluate the effect of training-time label smoothing on the selected configuration (ResNet-50 encoder, *score-sum* fusion, *unshared* weights). We compare $\varepsilon \in \{0.0, 0.2\}$ while keeping all other settings fixed. Calibration is assessed with NLL and ECE.

Outcomes. Table 4.5 reports Accuracy, ECE and NLL on the real subsets for $\varepsilon=0.0$ vs $\varepsilon=0.2$. On average across real subsets, accuracy drops from 0.85

to 0.80, while calibration degrades (ECE from 0.172 to 0.581; NLL from 0.659 to 1.794). Figure 4.5 shows the synthetic validation accuracy curves; Figures 4.6–4.7 summarize per-split ECE and NLL.

Real subset	Accuracy		ECE		NLL	
	$\varepsilon=0.0$	$\varepsilon=0.2$	$\varepsilon=0.0$	$\varepsilon=0.2$	$\varepsilon=0.0$	$\varepsilon=0.2$
Real / Good	1.00	1.00	0.021	0.629	0.0219	1.0980
Real / Partial occlusion	0.60	0.60	0.365	0.598	1.8700	2.7564
Real / Edges	1.00	0.80	0.027	0.488	0.0291	1.4280
Real / Interference	0.80	0.80	0.274	0.611	0.7166	1.8925

Table 4.5: Label smoothing ablation: real-set Accuracy, ECE, and NLL for $\varepsilon=0.0$ vs $\varepsilon=0.2$ (§3.3). Lower is better for ECE and NLL.

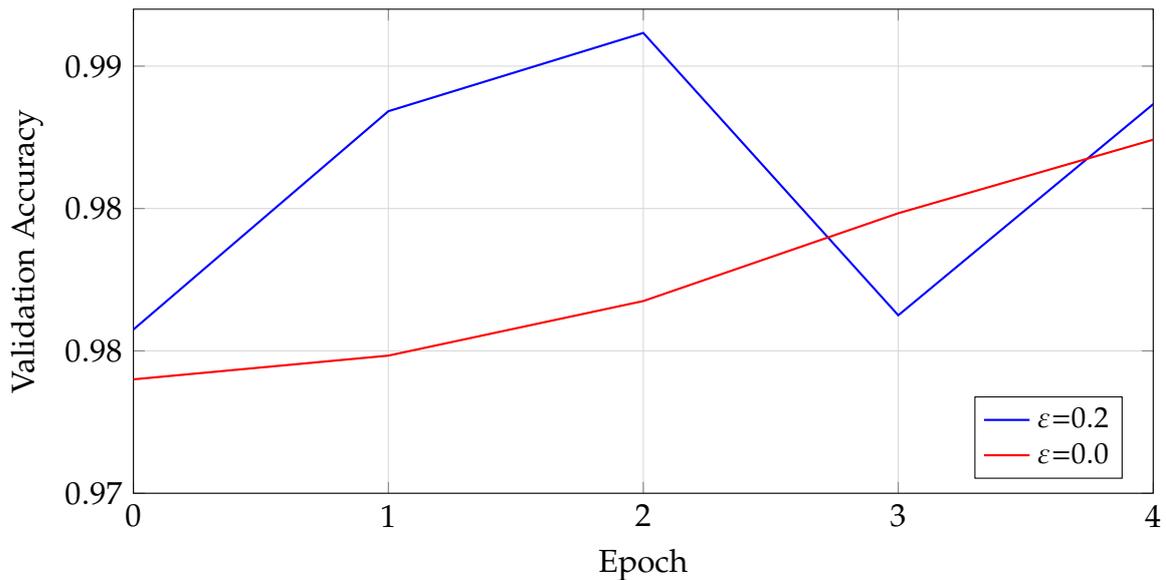
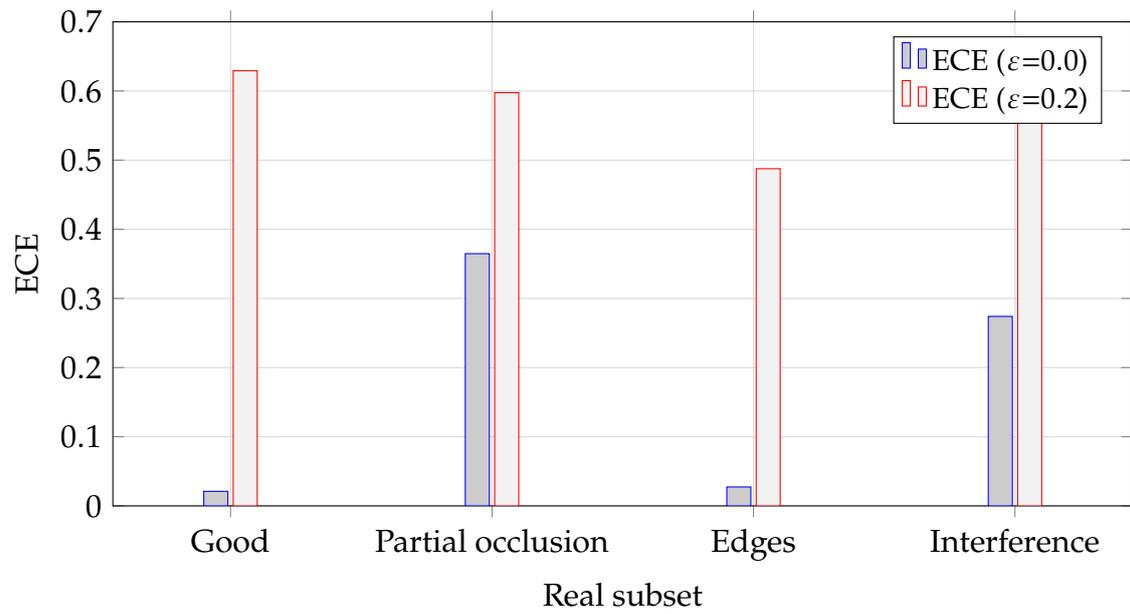
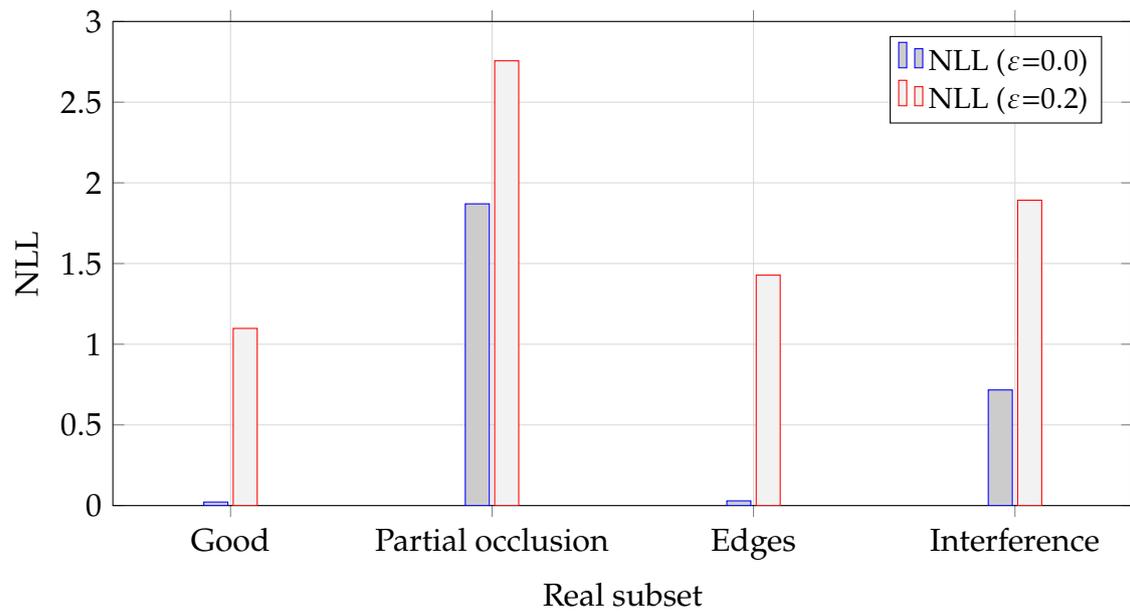


Figure 4.5: Validation accuracy over epochs for $\varepsilon=0.0$ vs $\varepsilon=0.2$ (synthetic validation).

Figure 4.6: ECE across real subsets for $\epsilon=0.0$ vs $\epsilon=0.2$.Figure 4.7: NLL across real subsets for $\epsilon=0.0$ vs $\epsilon=0.2$.

4.7 TEMPERATURE SCALING

Setup. As defined in §3.4, we apply post-hoc Temperature Scaling (TS) to the selected *score-sum* configuration. The temperature T is fitted once on the synthetic

validation set.

Outcomes. Table 4.6 reports Accuracy, NLL and ECE before/after TS across real subsets. Accuracy is unchanged; NLL decreases on all subsets; ECE improves on *Good* and *Partial occlusion*, and worsens on *Edges* and *Interference*. Figures 4.8–4.9 summarize ECE and NLL shifts.

Real subset	Accuracy		ECE		NLL	
	pre	post	pre	post	pre	post
Real / Good	1.00	1.00	0.776	0.228	1.611	0.305
Real / Partial occlusion	0.60	0.60	0.552	0.346	2.706	1.446
Real / Edges	0.60	0.60	0.564	0.653	2.737	1.819
Real / Interference	0.20	0.20	0.166	0.472	3.337	2.678

Table 4.6: Temperature Scaling on real test subsets (§3.3). ECE and NLL reflect calibration changes with temperature $T = 0.474$ (fitted on synthetic validation). Lower is better for ECE and NLL.

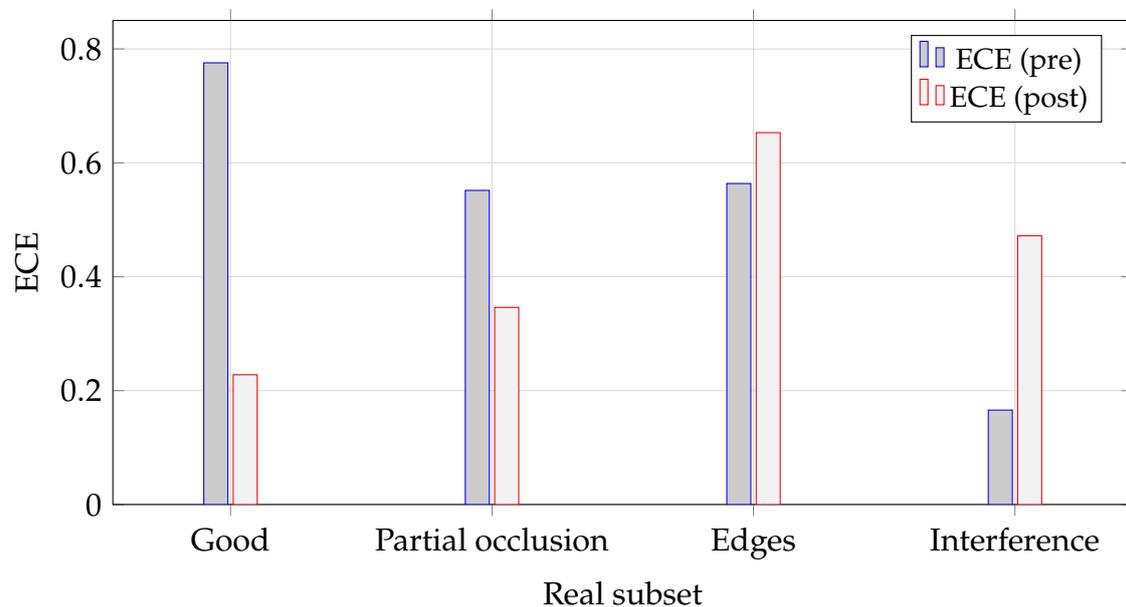


Figure 4.8: ECE before/after Temperature Scaling across real subsets.

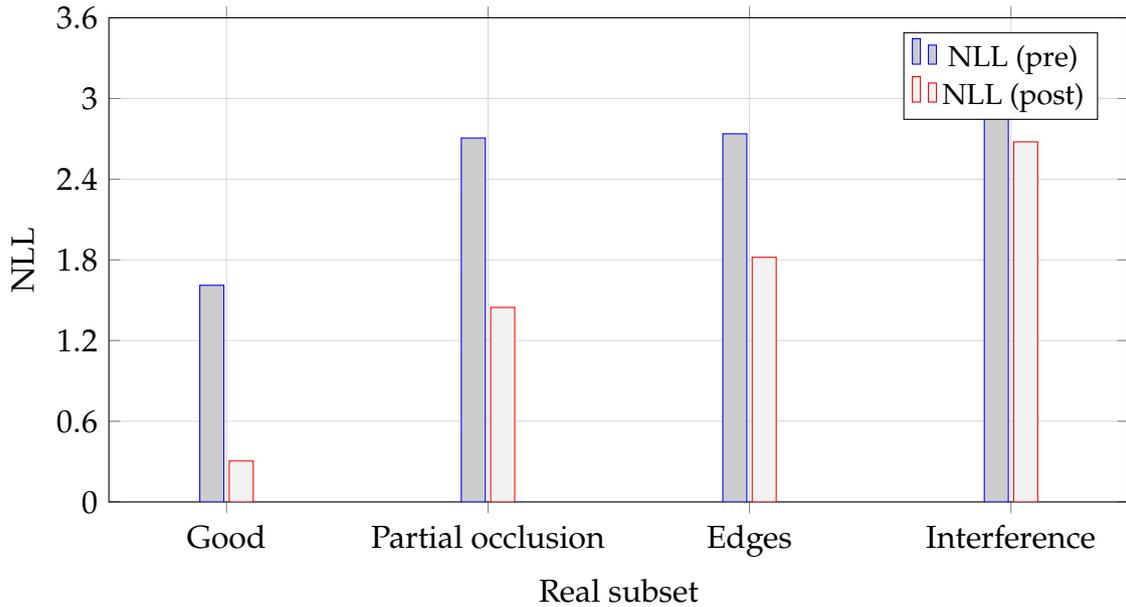


Figure 4.9: NLL before/after Temperature Scaling across real subsets.

4.8 CONSISTENCY ACROSS SEEDS

Setup. As defined in §3.4, we repeat training for two configurations with three independent runs each: *ResNet-50 (score-sum, all-shared)* and *ResNet-50 (early, S3 conv)*.

Outcomes. Table 4.7 reports mean \pm std across runs for all endpoints. *Score (sum)* shows zero variance on *Real / Good* and low variance on the stress subsets. *Early (S3, conv)* completely fails across all real subsets in every run. Figure 4.10 plots validation accuracy curves per run.

Configuration	Accuracy (Good)	Accuracy (Edges)	Accuracy (Interf.)	Accuracy (Occ.)	Accuracy (Synthetic)
Score (sum)	1.00 \pm 0.00	0.67 \pm 0.12	0.20 \pm 0.00	0.53 \pm 0.12	0.9885 \pm 0.0002
Early (S3, conv)	0.00 \pm 0.00	0.00 \pm 0.00	0.00 \pm 0.00	0.00 \pm 0.00	0.9654 \pm 0.0009

Table 4.7: Seed consistency (three runs per configuration): accuracy on real test subsets and best synthetic validation accuracy (mean \pm std). Higher is better.

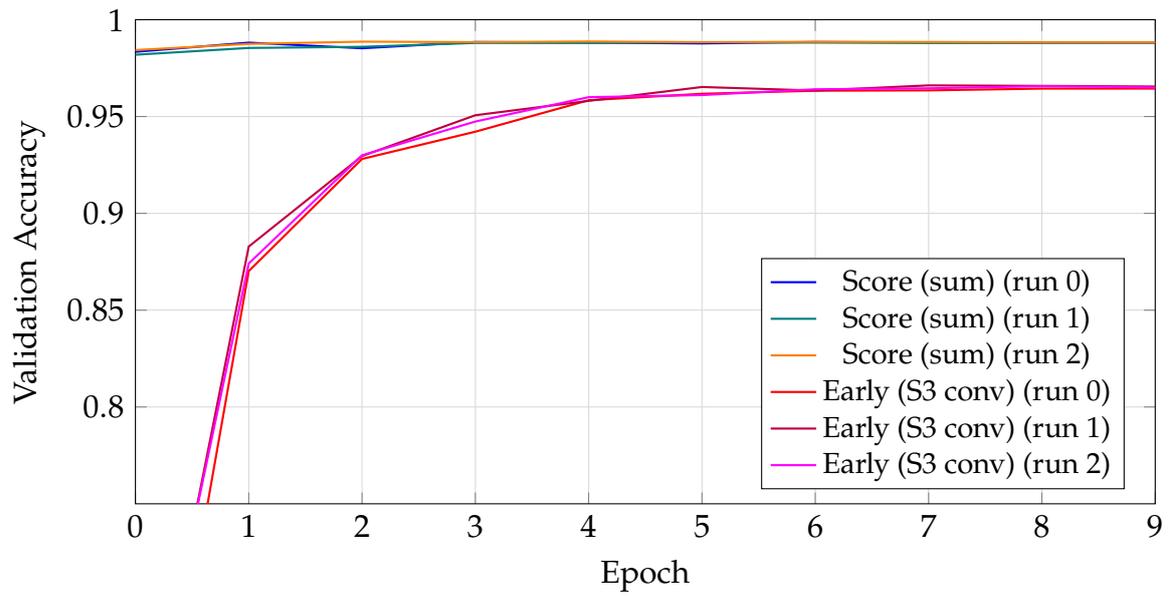


Figure 4.10: Validation accuracy over 10 epochs for three independent runs per configuration.

5

Discussion

This chapter interprets the experimental outcomes reported in Chapter 4. We discuss freezing regimes, fusion strategies, weight sharing, backbone capacity, regularization, calibration, and robustness. Each subsection analyzes why the observed rankings emerge and what they imply for deployment.

5.1 FREEZING POLICIES AND TRANSFERABILITY

As reported in Table 4.1 and Figure 4.1, the choice of which layers to freeze has a substantial impact on synthetic-to-real generalization. Freezing the first three convolutional blocks (Stage 1–3) of ImageNet pretrained ResNet-50 provides almost the same accuracy as a fully trainable backbone, while showing lower epoch-to-epoch variance. In comparison, freezing too much of the network is detrimental to performance: the fully frozen backbone produced the weakest results, and even freezing up to Stage 4 caused a noticeable drop. These results indicate that the most effective strategy is to preserve early, general-purpose features while fine-tuning deeper layers to adapt to the target domain.

This behavior aligns with the hierarchical nature of CNN representations. Early convolutional layers capture low-level features such as edges and color blobs, which are the most transferable across datasets and domains. In contrast, deeper layers encode increasingly task-specific abstractions that require adaptation for new distributions [24]. Freezing Stages 1–3 therefore preserves generic filters

learned from ImageNet while allowing higher layers to recalibrate to the textures, lighting, and occlusion patterns of our synthetic renders.

Partial freezing also reduces the number of trainable parameters, which acts as a form of regularization. This can limit overfitting to synthetic-specific artifacts (e.g., clean edges or simplified shading) and improves optimization stability.

Practical implications. Our results suggest freezing the first convolutional blocks is a strong default policy when training on synthetic data. This strategy is recommended unless a large real dataset becomes available, in which case updating the entire network would likely be more effective.

5.2 FUSION STRATEGIES

As shown in Table 4.2 and Figure 4.2, the fusion mechanism strongly affects both synthetic validation accuracy and generalization to real images. Two clear patterns emerge. First, *score fusion by sum* reached the highest synthetic validation accuracy (0.9885) and tied with *Early (S5, conv)* on the *Good Conditions* subset with 100% accuracy. Second, *early fusion at deeper stages* (particularly Stage 5, and to some extent Stage 4) provided stronger robustness on challenging real conditions such as *Edges*, *Occlusion*, and *Interference*, where score-based aggregation was less performant.

Score fusion. Averaging class scores across views (*Score sum*) combines evidence from all cameras, so that errors from one viewpoint are compensated by correct predictions from others, which could explain its top synthetic accuracy [11]. This makes score-sum particularly effective on the clean *Good Conditions* subset. However, on stress subsets (*Occlusion*, *Edges*, or *Interference*), its performance dropped markedly (accuracy as low as 0.20–0.60), showing that the lack of feature-level interaction limits its ability to reason jointly across partially corrupted views.

Early fusion. In contrast, *Early fusion after Stage 5* consistently outperformed *Score sum* on stress subsets, achieving an accuracy of 0.80 on all of them. This suggests that merging feature maps only after they encode high-level, domain-invariant shape cues allows the model to integrate multi-view evidence more effectively across viewpoints. Our findings agree with Seeland and Mäder [18], who report

that fusing at the last convolutional stage is among the most robust strategies. By contrast, fusing too early (Stages 2–3 with conv adapters) collapsed almost entirely on real subsets (accuracy \approx 0) despite high synthetic validation scores (\sim 0.96) and even with the first three ResNet stages frozen. The low-level filters from ImageNet (edges, textures) when pooled across different camera views may produce noisy, spatially misaligned representations. This works on clean synthetic validation but fails on real photos where edges and textures are more varied, producing domain overfitting. Pooling-based shallow fusions (e.g., S3/S4 max) mitigated this somewhat, but remained less performant than deeper fusion.

Late fusion. *Late fusion (FC)* produced balanced but not dominant results, with accuracy=0.80 on *Interference* and 0.90 on *Good Conditions*, showing that it keeps view-specific information at the cost of a heavier classifier head. Pooling embeddings directly (*Late max*) was consistently weaker, suggesting that simple reduction of embeddings loses important detail.

Practical implications. For deployment, two viable defaults emerge. *Score-sum* remains attractive for its simplicity, zero architectural cost, and strong performance on clean views, making it a safe baseline. However, in scenarios with frequent occlusions or clutter in the capture box, *early fusion at deep stages* (e.g., Stage 5 in ResNet-50) offers superior robustness with a moderate parameter footprint. Shallow early fusion should be avoided, as it systematically fails to generalize under domain shift.

5.3 WEIGHT SHARING SCHEMES

Table 4.3 and Figure 4.3 clearly show that the model with shared weights across all views achieved the best performance, the partially-shared variant came second, and the unshared configuration performed worst. This indicates that the more the weights are shared, the better the transfer to real test data.

A likely explanation is that sharing weights forces the network to learn features that generalize across all views, avoiding overfitting to view-specific artifacts present in the synthetic training data. Because gradient updates are aggregated across all views, the effect is similar to an intrinsic form of regularization: view-specific biases are averaged out, leading to more robust representations. Sharing

also reduces the number of trainable parameters, which lowers the model’s capacity to memorize view-specific details that do not transfer to the real domain. Conversely, the unshared configuration can over-specialize to synthetic patterns of each view and fails to generalize to real inputs.

Our observation that the *all-shared* model performs best aligns with the conclusions of Su *et al.* [20], who introduced a MVCNN that shares convolutional weights across different view branches and found it outperformed separate per-view feature extractors.

It is worth noting that the *first-4-shared* configuration was motivated by the intuition that the top view differs most from the four lateral views. In practice, however, this assumption did not translate into improved performance. The random rotations and physics-based placement in the synthetic dataset meant that the top camera often captured object poses overlapping with those seen from the sides, reducing its distinctiveness. As a result, a fully shared encoder was still able to extract features valid across all views, while partial sharing added capacity without a clear benefit.

Practical implications. For future experiments and deployment, fully shared weight architectures should be the default choice, unless there is strong evidence that views are highly heterogeneous or require view-specific specialization.

5.4 BACKBONE FAMILIES

As reported in Table 4.4 and Figure 4.4, ConvNeXt-Small and Swin-Tiny achieved the best overall performance on real test data, consistently reaching perfect accuracy across all subsets. ConvNeXt-Tiny and EfficientNet-B0 also transferred well, with only minor drops in more challenging conditions. In contrast, ResNet-152 and ViT-B/16, despite high synthetic validation accuracy, failed to generalize and severely underperformed on real subsets, indicating strong overfitting to the synthetic distribution.

These results show that moderate-capacity backbones with appropriate inductive structure generalized best from synthetic to real data, while very large models or architectures with weaker inductive biases collapsed under domain shift. EfficientNet-B0 is particularly notable: despite being the smallest backbone tested,

it maintained strong performance across real conditions, highlighting its efficiency and making it a viable choice when resources are constrained.

These results may not hold in all settings, however. With substantially larger or more diverse datasets, the higher-capacity models could outperform smaller ones. Likewise, in the target application with thousands of classes, smaller backbones may not be able to capture subtle distinctions, and deeper architectures could become necessary.

Practical implications. In this setting, ConvNeXt-Small offers both high real-world accuracy and acceptable resource usage. Swin-Tiny provides a strong alternative with slightly lower efficiency. EfficientNet-B0 stands out as a lightweight option. For scaling, the ConvNeXt-family architectures appear best suited based on the current findings.

5.5 CALIBRATION

As shown in Table 4.6, label smoothing with $\varepsilon=0.2$ worsened calibration on the real subsets. In fact, both ECE and NLL increased and no accuracy gains were observed (Table 4.5). On the other hand, temperature scaling tuned on the synthetic set reduced NLL on every real subset and lowered ECE for *Good* and *Partial occlusion*. For *Edges* and *Interference*, though, ECE did not improve and in some cases became worse (Table 4.6). Accuracy stayed unchanged as expected.

The smoothing factor of $\varepsilon=0.2$ was too strong for transfer across domains. It encouraged the network to output probability distributions that were too soft relative to the real data, leading to worse calibration.

Temperature scaling, on the other hand, worked as expected as a global rescaling. It made confidence scores better aligned overall (as seen in NLL) but could not handle the large variation across the different real-world conditions posed by *Edges* and *Interference* sets.

Because the evaluation subsets contained only 5–10 samples each, the resulting ECE estimates may be unreliable. In particular, the very low ECE on the *Good* set may be explained by the fact that all 10 samples were correctly classified.

Practical implications. Do not use label smoothing with $\varepsilon=0.2$ in this setting. Temperature scaling is safe to apply, mainly for its consistent NLL gains. For

deployment, calibration should be done on a small real set if possible, since a temperature parameter tuned only on synthetic validation data may not generalize across real-world conditions.

5.6 ROBUSTNESS & RELIABILITY

As shown in Table 4.7 and Figure 4.10, the ranking of models stayed the same across random seeds. The *Score-sum* configuration reached stable results with almost no variance, while the *Early* fusion setup failed consistently.

Practical implications. For these settings, model selection with only one run is often enough: the best design remains the best under new seeds in both synthetic and real validation.

5.7 LIMITATIONS

While the ablations in this thesis provide useful insights, several limitations constrain the scope of the conclusions.

First, the study was limited to *80 classes* with 750 variations each. This scale was chosen as a practical compromise: a larger dataset closer to the target application of $\sim 10,000$ classes would have required prohibitive rendering and training time, making it impossible to run the extensive ablation studies presented here. By restricting the class count, it was feasible to systematically evaluate freezing policies, fusion mechanisms, weight sharing, and backbone families within the available resources. Nevertheless, this scale difference means that the performance observed here may not directly translate to the final system.

Second, the evaluation sets contained photos of a single real physical object, meaning that only one class was represented in real data. This prevents assessing how the model would behave on other classes captured in real conditions.

These limitations emphasize that the findings should be interpreted as an exploratory baseline, establishing a foundation for future work at larger scale and with more diverse real-world evaluation data.

5.8 FUTURE WORK

Building on the limitations described in §5.7, several directions emerge for future work.

First, the most urgent step is to scale up the dataset by expanding the class set to thousands with additional CAD models.

Second, more real-world evaluation data is needed. The current experiments relied on synthetic renders for training and tested on very small real sets, with only one physical object available. A next step is to collect several real samples per class to better measure transfer from synthetic to real.

Third, further architectural exploration could be carried out. While this thesis compared a range of backbones and fusion strategies, newer lightweight models or alternative fusion mechanisms may offer improved accuracy. Testing such models on an expanded dataset would help confirm which configurations remain competitive when the problem grows in scale.

Finally, beyond accuracy, factors such as inference latency, memory footprint, and the ability to retrain as new parts are introduced will become critical in practice. Designing a pipeline that can incorporate new classes easily would enhance the system's adaptability.

These efforts would help transform the baseline established here into a system that can support the full industrial scenario.



Conclusions

This thesis addressed the problem of automated object classification in an industrial pre-kitting scenario, where thousands of distinct parts must be recognized reliably and at scale. The system is based on a five-camera imaging box, with the central challenge being that training must rely exclusively on synthetic images generated from CAD models. This creates a domain gap between synthetic training data and real deployment conditions.

To tackle this problem, we constructed a synthetic multi-view dataset in Blender and implemented a modular multi-view CNN pipeline with ImageNet-pretrained backbones.

6.1 CONTRIBUTIONS

The thesis makes several contributions to the study of synthetic-to-real transfer in multi-view classification. First, it introduced a *synthetic dataset* of five fixed viewpoints, closely mirroring the target imaging setup and incorporating controlled randomization to enhance diversity. On top of this dataset, a modular *multi-view CNN* pipeline was implemented, combining ImageNet-pretrained encoders with flexible fusion modules. Through systematic *ablation studies*, the work compared different freezing policies, fusion strategies, weight-sharing schemes, and backbone families, highlighting the configurations that most improved generalization to real captures. Complementary analyses of calibration and run-to-run consistency further strengthened the conclusions, confirming the stability of the selected

approaches. Finally, the thesis distilled these empirical findings into *practical guidelines* for building reliable synthetic-to-real classifiers and scaling them toward larger, more demanding industrial scenarios.

6.2 OUTLOOK

While the current study establishes a strong baseline, several directions remain open. Scaling the dataset to thousands of classes and collecting more real-world evaluation data will be essential to validate performance in production. Further exploration of lightweight backbones and alternative fusion mechanisms may yield better accuracy-efficiency trade-offs. Finally, developing a pipeline that can incorporate new parts seamlessly will be critical for long-term deployment in industrial settings.

In conclusion, this thesis demonstrates that a carefully designed multi-view CNN, trained entirely on synthetic renders, can achieve robust performance on real images in a challenging industrial scenario. The methods and findings presented here provide a foundation for future extensions and a practical guide for scaling toward the full production system.

References

- [1] Mona Alzahrani et al. "Deep models for multi-view 3D object recognition: a review". In: *Artificial Intelligence Review* 57.12 (Oct. 2024). ISSN: 1573-7462. DOI: 10.1007/s10462-024-10941-w. URL: <http://dx.doi.org/10.1007/s10462-024-10941-w>.
- [2] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (2021), p. 53. DOI: 10.1186/s40537-021-00444-8.
- [3] Blender Foundation. *Cycles Introduction*. <https://docs.blender.org/manual/en/4.5/render/cycles/introduction.html>. Accessed: 2025-08-19.
- [4] Blender Foundation. *Eevee Introduction*. <https://docs.blender.org/manual/en/4.5/render/eevee/introduction.html>. Accessed: 2025-08-19.
- [5] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: <http://www.blender.org>.
- [6] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [7] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: <https://arxiv.org/abs/2010.11929>.
- [8] Chuan Guo et al. "On Calibration of Modern Neural Networks". In: *CoRR* abs/1706.04599 (2017). arXiv: 1706.04599. URL: <http://arxiv.org/abs/1706.04599>.

- [9] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
- [10] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: <http://arxiv.org/abs/1502.03167>.
- [11] J. Kittler. “Combining classifiers: A theoretical framework”. In: *Pattern Analysis and Applications* 1.1 (Mar. 1998), pp. 18–27. ISSN: 1433-755X. DOI: 10.1007/BF01238023. URL: <https://doi.org/10.1007/BF01238023>.
- [12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: <https://doi.org/10.1145/3065386>.
- [13] Anders Krogh and John A. Hertz. “A simple weight decay can improve generalization”. In: *Proceedings of the 5th International Conference on Neural Information Processing Systems*. NIPS’91. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, pp. 950–957. ISBN: 1558602224.
- [14] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.
- [15] Ze Liu et al. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”. In: *CoRR* abs/2103.14030 (2021). arXiv: 2103.14030. URL: <https://arxiv.org/abs/2103.14030>.
- [16] Zhuang Liu et al. “A ConvNet for the 2020s”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11966–11976. DOI: 10.1109/CVPR52688.2022.01167.
- [17] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [18] Marco Seeland and Patrick Mäder. “Multi-view classification with convolutional neural networks”. In: *PLOS ONE* 16.1 (Jan. 2021), pp. 1–17. DOI: 10.1371/journal.pone.0245230. URL: <https://doi.org/10.1371/journal.pone.0245230>.

- [19] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [20] Hang Su et al. “Multi-view Convolutional Neural Networks for 3D Shape Recognition”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 945–953. DOI: 10.1109/ICCV.2015.114.
- [21] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.
- [22] Mingxing Tan and Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: <http://arxiv.org/abs/1905.11946>.
- [23] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.
- [24] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792. URL: <http://arxiv.org/abs/1411.1792>.
- [25] Qingnan Zhou and Alec Jacobson. “Thing10K: A Dataset of 10, 000 3D-Printing Models”. In: *CoRR* abs/1605.04797 (2016). arXiv: 1605.04797. URL: <http://arxiv.org/abs/1605.04797>.

Acknowledgments

I would like to express my gratitude
to Prof. Stefano Ghidoni
for his supervision.

I am especially thankful to Fabio Scarpa,
for his support, feedback,
and valuable guidance during my internship
and throughout the preparation of this thesis.

I also wish to thank the entire M31 team,
together with my fellow interns
and Moones Mobarki,
whose support made this experience
more enriching.