
TU Delft

# Product-ProtoNet

A simple architecture for classifying supermarket products, using just a few example images

Thesis Report by
Rick Dekker (4682548)

# Product-ProtoNet

## A simple architecture for classifying supermarket products, using just a few example images

by

Rick Dekker

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday March 8, 2024 at 14:00 AM.


| | | |
|---|---|---|
| Student number: | 4682548 | |
| Project duration: | June 6, 2023 – March 8, 2024 | |
| Thesis committee: | Prof. dr. ir. M. Wisse, | TU Delft, supervisor |
| | Dr. H. Ceasar, | TU Delft |
| | Dr. Ir. R. Sabzevari, | TU Delft |

Cover: Personal image; Albert in a test supermarket at AirLab

An electronic version of this thesis is available at http://repository.tudelft.nl/.


TU Delft

# Abstract

Airlab, a collaboration between TU Delft and Ahold Delhaize, is developing Albert, a robot tailored to work in a complex supermarket environment. Key to Albert is a product detection and classification module that tells it what products to grasp and where they are located in a shelf. Albert's existing YOLO-based product detector however has two significant issues: 1) It has noticeable positional biases and can not always identify products that are in a different place than usual; and 2) Adding new products without re-training the whole model is impossible. Especially in a dynamic supermarket environment with an ever-changing stock, the latter is a major issue.

To address the first problem, product localization and product classification will be split. This means that product class is independent of product location and locational biases will be less prevalent. The second problem will be the main focus of this paper. This problem is addressed through few-shot learning, which predicts similarity between query and target products. This simplifies adding new products to just supplying new target images. Few-shot learning also requires significantly less data to train on. In supermarkets with 300.000 different products, requiring only a few images per product is a major advantage. For this reason, this paper aims to deploy a few-shot model to classify products as either the target class or non-target class for Albert's picking task and defines the following research question: **"What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?"**

This paper first analyses the potential of using TRIDENT and P>M>F, two state-of-the-art few-shot models, for deployment on Albert, and evaluates them on the requirements of this paper. These are: 1) A minimum accuracy threshold of 90% on products classes seen and not seen during training; 2) A maximum inference time of 3.6 ms per image, to allow for real-time visual feedback that guides robot grasp control (visual servoing); and 3) A GPU memory usage below 4GB for compatibility with lower-end GPUs; Both TRIDENT and P>M>F pass the GPU memory requirement, but are unusable on a supermarket robot, as their high inference time makes them unsuited for visual servoing. P>M>F has a consistently 5%-10% higher accuracy than TRIDENT however, and its architecture allows for inference time optimization. This makes P>M>F the preferred model for Albert. However to work well, it still requires adjustments.

For this reason, this paper uses P>M>F's two key ideas to construct Product-ProtoNet, a new Albert-suitable few-shot model: 1) Using a good pre-trained feature extractor; and 2) Comparing query images to a set of classes and matching only to the likeliest. P>M>F uses a ProtoNet model for classification that essentially does this; Like ProtoNet, Product-ProtoNet constructs class prototypes from one or multiple examples of class images. Product-ProtoNet then uses a sigmoid classifier to predict if query images have the same class as those prototypes. It compares query images to a set of similar class prototypes (helper prototypes) and classifies them as the likeliest match. Product-ProtoNet uses a ViT pre-trained with DINO to extract image features. To bring down inference time, Product-ProtoNet computes product prototypes before deployment.

With an accuracy of 99.1% on product classes seen during training and 99.8% on novel classes in a realistic supermarket setting, an inference time of 2.89 ms and a memory usage lower than 4GB, Product-ProtoNet is the only model that passes all requirements of this paper. When deployed on Albert together with a YOLO product detector for product localization Product-ProtoNet successfully guides Albert to the right product in 97% of attempts. This means that real-life detection works well and that this implementation of Product-ProtoNet in Albert's perception pipeline is fast enough for visual servoing. This makes Product-ProtoNet the only *few-shot classifier that can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert.*

# Contents

# 1 Introduction

Supermarkets are busy places. Customers walk around to do their daily grocery shopping, products are being restocked, taken out of shelves and eventually sold at the cash register. Understandably, this is a hectic and ever-changing environment. Product stock may change, product packaging may change, aisles may at one moment be filled with people and fully empty at another.

AIRLab, a cooperation between Ahold Delhaize and TU Delft, has been tasked by Ahold Delhaize to produce a robot for exactly this very challenging environment. This robot, Albert, can recognize, pick and collect items from an online shopping list. Alberts' current product detection model (YOLO[41]-V6.3) can however not detect products it has never seen during training. Generally this is not a problem, but products' packaging may change and new products may be introduced to a supermarket. After all, changes in product package design are a positive purchase decision stimulant for a product [9] and producers might want to change their packaging regularly.

Re-training a model on 300,000 [29] diverse supermarket products for packaging updates or introductions of new products is a time-consuming task that generally requires huge amounts of data per product. For DarkNet for example, a framework that supports the training and testing of YOLO-V4, YOLO-V3 and YOLO-V2, 2000 images per class are recommended [2]. Few-shot models on the other hand need very few example images (shots) per class to effectively train with. As few-shot models learn to classify the similarity between query images and target classes, invariant of their actual class, adding new products is as simple as providing the model with a few new target images. This also eliminates the need to re-train an existing model. For this reason, this paper aims to deploy a few-shot model to classify products as target- or non-target product for Albert's picking task. Two few-shot models that look very promising are TRIDENT [49] and P>M>F [16].These state-of-the-art few-shot models achieve accuracies of respectively 96.0% and 98.4% on mini-ImageNet, a common few-shot dataset. To ensure safety, it is crucial for few-shot classifiers to not classify all input images as the most likely product class, which some classifiers do [16], especially when this input image is a human. Instead it should consistently classify this human as a non-product. To find the best few-shot classifier, this paper defines the following research question:

> **What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?**

Together with a YOLO-V6.3 product detector, which predicts the spatial location of products within an image, the proposed few-shot classifier will replace the existing fully YOLO-based detection and localization model that is currently on the robot. The current model predicts both product position and class and shows noticeable positional biases, as reported by people working with Albert. This bias easily occurs when products in a training dataset are not evenly spread trough a shelf (e.g. if cheap salami

is always on the bottom, bottom products are more often predicted to be cheap salami, independent from their real class). Utilizing a product detector for localization and a few-shot classifier for classification will decouple class from location and mitigate this locational bias. This paper specifically uses a YOLO-V6.3-nano-detector that has been trained on an zoom-augmented version of the SKU110-dataset [14] and has a mean average precision of 0.65 on a supermarket dataset made for our application [24], but any object detector should work.

With a well-working Albert that can accurately classify both familiar and novel products, Ahold Delhaize is interested in exploring how well robots can adapt to real life supermarket scenarios. As an added benefit, these robots can be used for picking and offering online orders directly in the supermarket, which would decrease the need for dedicated online distribution centers (HSC's [1]). Of course there also is the promising aspect that a robot that is good at picking products from shelves can easily be adapted to a shelf-stocking robot, as there are very little changes in core functioning. However, in order to achieve these objectives Albert first has to work well. To which end this paper focuses on finding *a few-shot classifier that identifies products in a supermarket environment, is able to detect non-classes, and meets the requirements of deployment on a robotic platform like Albert best*.

## 1.1. Research Question and Subquestions of this paper

This paper focuses on answering the following research question:

> **"What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?"**

To do so, first TRIDENT and P>M>F will be analysed to determine the optimal base classifier to work on a robot like Albert. Secondly, since few-shot classifiers operate under the assumption that the classes of query images always belong to a set of example classes, certain classifiers, such as P>M>F, simply select the most likely of the set as the designated query class [16]. This behavior is undesirable and unsafe. If P>M>F is chosen as the best or if either model does not meet Albert's requirements, a new model that can classify non-classes and meets Albert's requirements must be designed. Thirdly, the performance of the best model when deployed on Albert will be evaluated, as it is crucial to ensure that it can be used in real life. This yields the following sub-questions that will each be answered in their own chapter:

1. **Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?**
2. **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?**
3. **How well does this model work on Albert?**

## 1.2. Requirements

The best few-shot classification method heavily depends on what specifications this method should meet. Some important specifications this paper sets are:

1. **Inference time:** *The model's inference time should be as low as possible, but models with a high detection accuracy are preferred over those with a low inference time. To still have good closed-loop object tracking, this paper requires a maximum inference time of 3.6 ms per query image.*

   Albert uses visual predictions to update it's grasp and track products (visual servoing). This typically requires a detection rate of 30-100 Hz [24],[19],[60],[55]. This translates to an allowed inference time per detection of 10-33 ms. A YOLO-V6.3-nano model has an inference time of 1.3 ms on an NVIDIA Tesla-T4 GPU [26]. This means there will be 8.7-31.7 ms left to detect if the products seen are the target class or not. Assuming 9 product detections per camera image on average, the inference time per detected product is should be between 1-3.6 ms.

2. **Accuracy on seen products:** *The preferred few-shot method should have the highest possible accu-*

*racy in a realistic supermarket setting for product classes that are seen during training. As a design decision, this should be at least 90%.*

Papers that try to solve a similar problem in a similar setting, set a classification accuracy of 90% as an acceptable threshold [12]. Some others call an accuracy of 87.5% not enough [59]. In the end the acceptable accuracy of a model comes down to a design decision: What do customers and what does Ahold find an acceptable misdetection rate? In this paper an accuracy of 90% on will be deemed acceptable, but it is clear that higher accuracies are preferred.

Few-shot classifiers generally only evaluate their accuracy on classes not seen during training [34]. However for a model that will be used in a real supermarket, it is important that the product classes it has seen during training are also detected correctly. For this reason, this paper applies the 90% accuracy threshold to seen products as well.

Albert uses visual servoing with an algorithm that selects the class with maximum occurrence [24]. Which means that a product is correctly classified as long as the majority of detections has the correct class. To give an example: if a model manages to do three detections within 3.6 ms, with an accuracy of 75%, the chance that a majority of detections will be correct is 92%. With a base accuracy of 90%, this chance will become 99%. Generally this will mean that this paper chooses the fastest classifier with the highest accuracy.

Few-shot models are commonly evaluated in a few-shot setting [34], [51], [20]. This is a setting where all query images have the same class as all example images the model uses. In a realistic supermarket setting however, it is possible that not all products in a shelf are one of the example classes. To this end, this paper will evaluate models on the accuracy requirement in a **realistic supermarket setting**. A realistic supermarket setting is defined by this paper as a shelf with 10 products, very similar in brand, product type or color. Here a classifier has to distinguish 1 product from these 9 other products, without necessarily having information about all products. Arguably this is a very hard use-case, but as in supermarkets products of the same brand, color and product type often together, this seems more realistic and representative to Alberts' environment than evaluating classifiers on $n$ random classes, all of which a classifier has information about. Note that this realistic supermarket setting might not be fully representative of all real world scenarios and is mainly a metric to make evaluation of few-shot classifiers more comparable to real life. Real-world evaluation of the performance of classifiers when deployed on Albert is therefore still essential.

3. **Accuracy on unseen products:** *The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for novel product classes that are not seen during training. As a design decision, this should be at least 90%.*

   To ensure a comprehensive evaluation, this paper will use identical thresholds and measurement techniques to assess the accuracy of seen and unseen products. This will guarantee an accurate comparison between the two and give a realistic impression of the models' performance on both.

4. **Memory usage:** *As a design decision, inference GPU memory usage should be below 4GB.*

   Memory usage during inference is an important factor in how high- or low- end the hardware needed should be. Ideally the model should be able to be run on lower end GPU's that only have 4GB of Memory available, to keep the hardware requirements of the robot cheap and low.

# 2

# Related Work

In order to pick or create a well-working few shot classifier, it is important to first understand how few-shot classifiers work and how the few-shot problem can be defined. Few-shot learning tackles the problem of creating a generalized model from very few examples. Usually few-shot Learning is defined as **N-way, K-shot learning**, where **N** denotes the number of classes it tries to identify and **K** denotes the number of examples per class [51].

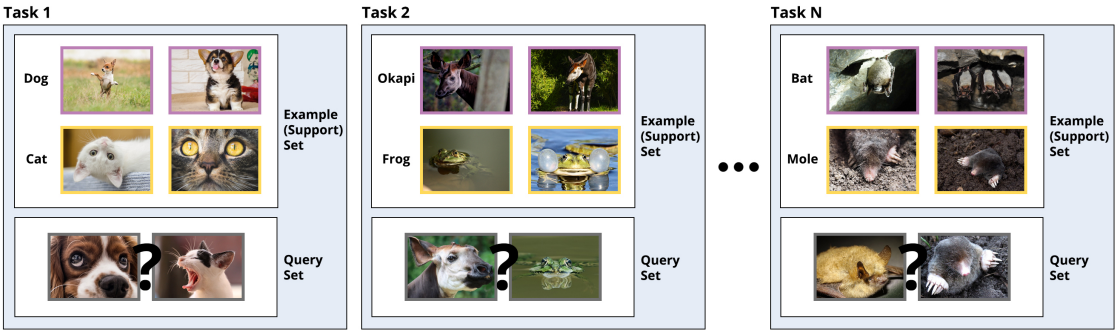


**Figure 2.1:** A figure representing the few-shot learning problem. Adapted from [34]. A model has to classify similarity between query images and example (or support) classes, independent of their actual class. It learns this by doing many few-shot tasks

Figure 2.1, based on [34], shows a typical few-shot classification problem. This particular problem classifies 2 classes with 2 examples per class, which makes it a 2-way, 2-shot problem. In this example, the first task a model has to do is to predict whether the images in the query set are *cats* or *dogs*. After this, it has to do a second task: predict whether a query image is an *okapi* or a *frog*. By doing many of these tasks, few-shot algorithms should be able to determine accurately to what example class a query image belongs, independent of its actual class. In that sense few-shot algorithms learn what makes images similar. This technique can be used for many different classes and also for classes that the model has not seen during training (unseen classes).

Few shot models are usually evaluated on classes not seen during training [34]. This means that a few-shot model trained on classes *cat*, *dog*, *okapi* and *frog* is evaluated on classes like *duck*, *shark* etc. to determine its accuracy on unseen classes, or unseen accuracy in short. This is the opposite of seen accuracy, where models are only evaluated on classes seen during training. Some papers however evaluate both seen and unseen accuracies of a model to evaluate how well a model generalizes from classes it has seen during training to classes it has not seen [37], [48], [61]. This paper will do the latter as good performance on both is important for Albert.

Based on literature that gives an overview of few-shot architectures, the primary concept of few-shot models typically belongs to one of the categories listed below, although combinations of categories are also possible [51],[34],[20]:

1. **Metric Learning** The idea behind metric methods for few-shot learning is that a network classifies the similarity between a query image and a number of example classes based on their distance in feature space [23], [56], [52], [6], [49], [50], [27], [13], [62]. Some methods encode classes to a prototype [50], [27] or try to relate classes in a Graph Neural Networks [13], [62]. Eventually the distance between query features and example class features is used to classify a query image. Distance functions can be anything, but are often euclidian, cosine or learnt functions.
2. **Optimization-based methods** The idea behind optimization-based methods is to take a model and provide it with the most optimal parameters to achieve many different tasks. Optimization-based methods could for example train a simple neural network with a fully connected layer to have the best input parameters for solving many different few-shot tasks (tasks can be similar to figure 2.1). However, this could also be used for more advanced networks like ProtoNet. The power of optimization-based methods lies in quickly finding parameters that generalize well to different tasks that individually can have very little input data. Overall optimization-based methods will increase the training complexity of a model, but inference complexity remains unchanged. [40], [11], [32], [3], [21], [43], [39], [38], [46]
3. **Sequence-based methods** The idea behind sequence-based methods is that they do not use a distance metric for label prediction, but directly predict from a sequence of input images. Because of the deep neural model directly connecting input and output, some papers also refer to these methods as *model-based methods* [34]. Only few few-shot learning models employ a sequence-based classification technique. Typically they work well on low-feature datasets like Omniglot [25].[44],[31]
4. **Transfer learning methods** The idea behind transfer learning methods is that they use pre-training on other datasets to extract better features for a few-shot task. The benefit of pre-training is that general image recognition strategies can be applied to both the pre-training and the few-shot dataset and knowledge from one dataset might be beneficial for the other. Knowledge applied in identifying flying birds might for example also be beneficial on different tasks like classifying airplanes or telling the difference between birds and mammals. Especially in few-shot learning where input data is very limited, it is beneficial to use as much pre-learnt image processing tactics as possible. [58], [17],[8], [47], [16], [18]
5. **Augmentation-based methods** A problem with few-shot learning is that the amount of data that is available is very limited. Augmentation-based methods aim to solve this by augmenting the input data and thus creating more data [5].

In their paper P>M>F, Hu et al. [16], notice that transfer learning methods usually work best. Usually transfer learning methods use large backbones like WRN (Wide Residual Network) [17], [8], [47] or ViT (Vision Transformer) [16], which they pre-train on a large dataset. Most other methods use ResNets (Residual Networks)[5], [18] and CNNs (Convolutional Neural Networks) [56], [50], [52], [27], [49], which they then train from scratch. The P>M>F paper argues that pre-training (P) is most important. Meta-training (M), the conventional way of training a few-shot model, comes second. Fine-tuning (F) has the least influence on a models' performance. Training a randomly initialized ViT-feature extractor with ProtoNet [50], a common few-shot architecture, yields an accuracy of 49.1%. When using a pre-trained feature extractor in the same setup, P>M>F achieves an accuracy of 98.0%. With fine-tuning P>M>F even achieves accuracies of 95.3% (1-shot) and 98.4% (5-shot).

Some metric methods however also perform very well. Especially TRIDENT [49] stands out, as it achieves a 5-shot accuracy of 95.95% and a 1-shot accuracy of 86.11% on mini-ImageNet. Both P>M>F and TRIDENT outperform all other models' accuracies on mini-ImageNet and are thus the best candidates to also perform well on a custom supermarket dataset. To gain more insight in how both models work, the specific architecture of both TRIDENT and P>M>F is explained in more detail in the next sections.

## 2.1. TRIDENT

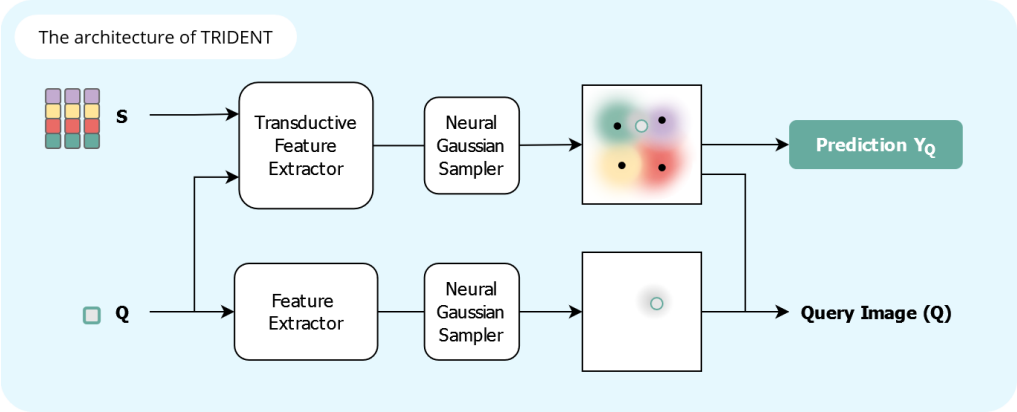


**Figure 2.2:** The simplified architectures of TRIDENT. Adapted from [49].

A simplified version of the architure of TRIDENT is visualized in figure 2.2 above. TRIDENT [49] is based on simple CNN-based feature extractors and a transductive attention module that uses both query (Q) and example- or support- input (S) to extract meaningful features. TRIDENT uses a neural gaussian sampler to predict the mean and standard deviation for class-specific and query image data. This is then used to predict the true class label of the query input and, together with a mean and standard deviation extracted from just the query information, used to reconstruct the query image. This query image reconstruction is used for self supervised loss and forces the model to pull class information and background information apart.

## 2.2. P>M>F

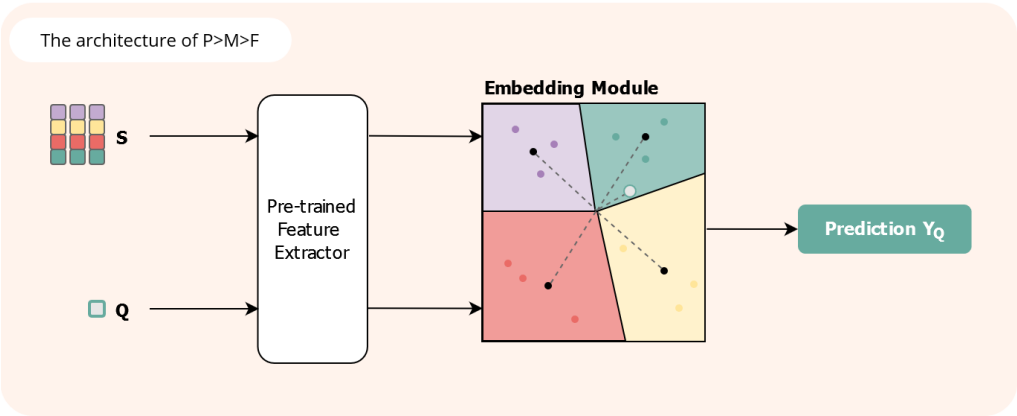


**Figure 2.3:** The simplified architectures of P>M>F. Based on [16].

The architecture of P>M>F can be simplified to figure 2.3, which is shown above. P>M>F [16] consist of a pre-trained feature extractor that feeds features to a ProtoNet [50]. Instead of the regular euclidian distance metric that ProtoNet uses, P>M>F uses ProtoNet with a cosine distance metric. The feature extractor that works best for P>M>F is a ViT that has been pre-trained with DINO, an unsupervised pre-training method that has proven to yield better class features than supervised training methods [7].

Vision Transformers became a popular backbone and classifier choice after the influential paper ”An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale” [10]. This paper proved that it was possible to apply the Transformer architecture that had commonly been used for deep language

models to images. This resulted in great performance gains. In ”An Image is Worth 16x16 Words...”, ViT’s are pre-trained on JFT-300M, a huge dataset, and fine-tuned on other datasets like ImageNet and CIFAR-100. There are different ways of pre-training a ViT however. P>M>F explores DINO [7], BEiT [4] and CLIP [36] as pre-training methods, pre-trained on respectively the ImageNet1K, ImageNet-21K and YFCC100m datasets. For P>M>F DINO yields the best performance.

DINO is a student-teacher method of pre-training a network that achieves good class separation of features [7]. When tested on ImageNet it is evident that features from subgroups like monkeys and birds are well separated. Further separation is evident among specific classes like orangutans and chimpanzees. DINO excels in matching query features to the right class with a KNN-classifier. It also reports that augmented images are close to their originals in feature space when using cosine distance. This makes sense, as DINO trains a student to output the same features as the teacher while both use different augmented versions of an image as input.

## 2.3. Assumptions of few-shot object classifiers

Both P>M>F and TRIDENT, as well as many other few-shot classifiers make some important assumptions about their data, listed below are two assumptions that are important for a few-shot model that should work well on Albert:

1. Conventional few-shot methods assume that the query class is the same as one of the support classes [51],[34],[20]. Because of this, some methods like P>M>F even assume that query classes can never be a non-support class [16]. In a supermarket setting however, support sets with less than 300.000 classes are logically preferred, as this saves on computing time. This might also be necessary if a product detector wrongly classifies non-products as a product. If for instance a human is wrongly classified as product, a model should be able to tell that this is a wrong classification instead of attributing it to one of the support classes. The perfect supermarket product classifier should therefore also be able to tell if a query image is not in our support set. In other words: also classify non-classes.
2. P>M>F and TRIDENT, like most well-performing few-shot learning models [17], assume that class features are Gaussian distributed. TRIDENT incorporates a neural Gaussian sampler that predicts a mean and a variance from class features and matches this to a query image using a learnt predictor. P>M>F uses a softmax function that basically attributes a query feature to the closest class mean.

# 3

# Datasets Used

This chapter will shortly go over the datasets used by this paper to evaluate few-shot classifiers as closely to a real-world supermarket scenario as possible. This chapter introduces two new datasets: AholdSet-V1 and it's extended version AholdSet-V2. Both are designed to be **representative** of a supermarket that Albert will be deployed in, to be **inclusive of different product contexts**, such as different lighting or angle and to be **diverse** enough to include a varied array of products and not just products of one subgroup.

## 3.1. AholdSet-V1

AholdSet-V1 is a balanced dataset that consists of 35 classes with 191 images per class. Dataset images consist of products that are cropped from annotated images of stocked shelves of the supermarket that Albert will eventually work in. These shelf images are representative of what the robot camera sees during operation and often include multiple products (see figure 3.1). To be **inclusive of different product contexts**, photos are taken from many different angles and in many different lighting conditions. Products are also frequently placed in different positions to ensure that the background and surroundings of a cut-out product do not influence its class prediction. To ensure that AholdSet-V1 contains a **diverse** array of products, different product subgroups were selected. Figure 3.1 for example shows that milk/yoghurt cartons, wok sauces, canned vegetables and boxed canned vegetables are selected. In the full AholdSet-V1, many other subgroups like baking- or spice mixes in different shapes and sizes are included as well.


**Figure 3.1:** Annotated image from which individual products are cropped for AholdSet-V1


**Figure 3.2:** The design decisions taken into account in creating a dataset that mimics the real world as close as possible.

To make AholdSet-V1 resemble the real world as close as possible, AholdSet-V1 makes a few design decisions, of which the most important ones have been listed in figure 3.2. Products have been photographed in different lighting conditions (figure 3.2a). Photos have been taken with and without motion blur (figure 3.2b), because an actual camera will move and sometimes have blurry images of products. Sometimes products may be partially obstructed by the gripper or only partially visible, which is why full and partial products have been included (figure 3.2c). Because a real supermarket has products that look very similar, at least to the human eye, very similar products have deliberately been included. Figure 3.2d shows an example of two of those very similar products that belong to a different class.

This paper evaluates the accuracy of classifiers on products seen during training (**seen accuracy**) and on novel products (**unseen accuracy**). Validating classifiers' accuracy on products not seen during training means that some product classes have to be kept apart to validate and test on, while the rest can be used for a training dataset. With only 35 products AholdSet-V1 is relatively small, and setting aside classes for validation and testing means there are few left for training, which increases the risk of overfitting on those specific classes. When AholdSet-V1 is divided into 21 training, 7 validation, and 7 testing classes (for 5-way classifiers, each set needs at least 5 different classes), both TRIDENT and P>M>F perform poorly. This poor performance can be seen in table 3.1 on the next page. Training and validating classifiers on RP2K[35] and testing them on AholdSet-V1 yields significantly better results however. RP2K is an extensive Chinese supermarket dataset, designed to be as close to real-world supermarkets as possible [35]. Because this training method yields a better accuracy on AholdSet-V1, the **unseen accuracy** of classifiers is evaluated by training and validating on RP2K and testing on the full AholdSet-V1. The **seen accuracy** of classifiers is evaluated fully on AholdSet-V1, for which 15% of all images per product are kept apart for validation and 15% for testing. The remaining 70% of images per class is used to train classifiers on.

| | Train | Val | Finetune | Test | Accuracy |
|---|---|---|---|---|---|
| *TRIDENT* | **AholdSet-V1** | **AholdSet-V1** | - | **AholdSet-V1** | 0.635 |
| | **RP2K** | **RP2K** | - | **AholdSet-V1** | ***0.823*** |
| | **RP2K** | **RP2K** | **AholdSet-V1** | **AholdSet-V1** | 0.589 |
| *P>M>F* | **AholdSet-V1** | **AholdSet-V1** | - | **AholdSet-V1** | 0.763 |
| | **RP2K** | **RP2K** | - | **AholdSet-V1** | ***0.844*** |

**Table 3.1:** The accuracy on unseen classes in AholdSet-V1. AholdSet-V1 likely overfits on the training data. Hence the small accuracy when training, validating and testing on AholdSet-V1.

## 3.2. AholdSet-V2

AholdSet-V2 is an unbalanced extension of AholdSet-V1. AholdSet-V2 extends AholdSet-V1 with 709 more uncropped images of shelves. The fact that this unbalanced dataset has a different number of images per class is not critical, as all classes have the same chance of being selected [33]. AholdSet-V2 uses the same design criteria as AholdSet-V1 and uses images with different lighting and blur, partially visible products and products that look visually similar.

An important reason for extending AholdSet-V1 to AholdSet-V2 is the assumption that the poor unseen performance of AholdSet-V1 is due to classifiers overfitting on the limited amount of training classes. Table 3.2 shows this effect measured by this paper on RP2K. Even though this test is very limited, training, validating and testing on more classes seems to be beneficial for the accuracy of a classifier. This means that extending AholdSet-V1 with more classes could also improve its accuracy. Interestingly RP2K does have a significantly higher accuracy (0.913|0.910) than AholdSet-V1 (0.635|0.763) with the same amount of classes. This could indicate that AholdSet-V1 is just a harder dataset, but many different factors could contribute to this low accuracy. AholdSet-V2 seems to be a better dataset however. AholdSet-V2 was derived from AholdSet-V1 by adding more classes and redistributing them across the sets while intentionally keeping similar product pairs together to increase dataset complexity. Testing AholdSet-V2 with P>M>F yields training-, validation-, and test accuracies within a 0.96-0.98 range, which indicates that AholdSet-V2's training set is representative of its validation and test set. This range also closely approaches the maximum accuracy achieved by P>M>F on mini-ImageNet [16]. Hence, AholdSet-V2 seems to be a qualitatively better dataset than AholdSet-V1 to evaluate models few-shot models on for applications like Albert.

| | # Training Classes | # Validation Classes | # Testing Classes | **RP2K Accuracy** |
|---|---|---|---|---|
| *TRIDENT* | 21 | 7 | 7 | 0.913 |
| | 40 | 9 | 9 | 0.923 |
| | 575 | 72 | 72 | 0.962 |
| *P>M>F* | 21 | 7 | 7 | 0.910 |
| | 40 | 9 | 9 | 0.940 |
| | 575 | 72 | 72 | 0.995 |

**Table 3.2:** The unseen accuracy of classifiers on RP2K as measured by this paper with a varying amount of training-, test-, and validation classes. Generally having more training-, validation- and test- classes gives a better accuracy.

Since AholdSet-V2 shows comparable training, validation, and testing accuracies when used with P>M>F, and its test accuracy closely matches the maximum accuracy of P>M>F, AholdSet-V2 seems to be a reasonable dataset to evaluate the accuracy of models on. To evaluate the **seen accuracy** of AholdSet-V2, 41 different classes were split into 15% validation- and 15% testing images per class. The rest of the images were used for training data. These 41 training classes containing 70% of their original amount of images were also used as training data in the unseen dataset to make validating and testing models for seen and unseen datasets easier. The unseen dataset includes 8 different unseen classes for validation. The **unseen accuracy** of models was calculated by testing them on 35 novel classes. A visualization of both seen and unseen dataset distributions can be found in figure 3.3 on the next page. The amount of

classes that are in the both seen and unseen parts of the dataset can be found in table 3.3.

| | Train | Validation | Test |
|---|---|---|---|
| *Seen Classes* | 41 | 41 | 41 |
| *Unseen Classes* | 41 | 8 | 35 |

**Table 3.3:** Class distribution in seen and unseen AholdSet-V2

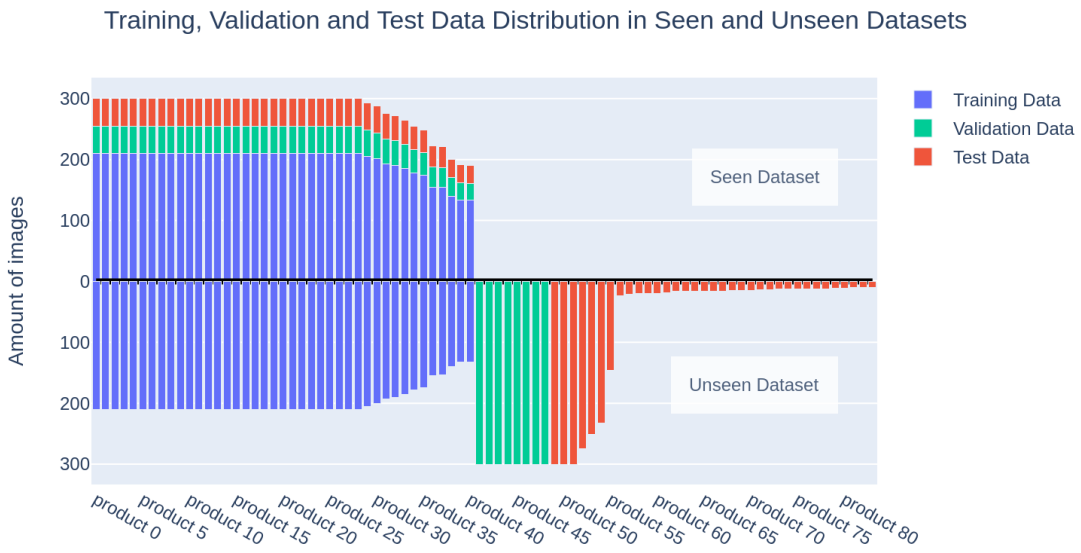


**Figure 3.3:** The distribution of images in the seen- and unseen AholdSet-V2. Training data is the same for both sets to make validating and testing the seen- and unseen dataset easier.

# 4

# Analysis of P>M>F and TRIDENT

Both TRIDENT and P>M>F are extremely good state-of-the-art classifiers that have respective 5-shot accuracies of 95.95% and 98.0% on mini-ImageNet. To answer the research question set in this paper: **"What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?"**, it is important to measure how both few-shot classifiers perform in terms of the requirements of this paper. This chapter will specify the experiments done to measure those requirements and the results that follow these experiments.

P>M>F however makes an assumption about its data that make it ultimately unsuitable to be used on a supermarket robot: query images are always classified as the most likely target product. Especially when a query image ends up being a human, classifying it as the likeliest product is dangerous. However P>M>F might still perform better on other requirements, which is why this chapter answers the sub-question: **Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?** If neither of them meets Albert's requirements or P>M>F is selected as the best model, the best model will be adapted to a model that can both classify non-classes and fits Alberts' requirements.

## 4.1. Experiments

To determine what few-shot classifier is best, it is important to evaluate how well P>M>F and TRIDENT pass the requirements set in this paper. P>M>F and TRIDENT are compared on four important metrics coming from the requirements: 1) Inference time; 2) Memory Usage; 3) Accuracy on seen classes; and 4) Accuracy on unseen classes; The amount of shots or example images that models use has influence on all of these metrics. Inference time will be longer and memory usage will be higher when comparing to more example images, but accuracy might also increase. For this reason, P>M>F and TRIDENT are compared in 1-, 3- and 5-shot settings.

### 4.1.1. Accuracy on seen- and unseen datasets

To test both models' seen- and unseen accuracy, they are evaluated on AholdSet-V1 in a few-shot setting. Eventually this paper is interested in determining the accuracy of a model in a realistic supermarket setting, where a classifier has to distinguish target picking classes from non-target classes, but P>M>F can not do this. To still compare TRIDENT and P>M>F, they both will be evaluated in the few-shot setting they are designed for, rather than in a setting where non-classes have to be identified. They model with the highest accuracy will be preferred in this case. Logically a model that can distinguish classes better, will also be better at distinguishing them from non-classes. This is because when a model learns class boundaries well, it can use that knowledge to decide if new items fit into a class or not. This means that all query image classes will be the same as example classes for all accuracy tests with TRIDENT and

P>M>F.

### 4.1.2. Memory usage and inference time

Another experiment will be done to test memory usage and inference time. For this a computer with an NVIDIA GeForce RTX 2080 SUPER GPU will be used. Both models will be tasked to repeatedly classify one image with a 5-way, N-shot-classifier. The inference time per image is calculated by running inference over 10.000 images, measuring the total inference time for this operation and dividing this by the amount of images. Memory usage is evaluated during inference. It is averaged over 5 different captures that are taken roughly 30 seconds apart.

## 4.2. Results

The figure below (figure 4.1) shows the results of P>M>F and TRIDENT on the requirements specified in this paper. These results have been produced with the two experiments set out in section 4.1. They will be evaluated on every requirement set in this paper separately in the sections below.

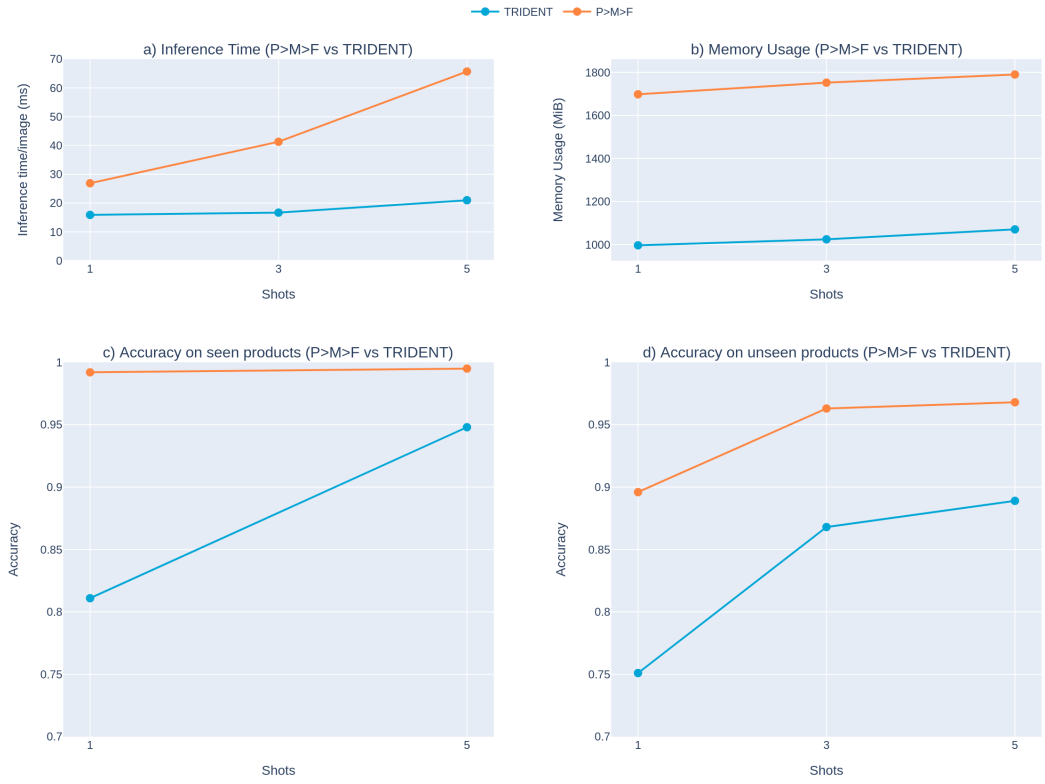


**Figure 4.1:** Comparison of P>M>F and TRIDENT (Accuracy on AholdSet-V1, Memory Usage and Inference Time). P>M>F has a consistently higher accuracy than TRIDENT in all settings and passes the memory usage requirement set in this paper, just like TRIDENT. Both models however fail to have inference times lower than 3.6 ms and are therefore not suited for visual servoing. However the architecture of P>M>F allows for inference time-optimizations. This and its consistently better accuracy make P>M>F preferred over TRIDENT.

### 4.2.1. Inference Time

The requirement this paper sets for inference time is: *The model's inference time should be as low as possible, but models with a high detection accuracy are preferred over those with a low inference time. To still have good closed-loop object tracking, this paper requires a maximum inference time of 3.6 ms per query image.* TRIDENT has a consistently lower inference time than P>M>F. However TRIDENT still

has a minimum inference time of 15.9 ms per image, which is much more than the 3.6 ms required for visual servoing. This makes both P>M>F, with an inference time between 26.9 and 65.7 ms, and TRIDENT unacceptable for use on Albert.

Doing heavy calculations before deployment however might heavily decrease inference time. Especially since products are static and will not change during deployment, product features could be calculated beforehand. For TRIDENT this is not possible. TRIDENT encodes query and support images transductively, which means that for every classification all support images, combined with all query images, have to be converted to features. P>M>F however compares query image features only to a support prototype. This support prototype is by default always calculated from example images of a product during deployment, but this can also very well be done before deployment. If product prototypes are calculated before deployment, P>M>F can skip heavy calculations during inference, which makes for a faster classifier. This change will also ensure that the inference time and memory usage of P>M>F become invariant to the number of example images, as the process of calculating prototypes from example images during deployment will no longer occur.

### 4.2.2. Memory Usage

This paper sets the following requirement for memory usage: *As a design decision, inference GPU memory usage should be below 4GB.* Even though P>M>F consistently uses about 70% more GPU memory than TRIDENT, both models pass this papers' requirement. With P>M>F and TRIDENT having a maximum memory usage of respectively 1790 and 1071 MiB, they can both run on lower-end GPU with 4GB memory.

### 4.2.3. Accuracy on Seen Dataset

The accuracy requirement to classify product classes seen during training set by this paper is: *The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for product classes that are seen during training. As a design decision, this should be at least 90%.* As discussed in section 4.1, the accuracy of P>M>F and TRIDENT has not been determined in a realistic supermarket setting, but rather in a few-shot setting on a realistic supermarket dataset. This means that it is not possible to use the 90% accuracy threshold set in the requirements as an absolute measure, since the way that this accuracy is measured is different. Instead, when it comes to accuracy, the best performing model will be the preferred model to use as a base classifier.

Seen accuracies are high for both models, especially in the 5-shot setting. Interestingly, P>M>F's 1-shot accuracy (99.2%) is not much different from it's 5-shot accuracy (99.5%). This indicates that P>M>F is better at extracting generalized class-representative features per image than TRIDENT. This is highlighted by the fact that using 5 instead of 1 example images per class only increases its accuracy by 0.3%. TRIDENT, on the other hand needs more examples per class to function well. With more examples TRIDENT predicts a more meaningful class mean and standard deviation in feature space. With 94.8% in a 5-shot setting, it has a 13.7% higher accuracy than in a 1-shot setting, but still performs worse than P>M>F. Because P>M>F has a consistently higher seen accuracy, even in a 1-shot setting, it is the preferred pick to classify seen products.

### 4.2.4. Accuracy on Unseen Dataset

This paper sets the requirement for product classes not seen during training to be: *The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for novel product classes that are not seen during training. As a design decision, this should be at least 90%.* Both models understandably perform worse on an unseen dataset than on a seen dataset. After all, they have only learned to extract meaningful features that differentiate between seen classes. Unseen classes might be characterized by different features. P>M>F however consistently outperforms TRIDENT on the unseen dataset as well. TRIDENT has a minimum accuracy of 75.1% and a maximum accuracy of 88.9%, while P>M>F has accuracies of 89.6% to 96.8%. Because P>M>F consistently performs better than TRIDENT on unseen products, P>M>F is the preferred choice when it comes to unseen accuracy.

## 4.3. Conclusion

This chapter addresses the subquestion: **Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?**. To determine the optimal classifier for Albert, P>M>F and TRIDENT are compared based on the criteria set in this paper. P>M>F consistently demonstrates exceptional accuracy for both seen and unseen products and outperforms TRIDENT in both settings. On AholdSet-V1, P>M>F achieves unseen accuracies ranging from 88.9% to 96.8%, and seen accuracies between 99.2% and 99.5%, whereas TRIDENT's highest unseen and seen accuracies are only 88.9% and 94.8%, respectively. Thus, in terms of accuracy P>M>F is the preferred choice as a classifier for Albert. Both P>M>F and TRIDENT meet the memory usage requirement of this paper (below 4GB), but their high inference time makes them ultimately unsuitable to integrate on Albert. TRIDENT, with the lowest inference time of the two, takes 15.9 ms per image, well beyond this paper's maximum requirement of 3.6 ms per image. Consequently, neither P>M>F nor TRIDENT are suitable for visual servoing

Pre-calculating prototypes for products before deployment, which is only possible with P>M>F, has the potential to significantly reduce inference time however. Additionally, this approach offers the advantage that P>M>F's inference time and memory usage become invariant to the number of shots or example images used for each product. This means that the classifier with the highest performance — 5-shot P>M>F in this case — can be selected without concern for increased inference time or memory usage due to additional example images. Especially since TRIDENT can never be fast enough to use on Albert, the high performance of P>M>F, coupled with its anticipated decrease in inference time, makes P>M>F the most attractive choice as a classifier for Albert. However, this means that P>M>F has to be modified to accommodate the classification of non-classes.

# 5

# Product-ProtoNet

With its high accuracy on both seen and unseen classes and its memory usage suited for a lower-end GPU, P>M>F is a great option as a base for a classifier for Albert. P>M>F however has two major problems that require solving in order for it to work on Albert: 1) It's inference time makes it unsuitable for visual servoing. For visual servoing a maximum of 3.6 ms per image is acceptable, but P>M>F takes at least 26.9 ms to classify one single image; and 2) P>M>F, like most few-shot models can not classify non-target classes and instead always matches images to their likeliest product class; In this chapter, solutions to both of these problems are proposed. The model that incorporates these solutions is called Product-ProtoNet, as it classifies products with a classifier based on ProtoNet, using insights given by P>M>F.

As the previous chapter answered the sub-question: **"Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?"** by declaring P>M>F the most attractive choice, this chapter will focus on this paper's second sub-question: **"If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?"**

## 5.1. Introduction of Product-ProtoNet

Because P>M>F makes for such an excellent base in terms of accuracy, Product-ProtoNet is largely based on P>M>F, especially on its suggested implementation of ProtoNet. Meaning standard ProtoNet with a cosine distance classifier instead of Euclidean and with a ViT-small feature extractor pre-trained with DINO instead of a CNN that is trained from scratch. Product-ProtoNet however makes significant changes to the ProtoNet architecture. Instead of utilizing a softmax predictor to assign the likeliest support class as the query class, Product-ProtoNet uses a sigmoid predictor that predicts what cosine distance is acceptable for query images to belong to a class. To maintain the ProtoNet-like functionality to assign query images to the most probable class, Product-ProtoNet compares query images to several helper classes and the target class, and selects the most probable option. Helper classes are classes that are most similar to the target class. They aid in determining the class of a query image, especially when comparing very similar products. The complete architecture of Product-ProtoNet is illustrated in figure 5.1 on the next page.

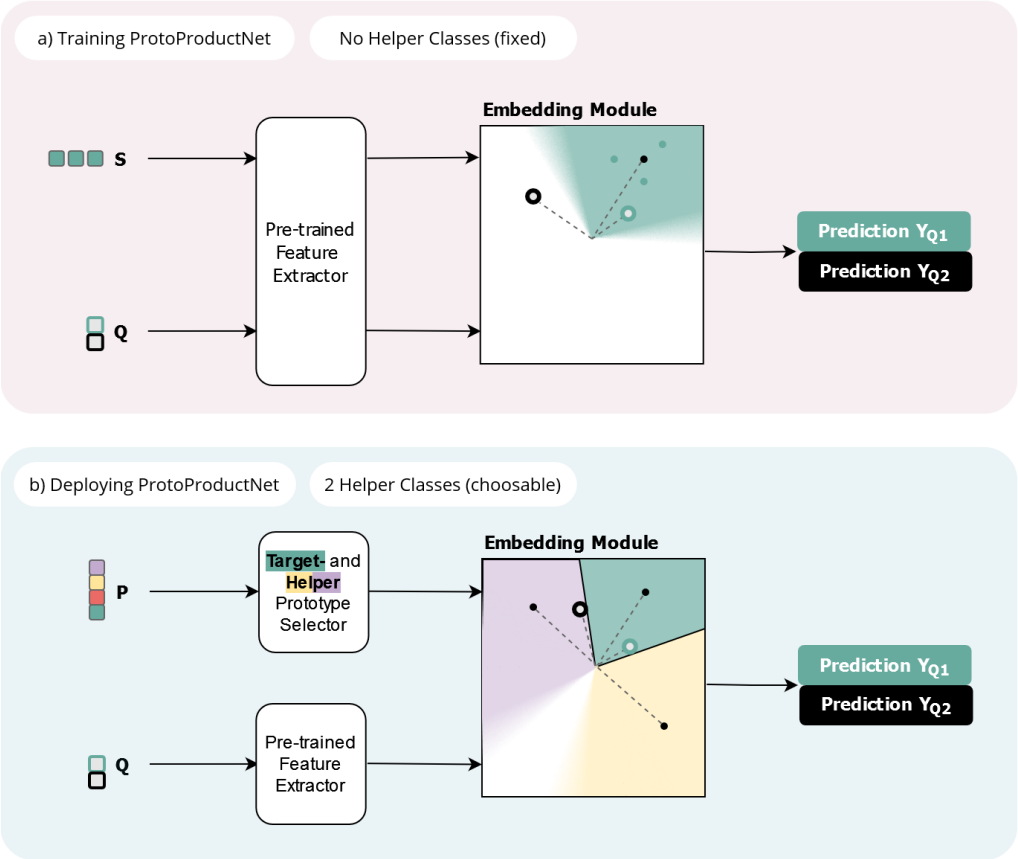


**Figure 5.1:** The architecture of Product-ProtoNet, which is very similar to the architecture of P>M>F. Only a simple sigmoid classifier is trained to tell classed apart based on their cosine distance. During deployment one can choose to add helper prototypes to achieve better performance on similar unseen products. Pre-computed prototypes are denoted as P, support images as S and query images as Q.

### 5.1.1. Mathematical formulation of Product-ProtoNet

The general idea of Product-ProtoNet becomes clear in figure 5.1. This section gives a mathematical formulation of this idea and makes Product-ProtoNet a reproducible and usable model.

Product-ProtoNet, just like ProtoNet uses prototypes to compare query images to. Given a small support set ($S_k$) of labeled class examples, $\{\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_N, y_N\}\}$ where $x_i$ denotes the example image and $y_i \in \{1, ..., K\}$ denotes the class this example has, a prototype ($c_k$) is constructed for every class $k$ by calculating the mean of all class image features, using the following formula:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i) \tag{5.1}$$

Here $f_\phi(x)$ is a feature embedding function with learnable parameters $\phi$. In ProtoNet $f_\phi(x)$ is a CNN, while Product-ProtoNet uses a pre-trained ViT as feature extractor.

ProtoNet uses a softmax function, together with a function of the distance between query features and prototypes $d(f_\phi(x), c_k)$ to predict the likeliness that a query image $x$ belongs to class $y$. This function compares the distance of query features to all prototypes and is given by:

$$p_{\phi,w,b}(y = k|x) = \frac{\exp(-(w * (d(f_\phi(x), c_k) + b))}{\sum_{k'=1}^{K} \exp(-(w * (d(f_\phi(x), c_k) + b))} \tag{5.2}$$

In ProtoNet, $d(f_\phi(x), c_k)$ represents the Euclidean distance between $f_\phi(x)$ and $c_k$. The weight $w$ and

bias $b$ are respectively fixed at 1 and 0, and are not learnable. P>M>F deviates from this and uses cosine distance along with learnable weight and bias parameters, which can sharpen the softmax output.

Product-ProtoNet however uses a different prediction metric. The output of a softmax-predictor will always sum to 1, and in situations where Albert sees products that do not belong to a target class, this behaviour is undesirable. With a softmax-predictor it is impossible that all example classes have a likeliness of 0, meaning they can not all be classified as not-a-product. This is one of the major problems of P>M>F and specifically this means that an algorithm that uses a softmax-predictor will always favor the likeliest match from a series of options, or prototypes in this case. Especially P>M>F with a high weight $w$ (see equation 5.2), will always match features that lay slightly closer to one prototype to that prototype class. For this reason Product-ProtoNet uses a sigmoid classifier for predicting the likelihood that a query image is the same class as a target prototype, as its output is unaffected by the relationship between prototypes and it can assign a likelihood of 0 for all prototype classes. The formula for this is given by:

$$p_{\phi,w,b}(y = k|x) = \sigma(w * d(f_\phi(x), c_k) + b) \tag{5.3}$$

Here $\sigma(x)$ is a sigmoid function that is used to clamp predictions between 0 and 1. $w$ and $b$ are a learnable weight and bias and can be used to translate a cosine distance to a likelihood.

As classes can also be not the target class, the likeliness that a class is not class k is given by:

$$p_{\phi,w,b}(y \neq k|x) = 1 - p_{\phi,w,b}(y = k|x) \tag{5.4}$$

Product-ProtoNet assumes that if $p(y \neq k|x) > p(y = k|x)$, $y$ is not class $k$.

Using a sigmoid- instead of a softmax classifier however means that essential information about the relationship between prototypes is lost. Picking the most likely from a number of examples seems like a reasonable classification strategy, especially for classes that look very similar. With a sigmoid classifier, every class is classified without regard for other classes however. If Albert were to classify e.g. chunky peanut butter as chunky or smooth peanut butter, a sigmoid classifier could wrongfully classify them as both viable options. This risk is especially apparent for unseen classes, for which a classifier has not learned to encode them apart in the feature space. A sharp softmax-classifier could instead attribute a query image to its likeliest class. It could correctly see that a query image of chunky peanut butter looks slightly more similar to a chunky peanut butter prototype and thus classify it as such. Implementing this likeliest-class selection in Product-ProtoNet gives a function that is very similar to equation 5.2 with a large weight $w$:

$$p_{\phi,w,b}(y = k|x) = \begin{cases} h_{\phi,w,b}(x, c_k), & \text{if } k = \text{argmax}_n(h_{\phi,w,b}(x, c_n)) \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

Here $h_{\phi,w,b}(x, c_k)$ is given by equation 5.3 and $n \in \{1, ..., K\}$, where K is the total number of classes or prototypes that is selected. This means that only the likeliest class $k$ from $K$ different classes can be the actual query class, as the likelihood $p_{\phi,w,b}(y = k|x)$ for all other classes is set to 0.

Arguably this has the most effect for classes that lay close together in feature space and not for randomly selected classes. After all, this paper trains a sigmoid predictor specifically to do the latter, and similar unseen classes are expected to be the hardest to classify with any classifier. To select closest classes during inference, Product-ProtoNet will use the following formula:

$$\text{top-K-selector}_n(d(c_k, c_n)) \tag{5.6}$$

$n \in \{1, ..., N\}$, where N is the total amount of product classes. Here a top-K-selector will select the K most similar classes that will be used in equation 5.5. Similar classes are defined as classes for which the cosine distance between target class $c_k$ and prototype $c_n$ is highest, meaning they are close together. These close classes are called helper prototypes, as they help distinguish a target picking prototype $c_k$ from other classes.

### 5.1.2. Other design choices

Images passed into Product-ProtoNet are assumed to be of one and only one product. Even if an image were to contain multiple products in very rare and exceptional cases, Product-ProtoNet should determine if the image contains the target class or not. This means that the model can be forced to predict whether a query image belongs to one and only one class. For this reason Product-ProtoNet will use binary cross entropy as a loss function, in contrast to ProtoNet that uses cross entropy.

Product-ProtoNet will be trained to classify one target class and one random non-target class. This forces it to learn what cosine distance is acceptable to classify query images as a target class. If two similar classes are selected, the classifier will likely tend to classify them as the same class, resulting in a high loss, telling the classifier to move these classes further apart in feature space.

A big advantage of using P>M>F as a base-architecture over TRIDENT is that P>M>F does not extract features transductively, a better accuracy in any setting. P>M>F only takes prototypes, constructed from example images, into account for calculating the distance to query features in feature space. As product prototypes are static, a prototype for every product can be calculated before deployment, saving on calculations during deployment. This means that Product-ProtoNet can be deployed in a memory- and inference time optimized way, where prototypes are calculated before query inference from all available product images. During runtime the model only has to compare a query image to the selected prototype.

### 5.1.3. Assumptions of Product-ProtoNet

Like any neural network, Product-ProtoNet makes assumptions that simplify reality. Listed below are the most important ones:

- Because this assumption works very well for P>M>F and TRIDENT, Product-ProtoNet too assumes a gaussian distribution of class features.
- A query image is deemed not to be the target picking class if it is more likely to not belong to the target class than to belong to it. Practically this means Product-ProtoNet uses a likelihood threshold of 0.5 to determine if query images belong to a class.
- Query images in Product-ProtoNet contain one and only one product at a time.
- Product-ProtoNet is able to linearly separate products on their distance in feature space. Without this, attributing query images to their closest prototype would not be possible.
- Training a sigmoid classifier on random classes is equally effective as training a softmax classifier on random classes.

### 5.1.4. Hyperparameters

To keep the results of Product-ProtoNet as comparable to the results of P>M>F as possible, Product-ProtoNet has been trained with on the same random seed and with the same backbone learning rate as P>M>F. Product-Protonet, however, is granted a higher learning rate than P>M>F for its weight and bias in its sigmoid classifier (see equation 5.3). This makes it less dependant on perfect initialization. An overview of the hyperparameters used is given in table 5.1.

| Hyperparameter | P>M>F | Product-ProtoNet |
|---|---|---|
| *n_training_epochs** | 2000 | 2000 |
| *scheduler** | cosine | cosine |
| *n_warmup_epochs** | 5 | 5 |
| *warmup_lr** | 1e-6 | 1e-6 |
| *minimal_lr** | 1e-6 | 1e-6 |
| *backbone_lr* | 5e-5 | 5e-5 |
| *clf_lr* | 5e-5 | 5e-2 |

**Table 5.1:** *same for classifier and backbone
All hyperparameters have been kept the same between P>M>F and Product-ProtoNet for optimal comparison. Product-ProtoNet however is allowed a higher learning rate for it's sigmoid classifier, as this makes it less dependent on perfect initialization.

### 5.1.5. Subquestions associated with Product-ProtoNet

The research question of this paper is: **What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?** Because P>M>F was selected as the best model to use as a base, but did not fit the requirements of this paper and could not classify non-classes, this chapter asks the following sub-question: **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?** Product-ProtoNet seems the ideal method to reduce inference time for visual servoing and to classify non-target products accurately. It retains the components that make P>M>F so succesful: 1) using a pre-trained feature extractor; and 2) picking the likeliest option from a number of prototypes, as P>M>F does with ProtoNet; It is however important to evaluate if Product-ProtoNet is indeed as perfect as it seems. To do this, this chapter's sub-question is split into several different smaller questions:

1. How good is Product-ProtoNet at classifying non-classes, considering how crucial this is for a few-shot model on Albert?
2. How does Product-ProtoNet compare to P>M>F, given that helper prototypes aim to give it ProtoNet- and thus P>M>F-like function?
3. How many helper prototypes are needed in a realistic supermarket situation?
4. How effectively does Product-ProtoNet distinguish class features in the feature space, as this is a crucial aspect for the effectiveness of helper prototypes?
5. How well does Product-ProtoNet perform on this paper's inference time, memory usage and accuracy requirements?

## 5.2. Experiments

To answer the question: **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?**, the sub-questions introduced in the previous section will be evaluated. The evaluation method for each specific subquestion is specified below. As AholdSet-V2 proved to be better fit to evaluate unseen accuracies on than AholdSet-V1 (chapter 3), all evaluations of Product-ProtoNet are done on AholdSet-V2.

### 5.2.1. Non-class classification capability

The first experiment tries to answer the sub-question: *How good is Product-ProtoNet at classifying non-classes, considering how crucial this is for a few-shot model on Albert?* To assess this, Product-ProtoNet with one helper prototype is used to predict if the non-target class, closest to the target class in cosine distance, is the target class or not. The amount of times Product-ProtoNet does this correctly is then divided by the total amount of attempts to get to an accuracy. To compare this to a baseline, the accuracy of P>M>F is evaluated on the same task.

### 5.2.2. Similarity to ProtoNet

The second experiment attempts to answer the sub-question: *How does Product-ProtoNet compare to P>M>F, given that helper prototypes aim to give it ProtoNet- and thus P>M>F-like function?* To see how well Product-ProtoNet compares to P>M>F, the accuracy of both methods is tested in a P>M>F-like setting. In this setting, query images can only belong to one of the support classes. Rather than evaluating on random classes, classes that are close in cosine distance will be chosen to evaluate classifiers in their extreme. To see if helper prototypes do indeed give Product-ProtoNet a P>M>F-like function, Product-ProtoNet is evaluated with- and without helper prototypes. If Product-ProtoNet with helper prototypes has a similar accuracy to P>M>F, it suggests two things: 1) Selecting the likeliest prototype from a number of prototypes is a key factor in P>M>F's success; and 2) Training Product-ProtoNet to distinguish between target and random non-target classes is as effective as P>M>F.

### 5.2.3. Helper prototypes in a realistic supermarket

The third experiment evaluates the sub-question: *How many helper prototypes are needed in a realistic supermarket situation?* As made clear in the introduction, this paper assumes that a realistic and arguably challenging supermarket setting is represented by a shelf with 10 very similar products on it, from which one has to be correctly classified as the target picking class. How accurately Product-ProtoNet can distinguish target classes from similar-looking classes is evaluated in this realistic supermarket scenario, with a varying amount of helper prototypes. The amount of helper prototypes is varied between 0 and 9, the maximum amount of visible non-target products.

### 5.2.4. Feature separation

The fourth experiment aims to answer the sub-question: *How effectively does Product-ProtoNet distinguish class features in the feature space, as this is a crucial aspect for the effectiveness of helper prototypes?* To achieve this, the paper utilizes T-SNE [54] (t-Distributed Stochastic Neighbor Embedding) to transform its original 384-dimensional prototype and example images into a 2D representation. T-SNE is a nonlinear dimensionality reduction algorithm designed to preserve the original distance distribution between points while projecting them to a lower-dimensional space. Because T-SNE can handle various distance distributions, the cosine distance between features can be used as a metric. As T-SNE will then visualize how far features are apart in cosine space, this projection can be used to analyze how well classes are separable in cosine space.

### 5.2.5. Performance on requirements

The fifth and final experiment with Product-ProtoNet answers an important part of the research question of this paper: **"What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?"** by answering the question: *How well does Product-ProtoNet perform on this paper's inference time, memory usage and accuracy requirements?* Product-ProtoNet is evaluated on this paper's requirements in a similar manner to TRIDENT and P>M>F. Save for the accuracy, which is calculated in a realistic supermarket setting instead of in a P>M>F-like manner (see section 5.2.3). The inference time for Product-ProtoNet is determined by measuring the time taken to classify 10,000 different images, and the average inference time per image is then calculated. It is assumed that a new target class is selected every 150 images. Similar to P>M>F and TRIDENT, inference times are computed using an NVIDIA GeForce RTX 2080 SUPER GPU. Product-ProtoNet's memory usage is monitored during inference and averaged over 5 different captures. Product-ProtoNet's inference time-, memory usage- and accuracy metrics are calculated for Product-ProtoNet with different amounts of helper prototypes.

## 5.3. Results

With the experiments set out in the previous section, the results of the analysis of Product-ProtoNet on various sub-questions are specified in this section. Every research question has its on section with results that apply to that specific question.

### 5.3.1. Non-class classification capability

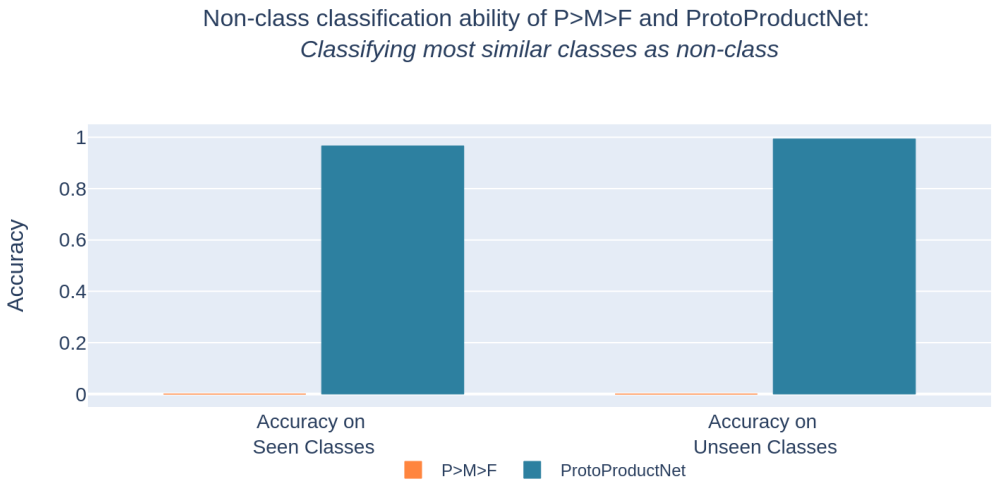


**Figure 5.2:** With an accuracy of 96.6% on seen classes and an accuracy of 99.3% on unseen classes, Product-ProtoNet is clearly good at classifying non-classes. P>M>F on the other hand can not do so and always classifies query classes as the target class, resulting in an accuracy of 0% for both seen and unseen classes.

Figure 5.2 showcases Product-ProtoNet's accuracy on the arguably very challenging task to distinguish very similar non-target classes from target classes. Product-ProtoNet achieves unseen and seen accuracies of respectively 96.6% and 99.3% on this task. P>M>F on the other hand is not designed for non-class classification, and it consistently misclassifies non-target classes as the target class. This leaves P>M>F with an accuracy of 0%. Thus, to answer: *"How good is Product-ProtoNet at classifying non-classes, considering how crucial this is for a few-shot model on Albert?"*, it is evident that Product-ProtoNet is indeed very good at classifying non-classes, as is supported by its impressive performance metrics.

### 5.3.2. Similarity to ProtoNet

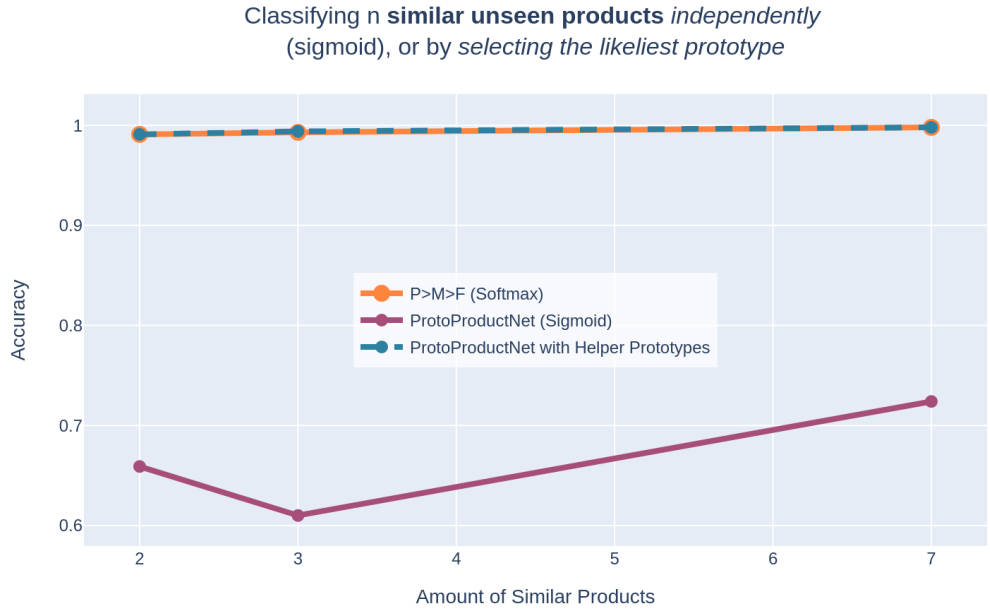


**Figure 5.3:** By transforming P>M>F into Product-ProtoNet (replacing its softmax classifier with a sigmoid classifier and incorporating helper prototypes as a ProtoNet-like mechanism to classify a product as its likeliest prototype), its performance closely matches P>M>F. Highlighting how important a ProtoNet-like mechanism is for both P>M>F and Product-ProtoNet.

Product-ProtoNet is heavily based on the implementation of ProtoNet as suggested by P>M>F. However, since a sigmoid-predictor can only predict if query images are in the right cosine distance threshold to be considered the same class as a target class, predicting this for classes that are very close together in feature space, but slightly different is hard. For this reason Product-ProtoNet uses helper prototypes to attribute query images to the closest class, instead of only looking at the absolute difference. This should give Product-ProtoNet a similar functioning to P>M>F.

That this indeed happens becomes clear in figure 5.3. Product-ProtoNet without helper prototypes functions poorly when classifying similar classes in a P>M>F-like setting, with an accuracy even dipping to 61.0%. When P>M>F is allowed to use helper prototypes to attribute query images to a likeliest class however, its accuracy becomes similar to P>M>F's accuracy. This indicates two things: 1) Choosing the likeliest from a number of classes is a mechanism that makes both P>M>F and Product-ProtoNet work very well; and 2) Training Product-ProtoNet with a sigmoid classifier to tell random classes apart from target classes is enough to make Product-ProtoNet yield a P>M>F-like accuracy.

Curiously, classifying 3 similar products seems slightly harder than classifying 2 similar products for a pure sigmoid classifier, but that might simply be because there are more products to misclassify, so overall products will have the right classification less often. Classifying 7 products on the other hand seems to go better more often, but this likely is because AholdSet-V2 has only 35 test classes and taking 7 classes that the most similar, might yield a 6th or 7th class that is not extremely similar to the target class, yet still the most similar of all other possible classes.

To answer the question: *"How does Product-ProtoNet compare to P>M>F, given that helper prototypes aim to give it ProtoNet- and thus P>M>F-like function?"*, Product-ProtoNet is indeed very similar to P>M>F, when tested in a P>M>F-like setting. For this it needs helper prototypes, which do indeed give it a ProtoNet-like function.

### 5.3.3. Influence of helper prototypes

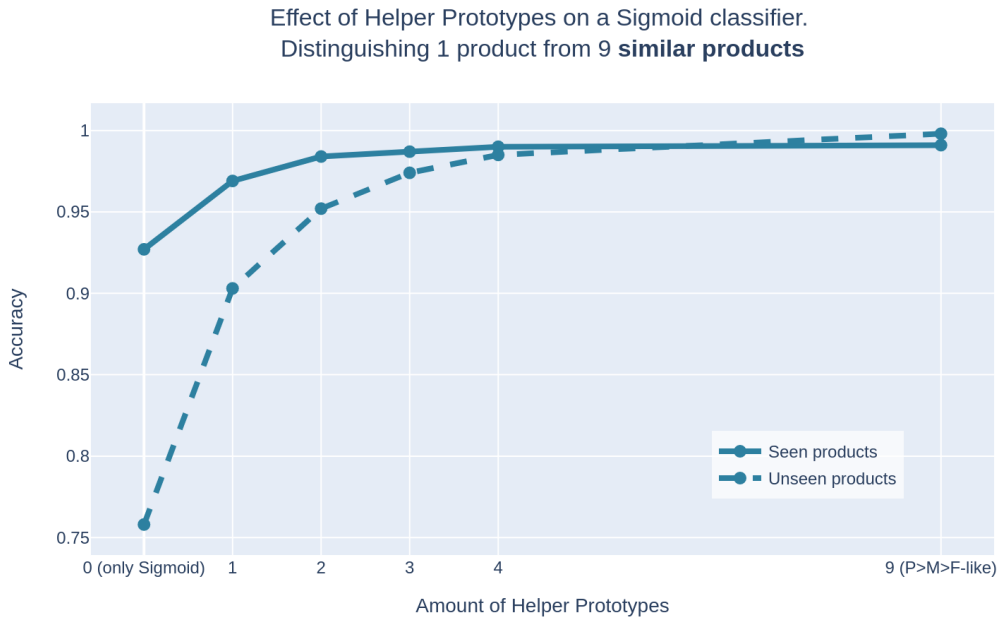


**Figure 5.4:** As expected, more helper prototypes help increase the accuracy of a classifier. Adding one prototype however gives the most significant increase in accuracy: 0.93 to 0.97 for seen classes and 0.76 to 0.90 for unseen classes.

Figure 5.4 shows the results of the experiment to answer the question *How many helper prototypes are needed in a realistic supermarket situation?* This scenario mimics a situation where one product is surrounded by 9 products, similar in feature space. This roughly resembles a shelf with 10 products with similar brand logos, colors, or general product categories. As expected, Product-ProtoNet with as many helper prototypes as visible products works best, especially for unseen products. One helper prototype is however enough to boost Product-ProtoNet's seen and unseen performance over the 90%-accuracy threshold set by this paper.

Interestingly, the features of unseen classes are better separable on cosine distance for unseen classes than for seen classes. This means that the effectiveness of using more prototypes is limited by the features extracted for seen classes. Effectively this means that for identifying seen classes 4 helper prototypes yield an accuracy of 99.0%, and 9 prototypes only raise this to 99.1%. For unseen classes 9 helper prototypes result in a 99.8% accuracy. Likely this means that training Product-ProtoNet to distinguish a target class from a random class is not enough to push it to always separate similar classes in feature space. As clearly it can only separate products seen during training in 99.1% of the cases. Likely the unseen set contains less classes that the classifier finds hard to separate in feature space, which is why Product-ProtoNet's unseen accuracy is higher. Likely the pre-training of Product-ProtoNet's feature extractor also contributes to this. Still, this 99.1% accuracy on seen classes is more than enough to pass the accuracy threshold of this paper.

### 5.3.4. Feature separation

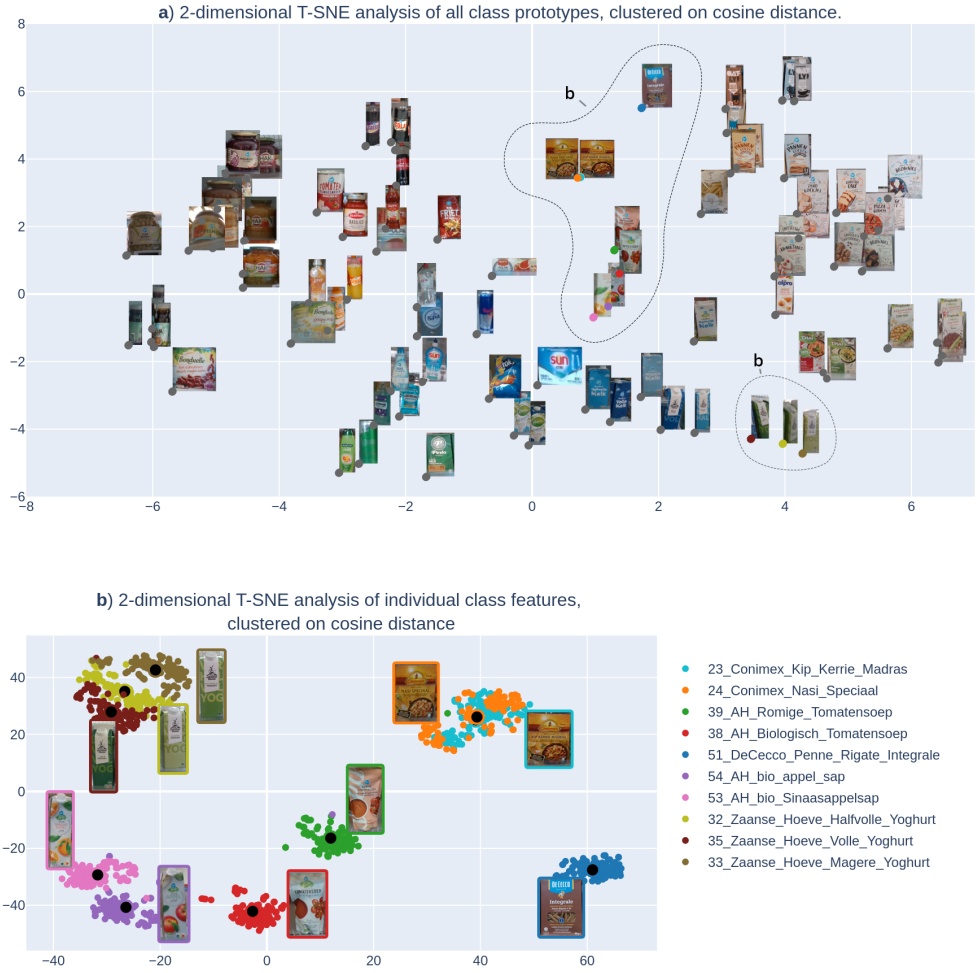


**Figure 5.5:** a) T-SNE dimensionality reduction of product prototype features with a cosine metric; and b) A selection of products with per-image features, visualized with T-SNE. Products with similar color, brand or product type are often closer in cosine distance. The analysis of individual class features shows that most products are separable on cosine distance, even for visually very similar products such as the Zaanse Hoeve products. Some extremely similar products, such as the Conimex spice mixes are barely seperable however.

Figure 5.5 shows a T-SNE projection of a) All product prototypes; and b) A selection of products for which all image features are visualized; In figure 5.5a it becomes clear that products of similar color, brand or product type are often closer together in feature space. Additionally, most product prototypes appear relatively separate in feature space. Notable outliers include the Conimex spice mixes and Kokh Thai red and green curry. The latter is suprising, because both products have a very different color scheme and other products with a similar layout but a different color schema are further apart in cosine distance. However only one of the Kokh Thai mixes was a training class, so Product-ProtoNet might not have had the initiative to put them apart in feature space.

Figure 5.5b shows a selection of products with their individual image features projected to 2D by T-SNE with a cosine distance metric. Most classes, even visually very similar products like the yoghurts of the

Zaanse Hoeve brand are well separable in cosine distance, save for a few outliers. However *23_Conimex_Kip_Kerrie_Madras* and *24_Conimex_Nasi_Speciaal* are barely separable in cosine distance. This could be attributed to their extreme visual similarity, even to the human eye, especially from further away. Both were training classes, but since training- and target classes are randomly selected, the probability of both being selected simultaneously is low. Moreover, considering that training images consist of a wide variety of image perspectives, the likelihood that for both classes example images from further away are selected is even lower. As such the model may not have had enough initiative to separate them in feature space, especially not for images from further away. So to answer the question: *How effectively does Product-ProtoNet distinguish class features in the feature space, as this is a crucial aspect for the effectiveness of helper prototypes?*, most products are very well separable on cosine distance in feature space, save for some outliers for which Product-ProtoNet might not have had enough initiative to put them apart in feature space.

### 5.3.5. Performance on requirements

This final results section answers the question: *How well does Product-ProtoNet perform on this paper's inference time, memory usage and accuracy requirements?* Arguably this is one of the most important sub-questions introduced in this chapter, as this determines whether Product-ProtoNet is suitable for Albert or not.

| Architecture | Helper Prototypes | Inference time / image (ms) | GPU memory usage (MiB) | Training Time (hr) |
|---|---|---|---|---|
| *P>M>F (5-shot,5-way)* | - | 65.55 | 1790 | 11 |
| *Product-ProtoNet* | 9 | 2.89 | 2460 | 8 |
| | 4 | 2.83 | 2462 | 8 |
| | 3 | 2.78 | 2460 | 8 |
| | 2 | 2.75 | 2554 | 8 |
| | 1 | 2.76 | 2464 | 8 |
| | 0 | 2.73 | 2496 | 8 |

**Table 5.2:** The inference time and GPU usage of Product-ProtoNet. The results of 5-shot, 5-way P>M>F (see chapter 4) have been added as a baseline. Comparing to multiple helper classes has no noticable effect on the inference time per image.

Product-ProtoNet's inference time-optimized deployment makes it an attractive choice for real-life deployment. It has an inference time of 2.73-2.83 ms (see table 5.2), 22 to 24-times lower than 5-shot, 5-way P>M>F. With any number of helper prototypes Product-ProtoNet passes the inference time requirement set by this paper: **The model's inference time should be as low as possible, but models with a high detection accuracy are preferred over those with a low inference time. To still have good closed-loop object tracking, this paper requires a maximum inference time of 3.6 ms per query image.** Table 5.2 shows that its memory usage also is consistently below the maximum of 4GB set by this paper in this requirement: **As a design decision, inference GPU memory usage should be below 4GB.**

| Architecture | Helper Prototypes | Accuracy in a realistic supermarket setting | |
|---|---|---|---|
| | | Seen Products | Unseen Products |
| *Product-ProtoNet* | 9 | 99.1% | 99.8% |
| | 4 | 98.7% | 97.4% |
| | 3 | 98.7% | 97.4% |
| | 2 | 98.4% | 95.2% |
| | 1 | 96.9% | 90.3% |
| | 0 | 92.7% | 75.8% |

**Table 5.3:** A tabular summary of the results in section 5.3.3. The accuracy of Product-ProtoNet on seen and unseen products in a realistic supermarket scenario.

To verify that Product-ProtoNet passes the seen and unseen accuracy requirements set in this paper:

**The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for product classes that are seen during training. As a design decision, this should be at least 90%.**; and **The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for novel product classes that are not seen during training. As a design decision, this should be at least 90%.**, the results in section 5.3.3, have been reformatted in tabular form (table 5.3). Product-ProtoNet passes the accuracy requirement for seen products without any helper prototypes, but needs at least one prototype to pass it for novel products. As this paper requires to the accuracy on both to be as high as possible, 9 helper prototypes seem to be most ideal for Product-ProtoNet in a realistic supermarket situation.

However, as discussed in the introduction of this paper, a faster yet slightly less accurate model is preferred over slower, more accurate ones. This is because Albert selects the product with the highest number of detections during a visual servoing period for picking. Since all of Product-ProtoNet's inference times fall between 2.73 and 2.89 ms, and the maximum required inference time for visual servoing is 3.6 ms, Product-ProtoNet will typically have one and sometimes two detections per visual servoing period for any number of prototypes.

In practical terms, this means that Albert often has only one detection to work with for any number of prototypes, making the accuracy of this single detection limiting to Albert's overall accuracy. Consequently, Product-ProtoNet with 9 helper prototypes, which has the highest accuracy of all, remains the preferred solution of this paper. Especially since with an inference time of 2.89 ms, a GPU memory usage of 2460 MiB and seen- and unseen accuracies of respectively 99.1% and 99.8% it also passes all requirements set in this paper gloriously.

## 5.4. Conclusion

This chapter tries to answer the following research question: **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?** This chapter takes the two most important concepts from P>M>F to create Product-ProtoNet: 1) A good pretrained feature extractor; and 2) Attributing query images to their most likely class, which P>M>F does with ProtoNet; Unlike from P>M>F, Product-ProtoNet can reliably separate non-target classes from target classes (section 5.3.1). That both Product-ProtoNet and P>M>F benefit from attributing query images to their most likely class becomes clear in section 5.3.4. In fact, Product-ProtoNet with helper prototypes, that essentially take up this function, has a similar accuracy to P>M>F. Furthermore it is clear that Product-ProtoNet benefits from having as much helper prototypes as visible non target products (section 5.3.3). Product-ProtoNet with 9 helper prototypes has an inference time of just 2.89 ms, a memory usage of 260 MiB and an unseen- and seen accuracy of respectively 99.1% and 99.8% in a realistic supermarket setting (section 5.3.5). This means that Product-ProtoNet is indeed a suitable model for Albert, that can detect non-classes and fits its requirements.

# 6

# Product-ProtoNet On Albert

With an inference time of 2.89 ms, a seen accuracy of 99.1%, an unseen accuracy of 99.8% and a memory usage lower than 4 GB, Product-ProtoNet seems the perfect model to work well on Albert. To evaluate Product-ProtoNet's accuracy on a dataset, distinguishing 1 product from 9 similar products has been deemed a realistic supermarket scenario. It is however impossible to design a realistic supermarket dataset evaluation method that perfectly covers all real world scenarios. Because of this, it is extremely important to assess Albert's real-world performance. This chapter evaluates this and answers this paper's final sub-question: **How well does this model work on Albert?**

## 6.1. Implementation of Product-ProtoNet on Albert

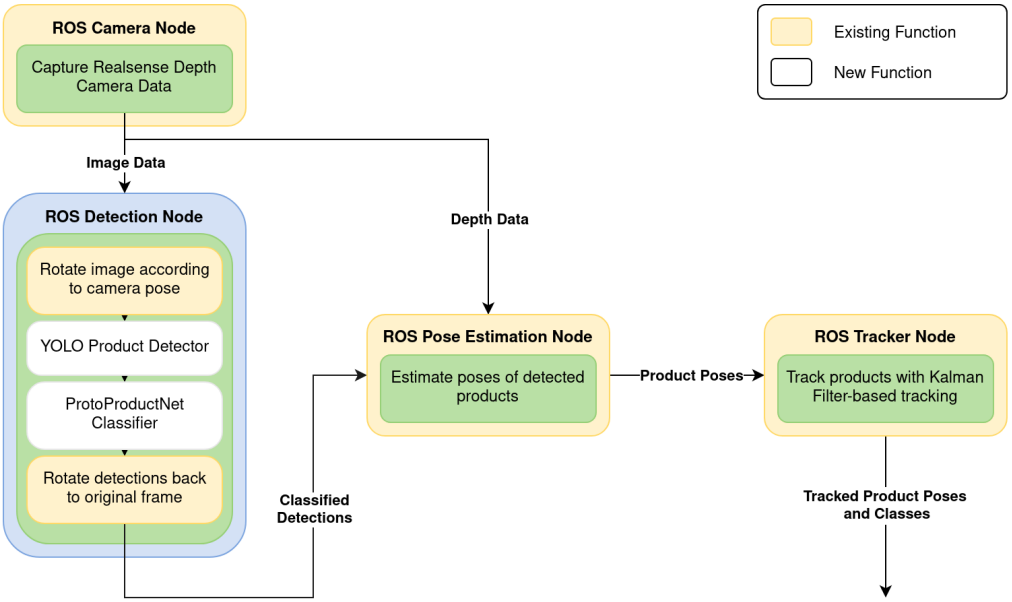


**Figure 6.1:** The general perception pipeline of the robot, based on [24]. This setup uses YOLO only to detect products and Product-ProtoNet to classify what those products are. As YOLO can not classify unseen classes, this setup is necessary to add new products without re-training the model.

Albert uses Product-ProtoNet combined with YOLO-V6.3 to predict the place of a product and if it matches a target class. Together they work in a Detection Node that is part of a bigger perception pipeline (figure 6.1). The YOLO Product Detector detects products in an image and outputs a bounding box for every

product. This bounding box is then cut out from the image and the cut-out of the product is then passed to Product-ProtoNet. Product-ProtoNet then classifies it as either the target picking class or not the target picking class. This classification combined with the original bounding box is then passed to a Pose Estimation Node that estimates the pose and location of products in 3D. A Tracker Node then collects all product poses and classes to track and update their 3D position estimation. It passes this information to Albert for picking purposes.

With respect to the previous perception implementation on Albert that made use of a YOLO-V6.3-nano object detector to predict product position and class, two major changes were made: 1) A new YOLO product detector was trained that only predicts product position; and 2) YOLO-detections of this new detector are classified by Product-ProtoNet instead of directly by YOLO; Because the new setup uses YOLO to only predict product position and not classify products, positional biases (e.g. milk is always at the top of the shelf) are less prevalent. Different products can be in many different positions on a shelf, which makes not one position preferred. Also, decoupling class from position makes it impossible for a model to predict classes based on their position in an image. Using Product-ProtoNet instead of YOLO to classify products also makes it possible to add new products easily, as the model does not need to be re-trained. Product-ProtoNet learns to classify images by comparing them to a target class. Simply adding images of a new target class is enough for classification, which can even be done on the fly. The code used for this new setup can be found here.

New products might also require a new picking strategy. A round can of tomatoes might require a very different grasp and handling than a bag of crisps and of course their weight also greatly varies. For this reason, the robot can also be taught new trajectories per product. This combination of classifying new products and the ability to learn a new picking strategy, makes it optimally adaptable to work in a supermarket environment where new products are added regularly to the contents of the store.

## 6.2. Experiment

With a YOLO-Product-ProtoNet-setup, Albert should be able to classify unseen products from very few example images. To test how well Albert can do so in real life, Product-ProtoNet is provided only one example image per product for 30 different unseen products (products are shown in figure 6.2 on the next page). Visually similar products are placed next to each other to make the simulation harder and product positions are changed regularly to make sure detection and grasping work from different perspectives. This paper then classifies the amount of times Albert goes to the correct product, and the amount of times Albert can succesfully pick this product. Both are vital for the functioning of Albert as a system, as they mean that Albert can identify and grasp products it has never seen before. Because Product-ProtoNet only influences the identification of products, this is the only part this paper will draw conclusions from. If this part works succesfully on Albert, it means that both object detection and visual servoing work correctly with Product-ProtoNet in a complex system.

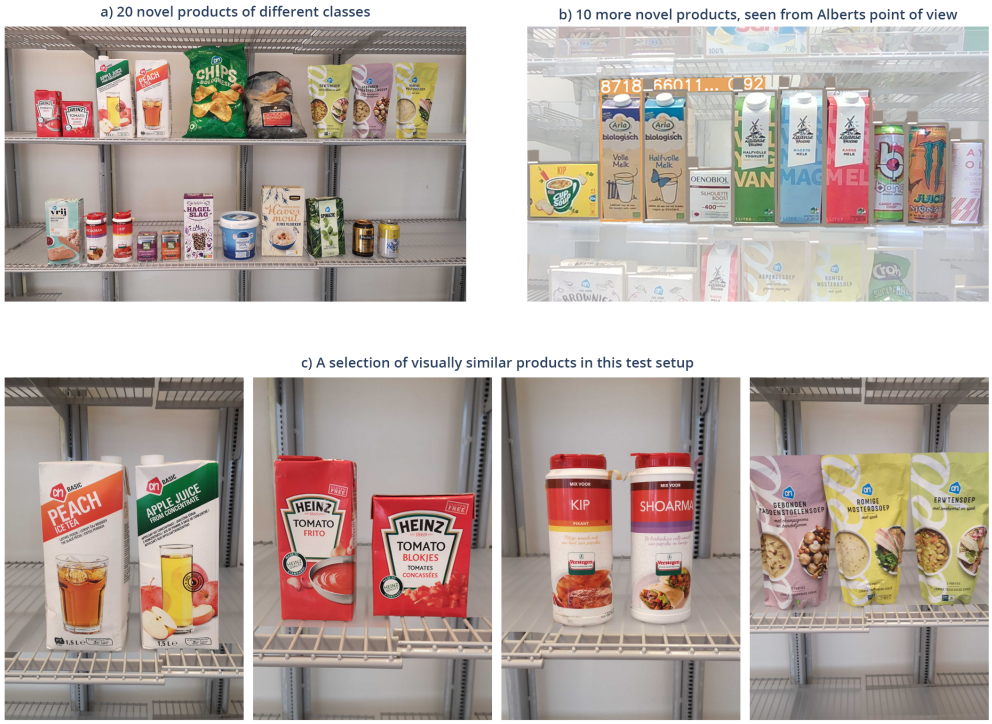


**Figure 6.2:** Products never seen during training that were used for the experiment. a) 20 novel products; b) 10 more novel products; and c) a selection of some of the visually similar products in this test set; This diverse and challenging test set aims to be as close as possible to a real supermarket situation.

## 6.3. Results

Table 6.1 shows how well Albert performs when a YOLO-V6.3 object detector is combined with Product-ProtoNet. Product-ProtoNet only influences the ”go to correct product”-task. With 90 tries, this task has an exceptional success rate of 0.97. With just one example image, Product-ProtoNet is able to identify a product consistently and steer Albert to the exact right product. This means that product detection works well and that Product-ProtoNet in the perception pipeline shown in figure 6.1 works fast enough for visual servoing.

| Subtask | Succes rate |
|---|---|
| Go to correct product | 0.97 |
| Grasp and collect product | 0.68 |

**Table 6.1:** The ”go to correct product”-task, the only task that Product-ProtoNet has influence on, has an extremely high succes rate of 1.0. Albert sometimes struggles with picking the product however, which is why the ”grasp and collect product”-succes rate is significantly lower.

The ”grasp and collect product” success rate of 0.68 shows that teaching the robot a new grasp is possible, even though Albert is not perfect and sometimes struggles with picking a detected product. However, since Product-ProtoNet has no influence on this task, it is only possible to acknowledge that teaching the robot a new grasp is possible, without drawing any further conclusions from it.

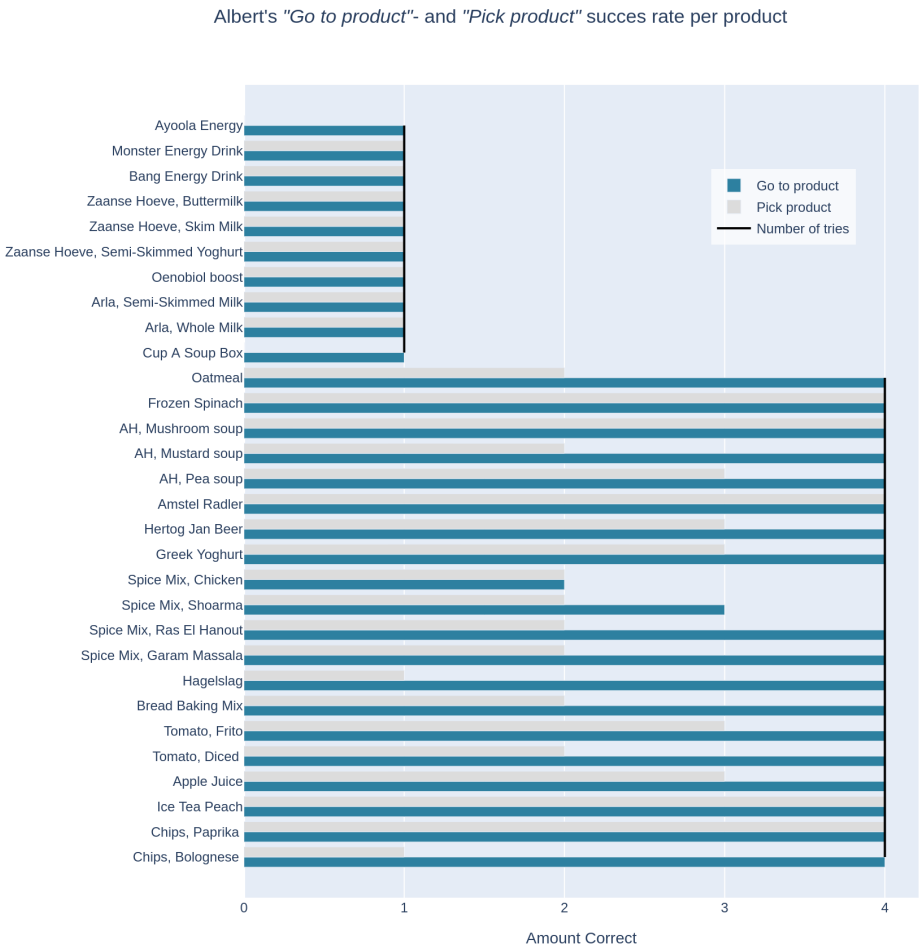


**Figure 6.3:** The amount of successful ”go to product”- and ”pick product” actions in comparison to the number of tries per product. Product-ProtoNet struggles only with detecting *Spice Mix, Chicken* and *Spice Mix, Shoarma*. During the experiment it became clear that they are often misidentified as each other.


**Figure 6.4:** a) The misidentified product pair; and b) A selection of training classes; Product-ProtoNet’s training likely makes it extract image features that are less informative for packaging designs with subtle color differences.

Even though the ”go to correct product”-action is done correctly for 97% of the tries, it is informative to identify what causes the few misdetections still made by Albert with Product-ProtoNet. In order to do this, all individual products with the amount of successful actions done by Albert have been visualized in figure 6.3. Most products are identified correctly every time, but two products stand out due to their misdetections: *Spice Mix, Chicken* and *Spice Mix, Shoarma*. These products are visually very similar (see figure 6.4) and during the experiment it became clear that Albert often confused the two. Albert would often pick the spice mix closest to the starting position of the arm. While to humans these spice mixes are relatively easily distinguishable (they have a colored band on the packaging that corresponds to the flavor) Albert often misidentifies them. Interestingly, products that might seem harder to distinguish, such as *Tomato, Frito* and *Tomato Diced* were always correctly classified. Possibly Product-ProtoNet has not learned that small changes in package coloring - such as a different-colored band for different products - correspond to different products, as many products in the training set have distinct colors for different product classes of the same brand (see figure 6.4). Another possibility is that Product-ProtoNet might not have had enough incentive to learn different representations for very similar looking classes, as it has been trained on random classes, and picking two random products from a supermarket often yields two entirely different products. In any case, this test contains only two products that are often misidentified, which is very little to draw any significant conclusions from. It is clear that Product-ProtoNet works extremely well in most of the cases, even for very similar products like *Tomato, Frito* and *Tomato Diced*, but that specifically identifying *Spice Mix, Chicken* and *Spice Mix, Shoarma* is sometimes hard.

## 6.4. Conclusion

To answer the question: **How well does this model work on Albert?**, Product-ProtoNet was deployed on Albert to classify images fed to it by a YOLO product detector. Albert was tested to identify 30 different products with just one example image. Of all 30 products tested, only 2 were hard to detect in any configuration. Likely this has to do with the specific way that Product-ProtoNet is trained. Yet still Product-ProtoNet functions amazingly on Albert and Albert achieves a ”go to correct product”-rate of 0.97. This means that product detection works well and that this implementation of Product-ProtoNet in Albert’s perception pipeline is fast enough for visual servoing.

# 7

# Discussion

There are always multiple ways to solve a problem. This paper makes specific decisions to create a product detector that can detect new products without having to be re-trained. To answer the research question of this paper: **What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?**, multiple approaches are possible however. This chapter will discuss some of them.

## 7.1. Selecting similar classes afterwards or during training

Product-ProtoNet works by training a sigmoid classifier to tell if query classes are the picking target class or not. During deployment, Product-ProtoNet selects the most similar classes to help pinpoint what class a query image is most similar to, if any. Note that this is exactly the opposite of training on very similar classes. Instead of learning how to tell similar classes apart during training, Product-ProtoNet does this during inference with helper classes. This method likely relies on a good pre-trained backbone to extract meaningful features with a limited amount of training.

Training on similar classes may have some positive effect when it comes to inference time. Even though inference time is only slightly affected by choosing helper prototypes in Product-ProtoNet (table 5.2), Product-ProtoNet tests this on a relatively small dataset (AholdSet-V2). Likely there will be a more visible effect on bigger datasets. So training on similar classes might be beneficial for inference time.

Training on similar classes however significantly slows down the training process. If a closest class selector has to select the closest classes for every iteration, the mean features of all classes need to be calculated using a frozen feature extractor from the previous iteration. The classes that are closest together will then be selected as model input. Especially for large datasets this operation is extremely time consuming. However, next to the increased training time, the risk to overfit on seen classes becomes higher when pre-selecting similar classes during training. The model is explicitly told to extract features that tell similar training images apart and might not to learn a general classification strategy for telling all classes apart. Training a model like this might force it to learn features that tell a Conimex spice mix apart from another Conimex mix and do very well on seen classes. However, as this model has for example not seen any spice mixes of another brand, it might not understand that spice mixes of that brand don't belong to the same class. Consequently, the model may have poorer generalization on unseen classes, matching different- branded spice mixes to their likeliest option might again be the best approach to increase accuracy.

In summary, class selection during training has the potential to further reduce inference time and increase accuracy on similar classes. However, it is certain to increase training time, has a higher risk of overfitting on seen close classes and might still need helper classes to distinguish unseen classes. It

would be interesting to explore this method further and compare class pre-selection to Product-ProtoNet with helper classes. However, the simplicity of Product-ProtoNet's basic method shows that it is not necessary for a well-performing model.

## 7.2. Does training with a sigmoid classifier actually help?

As helper prototypes have such a significant effect on the performance of Product-ProtoNet it is debatable if a sigmoid classifier actually does much when it comes to meta-training a pre-trained feature extractor, or that just a pre-trained classifier is already enough. After all, a good pre-trained feature extractor might already produce features that are informative enough to distinguish different classes (which is also clear for P>M>F [16]). Especially since Product-ProtoNet trains on random classes, most of them might even already be far enough apart in feature space to be distinguished from target classes.

This also becomes clear by the fact that Product-ProtoNet performs slightly worse on seen classes than on unseen classes when using helper prototypes (see figure 5.4). It means that a model that has been trained on the seen dataset, has in fact learned to separate seen features less well than unseen features. This can very well indicate that the pre-training of the feature extractor plays an important role and that the unseen features just happen to be better extractable with it than the seen features. The performance of Product-ProtoNet with only a pre-trained feature extractor will be extremely valuable to validate in future research.

Instead of a learnable sigmoid classifier, a soft cosine distance decision boundary might already be enough to determine whether a query image belongs to a class. In fact, choosing a tight decision boundary during training might even force a feature extractor to put features of the same class close together. This decision boundary would become a hyperparameter instead of a learnable function.

In short, the effectiveness of a learnable sigmoid classifier is debatable and it may not be the best tool to train a model to separate features of different classes. A user-defined smooth decision boundary could instead be used for deciding whether query features belong to a target class and likely be just as effective. This hyperparameter might even have training benefits, as it can be used to force features of the same class to be encoded within a certain distance of that class. However, it does not seem necessary for a well-working model, as the accuracy of Product-ProtoNet on a realistic supermarket scenario is already 99.1% for seen products and 99.8% for novel products.

# 8

# Conclusion

Albert is a useful supermarket order picking robot that collects shoppers' online orders and delivers them to a pickup section in a supermarket. This makes going to the supermarket an easy task and saves precious time. An unmissable part of Albert are its' product detection and localization abilities. Currently the robot uses a YOLO-V6.3-nano product detector that predicts both a products' location and class. Associated with this detector are two major problems: 1) It has noticeable positional biases and can not always identify products that are in a different place than usual; and 2) It is impossible to add new products without re-training the whole model. Especially in a supermarket with an ever changing stock, the latter is a major problem.

To solve the first problem, product localization is split from product classification and a different model is used for both. This means that product class is independent of product location and locational biases will be less prevalent. The focus of this paper is on the second problem however: adding new products without re-training the whole model. For this few-shot classifiers are the ideal solution. They classify products based on only a few images, which makes collecting product data much more manageable. As few-shot classifiers only learn to predict similarity between query and target images, adding new products is as easy as providing different target images. In order to find a few-shot classifier that can identify if query images are Alberts' target picking class, this paper asks the following question:

> **"What few-shot classifier can identify products in a supermarket environment, is able to detect non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?"**

There are many few-shot classifiers, and in order to pick the optimal few-shot classification method for Albert, this paper sets the following requirements:

1. *The model's inference time should be as low as possible, but models with a high detection accuracy are preferred over those with a low inference time. To still have good closed-loop object tracking, this paper requires a maximum inference time of 3.6 ms per query image.*
2. *The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for product classes that are seen during training. As a design decision, this should be at least 90%.*
3. *The preferred few-shot method should have the highest possible accuracy in a realistic supermarket setting for novel product classes that are not seen during training. As a design decision, this should be at least 90%.*
4. *As a design decision, inference GPU memory usage should be below 4GB.*

Two very promising few-shot models are TRIDENT and P>M>F. They have respective accuracies of 95.95% and 98.4% on miniImageNet, a dataset where most few-shot models are evaluated on. Both TRIDENT

and P>M>F outperform all other models, which makes them the perfect candidates to do well on Albert. However, like all few-shot models, P>M>F and TRIDENT assume queries exclusively belong to a specific set of classes, with no possibility of falling outside this set. However, from a safety perspective it is crucial to be able to classify non-products instead of attributing them to one of the products in a specific set. Particularly when Albert's product localization module mislabels humans as products, classifying humans as not the target picking class is a must.

For this reason, this paper first evaluates whether P>M>F or TRIDENT is a better base for building a classifier that can classify query images as non-target- or target picking classes, and evaluates both models against the paper's requirements. Secondly, this paper uses the key concepts of the best classifier to develop a model that can discern whether query images belong to a target product class or not. Thirdly, this paper evaluates how well this model, that meets all the requirements for integration on Albert, performs when actually integrated on the robot. To this end, the research question is split into three subquestions:

1. **Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?**
2. **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?**
3. **How well does this model work on Albert?**

To answer: **Which of the current state-of-the-art few-shot classification models TRIDENT and P>M>F can be used best as a classifier for Albert?**, both TRIDENT and P>M>F are evaluated on the requirements of this paper. Both models meet this paper's GPU memory usage requirement, but P>M>F consistently outperforms TRIDENT in terms of seen- and unseen accuracy. Both TRIDENT and P>M>F are unsuitable for a supermarket robot however, due to their high inference times that make them unfit for visual servoing. However, P>M>F only uses the mean of class example image features to construct what they call class prototypes to compare query images to. As these prototypes will not change during deployment, they can be calculated beforehand, which might greatly reduce inference time. This makes P>M>F the preferred few-shot model as a base for Albert, especially since it consistently has a 5% to 15% better accuracy than TRIDENT. P>M>F can however not classify non-classes and therefore has to be adapted to do this.

With P>M>F as best base-classifier, this paper answers the question: **If this classifier does not meet all requirements or can not classify non-target classes, how can it be modified to be fit for Albert?** This paper identifies two reasons why P>M>F works so well: 1) It uses good pre-trained feature extractor; and 2) It compares query images to a set of prototypes and matches only to the likeliest. P>M>F uses a ProtoNet model for classification that essentially does this; Based on these main principles of P>M>F, this paper proposes a new few-shot model: Product-ProtoNet. It employs a ViT pre-trained with DINO to extract features, classifies query images bases on their distance to prototypes with a learned threshold - enabling it to distinguish classes and non-classes - and utilizes helper prototypes to distinguish classes that are close together in feature space in a ProtoNet-like manner. Experiments with Product-ProtoNet show that it: 1) Indeed classifies non-classes correctly; 2) Has an inference time of only 2.89 ms and passes this papers memory usage requirement; and 3) Achieves impressive seen- and unseen accuracies of respectively 99.1% and 99.8% in a realistic supermarket setting; Product-ProtoNet passes all of this paper's requirements easily.

This leaves the question: **How well does this model work on Albert?** Experiments on Albert show that Product-ProtoNet is suited for visual servoing and correctly tracks and goes to the right product in a shelf with a succes rate of 0.97. With only one example image per product this is truly impressive and it is clear that this model works very well for Albert's specific requirements.

In short, this paper creates a new few-shot model that can determine whether a query image classifies as a target product or not, meets Alberts' requirements and has excellent performance when deployed on Albert: Product-ProtoNet. Thus Product-ProtoNet is this papers' answer to the research question: **What few-shot classifier can identify products in a supermarket environment, is able to detect**

**non-target classes, and meets the requirements of deployment on a robotic platform like Albert best?**

# 9 Future Work

As mentioned in the discussion, there is still valuable research to be done with Product-ProtoNet that could improve its inference time and ability to distinguish similar classes. This chapter will reiterate some of these concepts, as well as introduce some new research directions that could improve the functioning of Albert is some way.

## 9.1. Architectural improvements of Product-ProtoNet

Modifying Product-ProtoNet's architecture, as mentioned in the discussion, could improve inference speed and class separation. Two promising approaches are: 1) Pre-selecting similar classes during training, which may reduce inference time but have limited impact on unseen classes. 2) Employing a soft decision boundary instead of a sigmoid classifier, potentially improving model performance by encouraging class features to cluster closely and non-classes to separate further. Testing both methods can deepen the understanding of Product-ProtoNet. Both methods aim to reduce the reliance on helper prototypes, which speeds up inference, and boost accuracy.

## 9.2. Pruning Product-ProtoNet's backbone

Product-ProtoNet uses a ViT pre-trained with DINO as a feature extractor. To improve inference time, at the cost of a slightly worse accuracy this vision transformer could be pruned. Pruning removes connections and parameters that are deemed less important for a networks' performance. Pruning can reduce the floating point operations per second (FLOPs) needed by as much as 64%, while reducing the accuracy of a model only by a few percent [64].

## 9.3. Feature Matching

Feature matching is a simple alternative to few-shot learning that does not need a deep neural network or any training time [28]. Key points are identified on a reference image and matched to key points found in the image the robot camera captures. Feature matching might work well for the majority of products in a supermarket and save inference time and computational complexity.

For some supermarket products, e.g. non-rigid or rotated objects, however, this is a challenging task. A chips bag that is crumpled up or a peanut butter jar that has been rotated so that only the rear label is visible, likely will give no or very little matches. Deep learning on the other hand, might learn that the rear label of a peanut butter jar belongs to the same class as the front of it, and be more accurate in edge cases. Overall deep learning comes at an additional computing power cost, but achieves better accuracy than traditional computer vision methods like feature matching [28]. For this reason this paper has focused solely on deep learning methods, but it would be interesting to compare them to traditional methods like feature matching in future research. Especially since fast methods with lower accuracies

might eventually produce better results for visual servoing than slow methods with higher accuracies.

## 9.4. Segmentation vs Bounding boxes

Currently the position of products is predicted by a YOLO-V6-classifier. However for grasp prediction it might be nice to have a cutout that corresponds with the shape of the object instead of a rectangular bounding box like YOLO predicts. For this, using SAM (Segment Anything) [22] might be interesting, as this can be given the prompt - or be finetuned - to segment products, or can be used in combination with YOLO and Product-ProtoNet to cut out products from bounding boxes with the target class without any finetuning. However other semantic segmentation models like U-net [42], Mask-RCNN [15], YOLO-V8 [53] or FastSAM [63] can be used. Sadly FastSAM, the fastest of these classifiers already has an inference time of 40 ms on an NVIDIA GeForce RTX 3090 GPU [63] for a single MS COCO image with a median resolution of 640 x 480, which is too much for visual servoing. However, segmenting only small portions of an image, detected with YOLO-V6 and classified as target product with Product-ProtoNet might still be a viable option to get better grasp proposals and use visual servoing.

## 9.5. Fine-tuning or re-training YOLO

The YOLO-V6.3-nano detector model used for tests on Albert was trained on a zoom-augmented version of SKU and not fine-tuned for this papers' specific usecase. Likely there is still much to improve in terms of product bounding box detection. This was also observed during experiments, when product bounding boxes could not be detected in certain configurations. Fine-tuning or re-training this model on an unlabeled dataset with product annotations of Albert's specific environment would likely yield much better detections.

## 9.6. Predicting class and location together

This paper has specifically chosen to decouple class and location prediction as this reduces the positional bias for certain product classes. However, with a very balanced dataset, or with a architecture that needs very little fine-tuning it might be interesting to explore combined object detector and classifiers like Owl-ViT [30] and SEG-GPT [57]. Both can be used in a one-shot setting and segment or detect a query image in an image. Unofficial sources report an inference time of 300ms per query for Owl-ViT [45], but in order to verify if any of these models could be used in practice, they would both need to be tested.

# References


[1] Dec. 2023. URL: https://nieuws.ah.nl/in-nieuw-home-shop-center-van-albert-heijn-doen-robots-de-boodschappen/.

[2] AlexeyAB. *GitHub - AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet )*. URL: https://github.com/AlexeyAB/darknet.

[3] Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. "How to train your MAML". In: *CoRR* abs/1810.09502 (2018). arXiv: 1810.09502. URL: http://arxiv.org/abs/1810.09502.

[4] Hangbo Bao, Li Dong, and Furu Wei. "BEiT: BERT Pre-Training of Image Transformers". In: *CoRR* abs/2106.08254 (2021). arXiv: 2106.08254. URL: https://arxiv.org/abs/2106.08254.

[5] Yassir Bendou et al. "EASY: Ensemble Augmented-Shot Y-shaped Learning: State-Of-The-Art Few-Shot Classification with Simple Ingredients". In: *CoRR* abs/2201.09699 (2022). arXiv: 2201.09699. URL: https://arxiv.org/abs/2201.09699.

[6] Qi Cai et al. "Memory Matching Networks for One-Shot Image Recognition". In: *CoRR* abs/1804.08281 (2018). arXiv: 1804.08281. URL: http://arxiv.org/abs/1804.08281.

[7] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV].

[8] Xiangyu Chen and Guanghui Wang. "Few-Shot Learning by Integrating Spatial and Frequency Representation". In: *CoRR* abs/2105.05348 (2021). arXiv: 2105.05348. URL: https://arxiv.org/abs/2105.05348.

[9] Douglas Chiguvi and Thuso Sepepe. "An Assessment of Customer Perceptions Towards Product Packaging Design Changes: Insights from the Botswana Fast-Moving Consumer Goods Business". In: *Journal of Emerging Trends in Marketing and Management* 1.2 (2023), pp. 46–55.

[10] 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.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *CoRR* abs/1703.03400 (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[12] Klaus Fuchs, Tobias Grundmann, and Elgar Fleisch. "Towards Identification of Packaged Products via Computer Vision: Convolutional Neural Networks for Object Detection and Image Classification in Retail Environments". In: *Proceedings of the 9th International Conference on the Internet of Things*. IoT '19. Bilbao, Spain: Association for Computing Machinery, 2019. ISBN: 9781450372077. DOI: 10.1145/3365871.3365899. URL: https://doi.org/10.1145/3365871.3365899.

[13] Victor Garcia and Joan Bruna. *Few-Shot Learning with Graph Neural Networks*. 2018. arXiv: 1711.04043 [stat.ML].

[14] Eran Goldman et al. "Precise Detection in Densely Packed Scenes". In: *CoRR* abs/1904.00853 (2019). arXiv: 1904.00853. URL: http://arxiv.org/abs/1904.00853.

[15] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[16] Shell Xu Hu et al. *Pushing the Limits of Simple Pipelines for Few-Shot Learning: External Data and Fine-Tuning Make a Difference*. 2022. arXiv: 2204.07305 [cs.CV].

[17] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. “Leveraging the Feature Distribution in Transfer-based Few-Shot Learning”. In: *CoRR* abs/2006.03806 (2020). arXiv: 2006.03806. URL: https://arxiv.org/abs/2006.03806.

[18] Yuqing Hu, Stéphane Pateux, and Vincent Gripon. “Adaptive dimension reduction and variational inference for transductive few-shot classification”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 5899–5917.

[19] S. Hutchinson, G.D. Hager, and P.I. Corke. “A tutorial on visual servo control”. In: *IEEE Transactions on Robotics and Automation* 12.5 (1996), pp. 651–670. DOI: 10.1109/70.538972.

[20] Shruti Jadon. “An Overview of Deep Learning Architectures in Few-Shot Learning Domain”. In: *CoRR* abs/2008.06365 (2020). arXiv: 2008.06365. URL: https://arxiv.org/abs/2008.06365.

[21] Muhammad Abdullah Jamal, Guo-Jun Qi, and Mubarak Shah. “Task-Agnostic Meta-Learning for Few-shot Learning”. In: *CoRR* abs/1805.07722 (2018). arXiv: 1805.07722. URL: http://arxiv.org/abs/1805.07722.

[22] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].

[23] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. “Siamese neural networks for one-shot image recognition”. In: *ICML deep learning workshop*. Vol. 2. 1. Lille. 2015.

[24] Stijn Lafontaine. *Robotic Grasping from Supermarket Shelves using Visual Servoing*. 2023. URL: http://resolver.tudelft.nl/uuid:4fb4a898-0980-477a-862a-c51e3bac3ce5.

[25] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), pp. 1332–1338. DOI: 10.1126/science.aab3050. eprint: https://www.science.org/doi/pdf/10.1126/science.aab3050. URL: https://www.science.org/doi/abs/10.1126/science.aab3050.

[26] Chuyi Li et al. *YOLOv6 v3.0: A Full-Scale Reloading*. 2023. arXiv: 2301.05586 [cs.CV].

[27] Wenbin Li et al. “Revisiting Local Descriptor based Image-to-Class Measure for Few-shot Learning”. In: *CoRR* abs/1903.12290 (2019). arXiv: 1903.12290. URL: http://arxiv.org/abs/1903.12290.

[28] Niall O’ Mahony et al. “Deep Learning vs. Traditional Computer Vision”. In: *CoRR* abs/1910.13796 (2019). arXiv: 1910.13796. URL: http://arxiv.org/abs/1910.13796.

[29] Martha. *30 000 Different Products And Counting: The Average Grocery Store*. Feb. 2022. URL: https://www.icsid.org/uncategorized/how-many-products-are-in-a-typical-grocery-store/.

[30] Matthias Minderer et al. *Simple Open-Vocabulary Object Detection with Vision Transformers*. 2022. arXiv: 2205.06230 [cs.CV].

[31] Nikhil Mishra et al. “A Simple Neural Attentive Meta-Learner”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=B1DmUzWAW.

[32] Tsendsuren Munkhdalai and Hong Yu. “Meta Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 2554–2563. URL: https://proceedings.mlr.press/v70/munkhdalai17a.html.

[33] Mateusz Ochal et al. *Class Imbalance in Few-Shot Learning*. 2021. URL: https://openreview.net/forum?id=j0yLJ-MsgJ.

[34] Archit Parnami and Minwoo Lee. *Learning from Few Examples: A Summary of Approaches to Few-Shot Learning*. 2022. arXiv: 2203.04291 [cs.LG].

[35] Jingtian Peng et al. “RP2K: A Large-Scale Retail Product Dataset for Fine-Grained Image Classification”. In: *CoRR* abs/2006.12634 (2020). arXiv: 2006.12634. URL: https://arxiv.org/abs/2006.12634.

[36] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *CoRR* abs/2103.00020 (2021). arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.

[37] Shafin Rahman, Salman Hameed Khan, and Fatih Porikli. “A Unified approach for Conventional Zero-shot, Generalized Zero-shot and Few-shot Learning”. In: *CoRR* abs/1706.08653 (2017). arXiv: 1706.08653. URL: http://arxiv.org/abs/1706.08653.

[38] Jathushan Rajasegaran et al. “iTAML: An Incremental Task-Agnostic Meta-learning Approach”. In: *CoRR* abs/2003.11652 (2020). arXiv: 2003.11652. URL: https://arxiv.org/abs/2003.11652.

[39] Aravind Rajeswaran et al. “Meta-Learning with Implicit Gradients”. In: *CoRR* abs/1909.04630 (2019). arXiv: 1909.04630. URL: http://arxiv.org/abs/1909.04630.

[40] Sachin Ravi and Hugo Larochelle. “Optimization as a Model for Few-Shot Learning”. In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=rJY0-Kcll.

[41] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[43] Andrei A. Rusu et al. “Meta-Learning with Latent Embedding Optimization”. In: *CoRR* abs/1807.05960 (2018). arXiv: 1807.05960. URL: http://arxiv.org/abs/1807.05960.

[44] Adam Santoro et al. “One-shot Learning with Memory-Augmented Neural Networks”. In: *CoRR* abs/1605.06065 (2016). arXiv: 1605.06065. URL: http://arxiv.org/abs/1605.06065.

[45] Segments.ai. *Zero-shot Object Detection with Owl-ViT*. 2024. URL: https://segments.ai/blog/zero-shot-object-detection-with-owl-vit/ (visited on 02/26/2024).

[46] Marcin Sendera et al. *HyperShot: Few-Shot Learning by Kernel HyperNetworks*. 2022. arXiv: 2203.11378 [cs.LG].

[47] Daniel Shalam and Simon Korman. *The Self-Optimal-Transport Feature Transform*. 2022. arXiv: 2204.03065 [cs.CV].

[48] Xiahan Shi et al. “Relational Generalized Few-Shot Learning”. In: *CoRR* abs/1907.09557 (2019). arXiv: 1907.09557. URL: http://arxiv.org/abs/1907.09557.

[49] Anuj Singh and Hadi Jamali-Rad. *Transductive Decoupled Variational Inference for Few-Shot Classification*. 2022. arXiv: 2208.10559 [cs.CV].

[50] Jake Snell, Kevin Swersky, and Richard S. Zemel. “Prototypical Networks for Few-shot Learning”. In: *CoRR* abs/1703.05175 (2017). arXiv: 1703.05175. URL: http://arxiv.org/abs/1703.05175.

[51] Yisheng Song et al. *A Comprehensive Survey of Few-shot Learning: Evolution, Applications, Challenges, and Opportunities*. 2022. arXiv: 2205.06743 [cs.LG].

[52] Flood Sung et al. “Learning to Compare: Relation Network for Few-Shot Learning”. In: *CoRR* abs/1711.06025 (2017). arXiv: 1711.06025. URL: http://arxiv.org/abs/1711.06025.

[53] Ultralytics. *Ultralytics YOLO-V8*. URL: https://github.com/ultralytics/ultralytics.

[54] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[55] Markus Vincze and Gregory D. Hager. “Robust Image Processing and PositionBased Visual Servoing”. In: *Robust Vision for Vision-Based Control of Motion*. 2000, pp. 163–201. DOI: 10.1109/9780470546369.ch13.

[56] Oriol Vinyals et al. “Matching Networks for One Shot Learning”. In: *CoRR* abs/1606.04080 (2016). arXiv: 1606.04080. URL: http://arxiv.org/abs/1606.04080.

[57] Xinlong Wang et al. *SegGPT: Segmenting Everything In Context*. 2023. arXiv: 2304.03284 [cs.CV].

[58] Yan Wang et al. “SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning”. In: *CoRR* abs/1911.04623 (2019). arXiv: 1911.04623. URL: http://arxiv.org/abs/1911.04623.

[59] Yuchen Wei et al. “Deep learning for retail product recognition: Challenges and techniques”. In: *Computational intelligence and neuroscience* 2020 (2020).

[60] W.J. Wilson. “Visual Servo Control of Robots Using Kalman Filter Estimates of Relative Pose”. In: *IFAC Proceedings Volumes* 26.2, Part 3 (1993). 12th Triennal Wold Congress of the International Federation of Automatic control. Volume 3 Applications I, Sydney, Australia, 18-23 July, pp. 633–638. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)48804-5. URL: https://www.sciencedirect.com/science/article/pii/S1474667017488045.

[61] Congying Xia, Caiming Xiong, and Philip Yu. “Pseudo Siamese Network for Few-shot Intent Generation”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’21. <conf-loc>, <city>Virtual Event</city>, <country>Canada</country>, </conf-loc>: Association for Computing Machinery, 2021, pp. 2005–2009. ISBN: 9781450380379. DOI: 10.1145/3404835.3462995. URL: https://doi.org/10.1145/3404835.3462995.

[62] Ling Yang et al. “DPGN: Distribution Propagation Graph Network for Few-Shot Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[63] Xu Zhao et al. *Fast Segment Anything*. 2023. arXiv: 2306.12156 [cs.CV].

[64] Mingjian Zhu et al. “Visual Transformer Pruning”. In: *CoRR* abs/2104.08500 (2021). arXiv: 2104.08500. URL: https://arxiv.org/abs/2104.08500.