YOLO-Based Detection of Textile Care Symbols

Athul Shibu Kurian
The University of Texas at Arlington
Arlington, TX
axs5448@mavs.uta.edu

Divyang Vinodbhai Patel
The University of Texas at Arlington
Arlington, TX
dvp0564@mavs.uta.edu

Abstract

This project implements automated detection and classification of textile care symbols. We trained and benchmarked object detection models based on the YOLO (You Only Look Once) architecture. Specifically, we tested Ultralytics' YOLOv5m, YOLOv8m, and YOLOv11m, alongside legacy Darknet-based implementations of YOLOv3 and YOLOv4. Each model was trained on a custom-labeled dataset of textile care symbols, with hyperparameter optimization and data augmentation applied to enhance generalization performance. Model performance was evaluated using the mean Average Precision at IoU threshold 0.5 (mAP@0.5). Our evaluation shows that modern YOLO variants, particularly YOLOv11m, significantly outperform earlier versions, achieving a peak mAP@0.5 of 90.9%.

1. Introduction

Care symbols, also called laundry symbols, are pictograms or graphical symbols that are found in textile care labels. These symbols inform the consumer of the manufacturer's recommended instructions to wash, clean, or otherwise care for the textile product. Many care labels feature these symbols alone, without providing any additional written instructions for care. Some of the symbols may be confusing or may be difficult to interpret for the average consumer. Hence, a tool that automatically recognizes these symbols would be beneficial.

2. Problem Statement

The current standard for textile care symbols is ISO 3758:2023, developed by GINETEX, the international association for textile care labelling. The full set of symbols is organized into five broad categories: Washing, Bleaching, Drying, Ironing, and Professional Care. There are multiple symbols under each category.

Figure 1: Care symbol categories used in textile labeling [1].

These symbols are always monochromatic and use relatively simple geometric shapes to represent different care instructions. Additional modifications such as dots, lines, or crosses indicate specific requirements, such as temperature settings or restrictions on certain cleaning methods. For example, a symbol with a crossed-out triangle means bleaching is not allowed, while dots inside an ironing symbol indicate the maximum recommended ironing temperature.

In addition, a single textile care label usually contains multiple care symbols each representing a certain aspect of care. These symbols are typically small and closely spaced, making them challenging to detect individually. Moreover, the visual similarity between certain symbols, such as those indicating different temperatures or cleaning methods, can pose a challenge for recognition systems, requiring models to learn fine-grained distinctions to avoid misclassification.

Figure 2: A care label with four different care symbols.

There are currently more than forty different care symbols in total as defined by the ISO 3758:2023 standard. The model developed in this project detects and classifies 14 of those symbols.

Figure 3: The set of 14 target care symbols [2].

3. Related Work

Care label analysis presents unique challenges distinct from general object detection and unrelated to traditional OCR. While typical document or packaging understanding tasks often involve printed or digital text, care labels rely heavily on standardized pictographic symbols that communicate textile care instructions.

Kralicek et al. [3] were among the first to systematically address care label recognition, introducing a dataset and a domain-specific pipeline that combined scene text detection with symbol classification. Although their method involved text processing, it also emphasized the symbolic modality, treating care pictograms as visual units requiring detection and semantic interpretation. Their approach used weakly supervised training and bounding box merging techniques to manage clustered layouts and partially occluded symbols. Notably, the work highlighted the redundancy between symbols and text**,** though in the current context, symbol recognition alone is sufficient and often more robust when text is faded or absent.

Symbol detection on care labels shares traits with graphic symbol recognition and small object detection, but differs in its reliance on strict visual standardization and tight spatial arrangements. These factors motivate the use of specialized detectors capable of preserving fine-grained spatial resolution, avoiding the reliance on textual co-occurrence, and handling dense layouts with minimal false suppression, all of which are central to the current work.

4. Problem Solution

Since this is both an object detection and classification problem, we selected a model based on the YOLO (You Only Look Once) architecture. YOLO is a single-stage object detector that performs object localization and classification in one forward pass. Care labels often contain several small, closely packed symbols that must be identified both individually and simultaneously. YOLO is particularly well-suited for this task because it is designed to detect multiple objects in a single image efficiently, regardless of their location or size.

4.1. Dataset

The dataset used in this project initially consisted of 384 color images of varying resolutions. These images were compiled by combining an existing labeled dataset [4] with additional images collected from various internet sources. While the original dataset provided only image-level class labels, indicating which care symbols were present, it lacked spatial annotations specifying where each symbol appeared within the image. To enable object detection, we manually annotated each image using the YOLO format, assigning precise bounding boxes and corresponding class labels to every visible care symbol. Because most images contained multiple symbols, several bounding boxes were typically required per image. In many cases, the symbols were closely packed together, resulting in overlap between bounding boxes.

This foundational dataset served as the basis for further augmentation and training of the care symbol detection model.The data was labeled into 14 essential classes, each corresponding to one of the 14 care label symbols selected for this project: 30C, 40C, DN_bleach, DN_dry_clean, DN_iron, DN_tumble_dry, DN_wash, hand_wash, iron_low, iron_medium, non_chlorine_bleach, normal_dry_clean_solvents, tumble_dry_low, and tumble_dry_medium. Some images also contained additional care symbols that did not belong to these 14 classes. During the annotation process, these extra symbols were temporarily labeled as "other" to preserve their bounding box information. However, in the final training data, all "other" annotations were masked out and excluded from the learning process. In total, this annotation process resulted in 1,975 labeled instances across the 14 classes. The initial version of the labelled dataset also exhibited significant class imbalance, with some classes appearing far more frequently than others.

Figure 4: Sample images from the dataset, with the bounding boxes highlighted [4]

4.2. Data Preprocessing

To ensure consistency, all images in the dataset were preprocessed to a fixed resolution of 640 × 640 pixels. This resizing was achieved by first shrinking the original image proportionally to fit within a 640 × 640 frame, preserving the aspect ratio to avoid distortion. The remaining empty space was then padded with black borders to fill the frame completely, resulting in uniformly sized square images. In addition to resizing, all images were converted to grayscale to reduce input dimensionality and focus the model on the structural and geometric features of the care symbols, which are typically monochromatic by design. This preprocessing step also helped standardize the appearance of the dataset, removing color-related variability that is irrelevant to symbol interpretation, and ultimately facilitated more efficient model training and generalization.

To remove symbols not included in the 14 target classes, we applied a white masking step during preprocessing. Certain images contained additional care symbols that were not part of the selected label set; although these were initially annotated as "other", they were not intended for training. Instead of discarding entire images that contained these irrelevant symbols, we preserved the useful portions by drawing solid white rectangles over the bounding box regions corresponding to "other" labels. This effectively erased the non-target symbols while retaining the rest of the annotated content. By masking these areas rather than cropping or removing images, we maintained valuable training data without introducing noise from irrelevant classes. This step was crucial in ensuring the model trained only on the intended care symbol classes and was not distracted by unwanted or ambiguous visual elements.

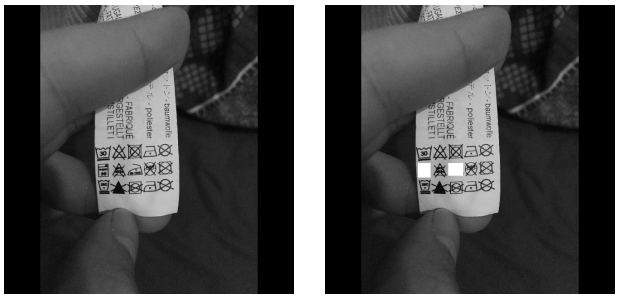

Figure 5: An image before and after applying white masks.

To enhance the robustness of the object detection model and improve generalization, the training dataset was augmented using a variety of image transformation techniques. Each training image was subjected to random rotations within a range of -15 to +15 degrees to simulate natural variations in camera orientation or label placement. Additionally, more substantial rotations of 90 degrees clockwise, counterclockwise, and 180 degrees (upside down) were applied to introduce greater geometric diversity. To address the issue of class imbalance in the dataset where some care symbol classes appeared much less frequently than others, a targeted sampling strategy was employed. Specifically, images containing underrepresented classes were selected more frequently during the augmentation process to artificially boost their representation in the training set. This oversampling approach ensured that the model received sufficient training examples for each class, thereby reducing bias toward more common symbols and improving overall classification performance across all 14 target categories. After applying all augmentations and sampling strategies, the final training set consisted of 1,227 images and 6,224 labeled instances across all classes.

4.3. Method

The YOLO models used in this project follow a unified architecture that performs object detection in a single forward pass through the network. These models are composed of a convolutional backbone, typically pretrained on a large-scale dataset, which is responsible for extracting low- to high-level spatial features from the input image. These features include edges, shapes, textures, and patterns that help the model distinguish between different care label symbols. On top of the backbone sits the detection head, which interprets the extracted features to generate bounding box predictions and class scores. YOLO's architecture is fully convolutional, allowing it to process images of varying sizes efficiently while maintaining spatial awareness.

During inference, the input image is divided into a fixed grid, and each grid cell is tasked with predicting a

predefined number of bounding boxes. For each bounding box, the model outputs the coordinates (center x, center y, width, height) relative to the grid cell, an objectness score representing the confidence that an object is present, and a set of class probabilities for all 14 care label categories. To improve flexibility in detecting objects of different shapes and sizes, YOLO uses multiple anchor boxes per grid cell, enabling it to predict overlapping or variably sized symbols more effectively. After the full image is processed, all bounding box predictions are aggregated, and non-maximum suppression (NMS) is applied to eliminate redundant or low-confidence detections, retaining only the most probable predictions per class. This process yields the final set of bounding boxes with class assignments and confidence scores for each detected care symbol in the image.

The YOLO loss function is composed of multiple components that reflect the different objectives of object detection. It simultaneously considers the accuracy of bounding box predictions, the confidence that an object is present in each location, and the correctness of the predicted class label. Each predicted bounding box is evaluated based on how well it aligns with a corresponding ground truth box in terms of position and size. The objectness component ensures that the model learns to distinguish between grid cells that contain objects and those that do not, while the classification component evaluates how accurately the model assigns class labels to detected objects. This structure will allow the model to learn both the spatial arrangement and the class labels of multiple care symbols that may appear together in a single image. Modern versions of YOLO use more sophisticated loss functions along with other improvements.

4.4. Model Architectures

We experimented with different YOLO implementations and versions to find the optimal model. The tested implementations were YOLOv11m, YOLOv8m, and YOLOv5m from Ultralytics [6], and AlexeyAB's YOLOv3, and YOLOv4 [7].

The YOLOv11m model uses a custom backbone designed by Ultralytics and consists of 231 layers. It begins with a series of convolutional layers that progressively expand the input from 3 to 512 channels, followed by multiple C3k2 blocks that maintain or increase depth. Key components include an SPPF module and a C2PSA attention block, both operating on 512-channel features. The network performs up-sampling and concatenation to fuse features at different scales, followed by additional C3k2 blocks. The detection head takes feature maps of 256, 512, and 512 channels and outputs predictions for each class. The full model has 20,063,802 parameters, 20,063,786 gradients, and runs at 68.2 GFLOPs.

The YOLOv8m model also uses a custom Ultralytics backbone but follows a slightly more compact architecture with 169 layers compared to YOLOv11m's 231. It begins with convolutional layers that expand the input from 3 to 576 channels through progressive stages, each followed by C2f blocks: modular structures used for efficient feature extraction and reuse. Unlike YOLOv11's use of C3k2 and C2PSA modules, YOLOv8m relies solely on C2f blocks throughout its backbone and neck. An SPPF (Spatial Pyramid Pooling Fast) module is included at the deepest point to aggregate multi-scale context. The model then up-samples and concatenates feature maps from earlier layers, followed by additional C2f blocks to refine features. The detection head receives input from three scales with 192, 384, and 576 channels, respectively, and outputs predictions for all 14 classes. YOLOv8 is slightly heavier than YOLOv11m in terms of computational cost, with 25,864,426 parameters and 79.1 GFLOPs.

The YOLOv5m model by Ultralytics consists of 197 layers and uses a CSP-based architecture, employing a series of convolutional and C3 (Cross Stage Partial) blocks for feature extraction, gradually expanding the input from 3 to 768 channels. It includes an SPPF module at the deepest point to aggregate multi-scale features and uses up-sampling and concatenation to fuse features across scales. These are refined with additional C3 blocks before reaching the detection head, which operates on feature maps of 192, 384, and 768 channels to produce outputs for 14 classes. YOLOv5m has 25,073,242 parameters and runs at 64.4 GFLOPs, making it more lightweight than YOLOv8 and YOLOv11m.

AlexeyAB's YOLOv3 is built on the Darknet53 backbone, optimized for real-time inference. It uses successive convolutional layers with shortcut connections (residual blocks), progressing from 416×416×3 inputs through increasingly deep feature maps down to 13×13×1024 in the final stages. YOLOv3 performs multi-scale detection at 52×52, 26×26, and 13×13 resolutions, improving small and large object recognition. Detection heads are connected via up-sampling and route layers, ensuring rich spatial context. The network outputs predictions at three scales, using anchor boxes with losses normalized for IoU, objectness, and classification. It contains roughly 65 million parameters and 106 layers.

AlexeyAB's YOLOv4 improves upon YOLOv3 by incorporating the CSPDarknet53 backbone, which enhances gradient flow and reduces computational cost without sacrificing accuracy. This version adopts Cross Stage Partial (CSP) connections within residual blocks to optimize feature reuse and network efficiency. Like YOLOv3, it operates on 416×416×3 input dimensions and performs multi-scale detection at 13×13, 26×26, and 52×52, facilitating robust detection of objects at varying sizes. YOLOv4 integrates additional modules such as Spatial Pyramid Pooling (SPP) and PANet for better aggregation of spatial and semantic information. The architecture uses upsampling and routing to combine

features across different levels, improving localization and classification precision. Detection heads rely on anchor boxes with loss functions tuned for objectness, IoU, and class probability. YOLOv4 contains approximately 64 million parameters and over 160 layers, making it significantly deeper and more powerful than its predecessor while still maintaining high inference speed.

4.5. Training

For the Ultralytics models, the training function features a wide range of augmentations such as random horizontal flips, translation, random erasing, Gaussian blur, median blur, CLAHE (Contrast Limited Adaptive Histogram Equalization), and grayscale conversion. In the first round of training, all augmentations were disabled to establish a baseline. The model was trained on 1,227 images containing 6,244 annotated instances, using a batch size of 64 for 200 epochs. Input images were resized to 640×640 and converted to 3-channel format by duplicating grayscale values across channels. The optimizer used was AdamW with a learning rate of 0.000556 and momentum of 0.9. A total of 643 out of 649 layers were transferred from pretrained weights, and the final detection head's convolutional weights were frozen during training. The training was conducted on an Nvidia A100-SXM4-40GB GPU.

In subsequent rounds of training, we experimented with different augmentations to get the best results. Each round tested various combinations of transformations to find the most effective setup for improving generalization. In the final round, the best-performing augmentation configuration was selected based on empirical results. This included horizontal flipping with 50% probability, image translation up to 10% of the original dimensions, and random erasing applied with 40% probability. Mosaic augmentation was enabled throughout most of the training but was turned off for the last 10 epochs to allow the model to fine-tune on natural image layouts. Scale jittering was applied with a scaling factor of 0.5. Additional augmentations included Gaussian blur and median blur, each with a 1% chance and a blur kernel size between 3×3 and 7×7. Grayscale conversion was also applied randomly with a 1% chance, and CLAHE was used at 1% probability with clip limits between 1.0 and 4.0 and an 8×8 tile grid. Other transformations such as vertical flipping, rotation, shear, HSV color jittering, mixup, *2*, copy-paste, and perspective distortion were all disabled. The final round also used a batch size of 64 and trained the model for 200 epochs with the same optimizer settings, pretrained weights, and AMP configuration.

For both YOLOv3 and YOLOv4, we used the original implementations provided by Alexey Bochkovskiy from his official Darknet GitHub repository [7]. The models were trained for 200 epochs on an NVIDIA A100-SXM4-40GB GPU with no augmentations applied during training to ensure a controlled evaluation of the architectures. The optimizer used was AdamW with a learning rate of 0.0005 and a momentum of 0.9. Training was performed with a batch size of 64, and the models were initialized with pretrained weights from darknet53.conv.74, enabling more stable convergence.

To visualize the model's predictions, we developed a Python script that processes the output bounding boxes from YOLO and overlays them directly onto the original images. The script extracts the coordinates, class IDs, and confidence scores from the model's output and draws red bounding boxes around each detected care symbol. For clarity, a label is added above each box showing the class ID, using a white font over a red background to ensure visibility against varying image content. The function handles all box drawing using the Python Imaging Library and ensures that boxes are rendered cleanly and consistently. In addition, the script generates a legend that is overlaid on the bottom of the image. This legend displays a mapping of detected class IDs to their corresponding class names using the model's internal label map. The legend is rendered on a semi-transparent black background to avoid obstructing image content while maintaining readability. This post-processing step made it easier to inspect the detections and verify model predictions.

4.4. Evaluation metric

To assess the performance of our object detection model, we use two key metrics: Intersection over Union (IoU) and mean Average Precision (mAP).

Intersection over Union (IoU) is a fundamental metric used to evaluate how well a predicted bounding box aligns with the corresponding ground truth bounding box. It is defined as the ratio between the area of overlap and the area of union between the predicted box B_p and the ground truth box B_{gt}.

$$IoU = \frac{|B_p \cap B_{gt}|}{|B_P \cup B_{gt}|} \quad (1)$$

An IoU value of 1.0 indicates perfect overlap, while a value of 0.0 means no overlap at all. In object detection tasks, a prediction is considered a true positive if its IoU with a ground truth box exceeds a predefined threshold. Otherwise, it is treated as a false positive.

To summarize model performance across all classes and predictions, we use mean Average Precision (mAP) as the primary evaluation metric. Specifically, we report mAP@0.5, which measures the average precision for each class at an IoU threshold of 0.5. The precision-recall curve is computed for each class by varying the confidence threshold of the model's predictions. Average Precision (AP) is then calculated as the area under this precision-recall curve for each class, and mAP is obtained by averaging the AP values across all 14 symbol classes.

This metric is particularly effective because it captures both the model's ability to localize objects accurately (through bounding box alignment) and to correctly classify them. A high mAP@0.5 indicates that the model is consistently making precise detections with strong class confidence and spatial accuracy.

Before computing IoU and mAP, the model's raw predictions are filtered using Non-Maximum Suppression (NMS) to remove redundant boxes. NMS keeps the highest-confidence bounding box for each object and suppresses others that have significant overlap with it. In our setup, the IoU threshold for suppression was set to 0.7, meaning boxes with more than 70% overlap with a higher-scoring box were discarded. This step ensures that only one box is retained per object, reducing duplicates and improving detection clarity.

4.5. Results

Under baseline training conditions with all augmentations disabled, the Ultralytics models demonstrated strong detection performance on the validation set, which contained a total of 448 annotated instances. YOLOv11 achieved the highest mAP@0.5 score of 85.3%, followed closely by YOLOv8 at 83.4%, and YOLOv5 at 82.5.% With the opti mal augmentation setup applied, previously tuned through experimentation, the Ultralytics models demonstrated improved detection accuracy on the 448-annotation validation set. YOLOv11 achieved the highest mAP@0.5 at 90.9%, marking the best score among all tested models. YOLOv8 followed with 88.4%, and YOLOv5 reached 86.7%, both showing consistent gains over their baseline runs. The YOLOv4 and v3 models showed lower performance, with the YOLOv4 achieving an mAP of 75.8% and YOLOv3 reaching 72.4% on the validation set.

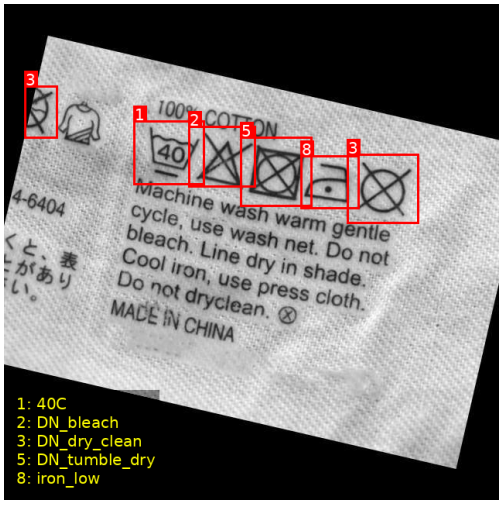

Figure 6: Predicted bounding boxes and class labels on a validation image containing multiple care symbols.

Table 1: Highest mAP@0.5 achieved by each model on the validation set

Model	mAP@0.5
Ultralytics YOLOv11m	0.909
Ultralytics YOLOv8m	0.884
Ultralytics YOLOv5m	0.867
AlexeyAB's YOLOv4	0.758
AlexeyAB's YOLOv3	0.724

The per-class performance of the best model, YOLOv11, as summarized in the results table, highlights a strong overall detection accuracy with a mean Average Precision at IoU threshold 0.5 (mAP@0.5) of 90.9%, a precision of 82.3%, and a recall of 86.0% across all 14 care symbol classes. The highest-performing classes included DN_tumble_dry, DN_bleach, DN_iron, and DN_dry_clean, each achieving mAP@0.5 scores exceeding 97%, with strong recall values, reaching 1.0 for DN_iron and 0.987 for DN_tumble_dry. Symbols like normal_dry_clean_solvents, iron_low, and non_chlorine_bleach, also performed well, showing mAP scores around 96–97%, paired with high precision and recall. Moderate performance was seen for DN_wash and hand_wash, with mAP@0.5 values of 92.2% and 92.9% respectively. The weakest results were observed in tumble_dry_medium, 40C, iron_medium, and 30C, which had lower precision and recall metrics, especially tumble_dry_medium, with a recall of 0.565 and mAP@0.5 of 71.2%.

Table 2: Per-class detection performance of YOLOv11m on the validation set, measured by precision, recall, and mAP@0.5 across 14 care symbol classes.

Class	Precision	Recall	mAP@0.5
DN_tumble_dry	0.922	0.987	0.99
DN_bleach	0.942	0.956	0.984
DN_iron	0.758	1.0	0.978
DN_dry_clean	0.945	0.974	0.975
normal_dry_clean	0.844	0.964	0.969
non_chlorine_bleach	0.868	0.958	0.962
iron_low	0.872	0.937	0.958
hand_wash	0.945	0.833	0.929
DN_wash	0.924	0.717	0.922
tumble_dry_low	0.699	0.905	0.904
30C	0.633	0.837	0.835
iron_medium	0.748	0.729	0.835
40C	0.655	0.683	0.768
tumble_dry_medium	0.772	0.565	0.712
All classes	0.823	0.86	0.909

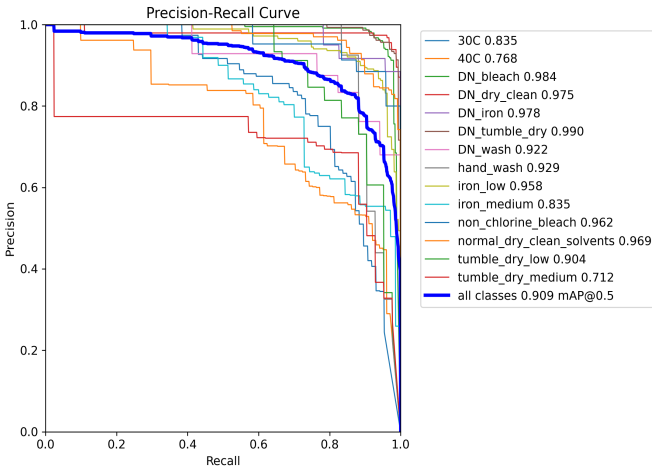

Figure 7: Precision-recall curves for each care symbol class, along with the overall curve (bold blue) for YOLOv11.

5. Conclusion

The evaluation of various YOLO models on the care symbol recognition task revealed significant differences in performance across architectures and training configurations. In baseline training, where augmentations were disabled to establish a controlled comparison, Ultralytics' YOLOv11 achieved an mAP@0.5 of 85.3%, with YOLOv8 and YOLOv5 following at 83.4% and 82.5%, respectively. These results were already strong, given the absence of augmentation-driven regularization. However, with the full augmentation suite enabled, including geometric and pixel-level transformations, the models achieved notably higher scores: YOLOv11 reached a peak mAP@0.5 of 90.9%, YOLOv8 improved to 88.4%, and YOLOv5 climbed to 86.7%. This confirms that carefully tuned augmentations played a key role in enhancing the models' ability to generalize, especially given the limited dataset size and class imbalance in the 448-annotation validation set. Notably, the use of mosaic augmentation during most training was instrumental, likely due to its synthetic enlargement of spatial context, which helps models better recognize symbols with limited examples by mixing multiple training images into one. Disabling mosaic in the final epochs allowed the model to adapt to more realistic layouts, further refining performance.

The architectural differences across models also contributed meaningfully to the performance hierarchy. YOLOv11's strong results can be attributed to its deeper 231-layer architecture and the inclusion of specialized modules such as the SPPF and C2PSA blocks, which enhance spatial feature aggregation and attention. Its backbone, although custom-designed, closely follows Ultralytics' philosophy of modularity and deep feature reuse. YOLOv8, while heavier in terms of parameters and GFLOPs, uses C2f blocks for efficient feature encoding but lacks the attention mechanism found in YOLOv11, which may explain its slightly lower performance. YOLOv5, based on a CSP-inspired structure with C3 blocks and fewer layers (197 in total), also performed well but remained a step behind its successors, reflecting how architectural innovations like dynamic attention and deeper layer hierarchies contribute to performance gains. All three Ultralytics models employed similar training setups and benefited from transfer learning, but the combination of architecture depth, modular blocks, and augmentation compatibility gave YOLOv11 its edge.

In contrast, the legacy Darknet-based models, YOLOv3 and YOLOv4, showed comparatively lower performance under identical training constraints (no augmentations, 200 epochs, batch size 64). YOLOv4, leveraging the CSPDarknet53 backbone and additional enhancements like SPP and PANet, outperformed YOLOv3 with an mAP@0.5 of 75.8% versus 72.4%, but still lagged the more modern Ultralytics models. YOLOv3, built on the original Darknet53 backbone with 106 layers, demonstrated solid but outdated performance. Its results highlight the diminishing returns of deeper but less modular architectures lacking modern components like attention blocks, adaptive upsampling, or extensive data augmentation compatibility. While both YOLOv3 and YOLOv4 remain valuable for real-time detection tasks due to their relatively low inference times, their limited adaptability in this specialized domain underscores the impact of backbone design and training strategies on model success. Ultimately, YOLOv11's results show that contemporary architectural decisions, when combined with targeted augmentation, can significantly boost performance, even in domains with small and imbalanced datasets.

References

[1] GINETEX, "Care Symbols," GINETEX, Available: https://www.ginetex.net/gb/labelling/care-symbols.asp. Accessed: Feb. 27, 2025.

[2] Wikipedia contributors, "Laundry symbol," *Wikipedia, The Free Encyclopedia*, Available: https://en.wikipedia.org/wiki/Laundry_symbol. Accessed: Feb. 27, 2025.

[3] T. Kralicek and J. Matas, *Care Label Recognition*, in Proceedings of ICDAR, 2019. https://cmp.felk.cvut.cz/~matas/papers/kralicek-2019-care_labels-icdar.pdf

[4] songogong, "LaundryCoach Dataset," Roboflow Universe, Feb. 2024. Available: https://universe.roboflow.com/songogong/laundrycoach. Accessed: May 4, 2025.

[5] Redmon, Joseph, Santosh Kumar Divvala, Ross B. Girshick and Ali Farhadi. “You Only Look Once: Unified, Real-Time Object Detection.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 779-788

[6] Ultralytics. Models Supported by Ultralytics. Ultralytics Documentation. Available at: https://docs.ultralytics.com/models. Accessed 5 May 2025

[7] Bochkovskiy, A. AlexeyAB/darknet: YOLOv3 Implementation. GitHub repository. Available at: https://github.com/AlexeyAB/darknet