Handling the unknown

Towards on-the-job Object Recognition

David ter Kuile

Delft University of Technology

TU Delft

Handling the unknown

Towards on-the-job Object Recognition

by

David ter Kuile

to obtain the degree of Master of Science in Mechanical Engineering
at the Delft University of Technology,
to be defended on October 27, 2021 at 14:00.

Student number: 4310268
Project Duration: October, 2020 - October, 2021

Thesis committee:

Dr. ing. Jens Kober	Supervisor
Dr. Osama Mazhar	Daily supervisor
Dr. Wei Pan	

TU Delft

Abstract

As robots are becoming a more integral part in our daily lives, it is important to ensure they work in a safe and efficient manner. A large part of perceiving the environment is done through robot vision. Research in computer vision and machine learning lead to great improvements in the past decades and robots are able to outperform humans on certain tasks. However, these tasks are often in closed set condition. This makes translation to a real world application challenging, as this is in open set condition. The open set condition implies that incomplete knowledge of the world is available at training time. It is important for a robot, or agent, to be aware of this limitation. In vision tasks this is defined as open world recognition and allows an agent to detect and incrementally learn unknown objects.

The key contributions of this report are an autonomous data collection protocol for synthetic data creation, the open world algorithm Learning to Accept Image Classes (L2AIC) and an on-the-job recognition approach that combines open world recognition with autonomous data collection. L2AIC is a deep meta-learning model that classifies objects by comparing it to its memory in an n-way k-shot manner. New classes can be incrementally added to the memory without needing to retrain the model. The autonomous data collection protocol consists of two steps. First, a 3D model is reconstructed from an unknown object with an RGB-D camera. Secondly, from this 3D model a synthetic dataset is created that is added to the memory of L2AIC.

Results show that the on-the-job recognition approach is successful in learning to recognize a single unknown object using the L2AIC model with small-fc architecture an ResNet152 encoder. The encoder is loaded with pretrained weights on the ImageNet dataset. No additional fine-tuning is required, this has an adverse effect on the performance. Using the autonomous data collection protocol two datasets were created, varying the distance from the camera to the object. It was found that the synthetic dataset containing close-up images achieves the best performance. The performance of L2AIC with this close-up synthetic dataset is similar to using a dataset of actual images of the object. This means that for a single object it currently is not efficient to create a synthetic dataset. A more extensive study is required to compare performance of both datasets when increasing the number of encountered objects. Finally, the on-the-job approach shows it is capable of recognizing other instances of the same object class, using only the dataset of a single instance. This shows the on-the-job recognition approach is able to generalize well. Future work could be focused on improving this generalization with, for example, the help of Generative Adversarial Networks (GANs).

Nomenclature

Abbreviations

Abbreviation	Definition
L2AIC	Learning to Accept Image Classes
L2AC	Learning to Accept Classes
GAN	Generative Adversarial Network
KB	Knowledge base
AN	AlexNet
EN	EfficientNet
RN50	ResNet50
RN152	ResNet152
FC7	Feature layer used for feature extraction
ICP	Iterative Closest Point
ToF	Time-of-flight
WI	Wilderness Impact

Symbols

Symbol	Definition
$\mathcal{C}^{\mathcal{K}}$	The set of classes known to the classifier at training time
$\mathcal{C}^{\mathcal{N}}$	The set of classes known to the classifier, but not used in training
$\mathcal{C}^{\mathcal{U}}$	The set of unknown classes
$\mathcal{C}^{\mathcal{M}}$	The set of classes in the memory of L2AIC/L2AC
$\mathcal{R}_{\mathcal{O}}$	Open space risk
$\mathcal{R}_{\mathcal{E}}$	Empirical risk
n	Number of classed used for comparison by L2AIC/L2AC
k	Number of samples per class used for comparison by L2AIC/L2AC
h	Classification threshold of L2AIC/L2AC
x_i	Feature vector i
r_k	Similarity score predicted by the matching layer
f_{sim}	Similarity function
$\bar{x}^c$	Average feature vector of class c
$\mathbf{r}_c$	Position of the virtual camera with respect to the object
$\mathbf{r}_o$	Position of the object
$\mathbf{q}_o$	Orientation of the object
ϵ_K	Known classification error
ϵ_U	Unknown classification error
ϵ_{OW}	Open world error
$\mathcal{D}^{photos,memory}$	Dataset with images of teapot during reconstruction process
$\mathcal{D}^{photos,test}$	A test dataset with photos of a teapot
$\mathcal{D}^{syn}$	Synthetic dataset with images of a teapot
$\mathcal{D}^{syn,close}$	Synthetic dataset with close-up images of a teapot
$\mathcal{D}^{Tiny,test}$	The test dataset of TinyImageNet containing only the teapot class
$\mathcal{D}^{Tiny,memory}$	The train dataset of TinyImageNet containing only the teapot class

Contents

List of Figures

List of Tables

1 Introduction

Robots are becoming a more integral part of our daily lives. Autonomous vacuum cleaners or lawn mowers are already deployed for simple household tasks, while more complex robotic systems are being rapidly developed. Current self driving vehicles are already capable of handling many difficult tasks in traffic. At the same time humanoid robots, such as Atlas [8] and Talos [49], are getting more agile and human-like. For these robots, autonomy is a critical requirement in order to perform tasks in a safe and efficient manner. To achieve this autonomy, the robot, or agent, uses sensors to perceive the world. Analogue to humans, a crucial part of this perception is done through vision. Research in computer vision has seen some major breakthroughs the last decades, leading to computers outperforming humans on certain vision tasks. However, as pointed out by [66] and [4], computer vision research has been focusing too much on outperforming the latest benchmark score instead of practical implementation. This practical implementation is what distinguishes robot vision from computer vision [63]. As an active agent deployed in an environment, a robot has to make decisions and execute actions based on its visual input. In robot vision, perception is only one component of a larger and more complex problem, where safety and efficiency are important factors to take into account.

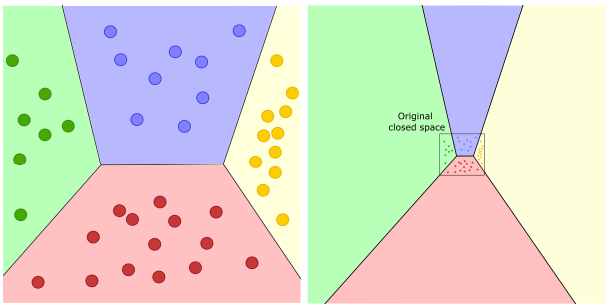


Figure 1.1: (a) *Closed set condition:* Feature representation of a classifier in closed set condition. The classifier has created decision boundaries based on the training samples. **(b)** *Open set condition:* Zooming out, it can be seen that the original closed space is only a fraction of the total feature space. However, the closed set classifier has labeled the entirety of this open space, introducing possible misclassifications in open space. [9]

In computer vision, the research has been focused mainly on *closed set* condition. All test classes are known at training time [60]. However, robot vision has to deal with an *open set* environment, that could potentially contain unknown objects. In this more realistic scenario, the agent has to assume it has incomplete knowledge at training time. This is visualized in Figure 1.1a. A closed set classifier assumes performance in closed set condition. When zooming out, it shows the original closed space is only a fraction of total feature space. Labeling data far form the training samples increases the probability of misclassifying objects unknown to the agent. In *open set recognition* the agent has the ability to reject a sample and label it as unknown. But acknowledging the unknown is not enough.

Preferably, the agent should be able to learn from new experiences. This is considered *open world recognition*. Besides detecting an unknown object, the agent should learn to recognize this novel object in the future [6]. Several studies have been performed to develop open world recognition models that are capable of successfully learning newly encountered object classes [9], [6], [5].

However, these methods, on their own, are not suitable to be performed autonomously. In [39] is pointed out that current open world models overlook the problem of acquiring data of newly encountered classes. *On-the-job* learning is introduced, which combines open world recognition with autonomous data collection. In [40] an on-the-job recognition approach is proposed that uses web scraping to gather training data of unknown objects to train an open world recognition algorithm. Unknown objects are reverse-image searched on the internet and the results are used as training data for the unknown object. They show their algorithm is able to successfully recognize new classes from images from the web.

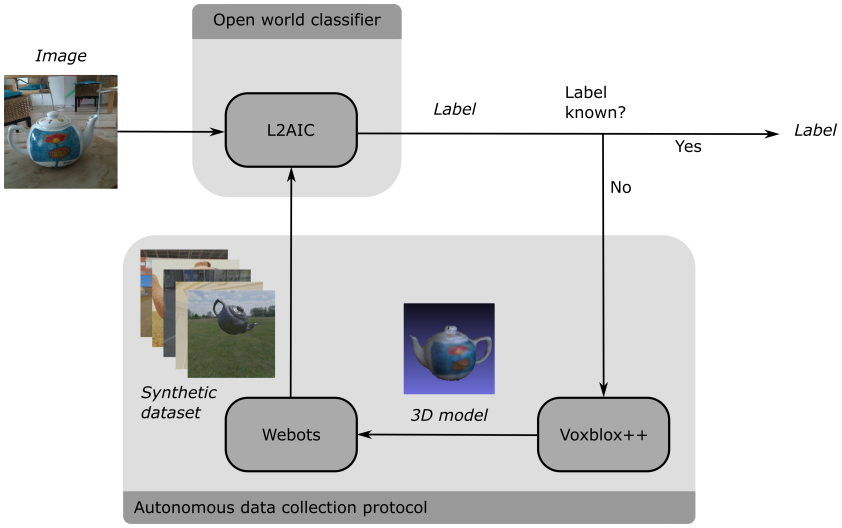


Figure 1.2: On-the-job object recognition. An object is presented to the open world recognition algorithm L2AIC. The L2AIC model gives the correct label to this object if it is of a known class. In the case the object is unknown, the autonomous data collection protocol is initiated. A 3D reconstruction is made using an RGB-D camera and Voxblox++. With the acquired 3D model a synthetic dataset is created. This dataset is added to the memory of the L2AIC model, increasing the knowledge of the model.

In this report an on-the-job object recognition approach is introduced that combines open world recognition with an autonomous data collection protocol to gather new data. An overview of the approach is given in Figure 1.2. Open world recognition is done with L2AIC, a deep meta-learning classifier. The autonomous data collection is done in two steps. First, an unknown object, identified by L2AIC, is scanned with an RGB-D camera and reconstructed in 3D with Voxblox++ [23]. In the second step, a synthetic dataset is created from this 3D model with the robotic simulation program Webots [44]. The synthetic dataset is used by L2AIC to learn to recognize the unknown object. Previous work on synthetic datasets showed that using 3D CAD models could achieve similar performance as when using real datasets [26]. This report is the first study to combine the use of 3D scanned models and synthetic data.

The key contributions of this report are the following:

- An on-the-job learning approach for open world recognition
- A fully automated protocol to synthetic datasets, the first using 3D scanned models.
- L2AIC, an open world recognition model derived from an open world text classifier.

In the remainder of this report the protocol is further elaborated. In Chapter 2 an overview of the theory is given. In Chapter 3 the open world recognition algorithm L2AIC is introduced. In Chapter 4 the autonomous data collection protocol is explained. Chapter 5 gives an overview of the experiments done in the report. In Chapter 6 the results are listed and these are discussed in Chapter 7.

2

Background

As introduced in the previous chapter, the on-the-job recognition protocol consists of an open world image classifier and an autonomous data collection protocol. The open world image classifier L2AIC, elaborated in Chapter 3 is adapted from the text classifier L2AC [67]. The autonomous data collection protocol is done by creating synthetic images from 3D reconstructed objects. In this chapter an overview is given of the existing theory and related work. First, in Section 2.1, open world recognition is explained. Then, in Section 2.3, the open world meta-classifier L2AC is introduced. Section 2.4 and Section 2.5 will explain more about 3D reconstruction and synthetic data, essential for the autonomous data collection protocol of Chapter 4. The chapter concludes with a brief summary in Section 2.6.

2.1. Open world recognition

In this section the concept of open world is explained in depth. As stated in Chapter 1 the focus of object recognition studies has been mainly on closed set conditions. The past decade the performance of object recognition models have improved significantly, but they assume a closed set condition. Currently the best closed set models are able to achieve a top-1 error of 18.68 % on the ILSVRC ImageNet challenge [27]. A remarkable feat, given this dataset contains millions of images from 1000 classes. However, during all these advancements the research has been focused on improving performance on these bench-mark datasets. The model can recognize all thousand classes of ImageNet very accurately. But as soon as it is shown a new class it will fail 100% of the time as it is designed for closed set conditions. Due to the focus on beating the bench-mark record the original goal of of object recognition has been lost [66], recognition in the real world.

The open set problem

Object recognition in the real world introduces the *open set problem* [60]. The open set problem indicates that during training time a model has incomplete knowledge of the world. This knowledge consists of a finite number of known classes, in a world that contains an infinite number of unknown classes [15]. This is in contrast with the closed set condition, where all classes encountered during deployment have been seen during training. A recognition model suitable for open set recognition has the ability to *reject* classes and classify them as unknown.

In [59] and later extended by [20], a speech of Donald Rumsfeld [57] is used to discern the different types of known and unknown objects that an agent can encounter during deployment in an open world setting. They are classified in four categories.

1. **Known known classes**: The *known known classes* are the set of classes that the agent has learned during training. When the agent is presented with an object from this set of classes, it should be able to classify it correctly. This is traditional closed-set multi-class recognition.
2. **Known unknown classes**: The *known unknown classes* are part of the training data of a binary classifier. These classes represent the set of negative samples the agent is shown during training. It cannot discern between the different objects that are part of this set of classes, but is has learned to classify them as negative.
3. **Unknown known classes**: The *unknown known classes* are the focus of zero-shot learning. The agent is learns features of an specific object class in some domain and is able to translate this to the visual domain, where the objects are encountered. The agent is aware of the existence of these classes and able to classify them correctly, although it has never seen these before in the visual domain.
4. **Unknown unknown classes**: The remaining classes are part of the *unknown unknown classes*. These are the set of classes that have not been seen during training and the agent is unaware of their existence until the moment such a class will be encountered during deployment. In the open set condition these classes have to be dealt with correctly.

Based on this categorization a more concrete definition of open set recognition can be formulated. In open set recognition the agent has learned to classify the known known classes ($\mathcal{C}^{\mathcal{K}}$). Objects of these classes it should recognize. Every object not part of this set of classes belongs to the set of unknown unknown classes ($\mathcal{C}^{\mathcal{U}}$). The agent is unaware of the existence of an object of this class, but should be able to classify it as unknown.

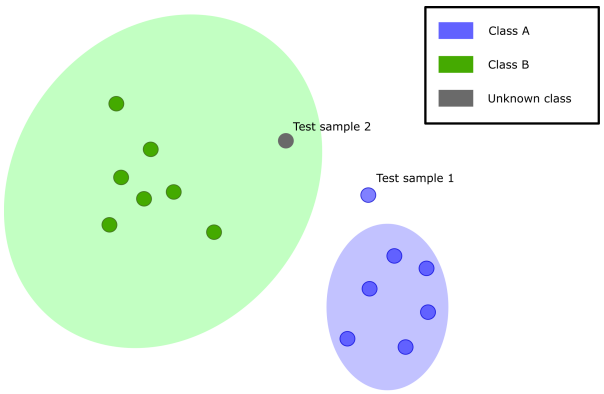


Figure 2.1: A visualization of open space risk in feature space. Classifier A is tightly bound around its training samples. This reduces open space risk. This is, however, at the cost of a larger empirical risk. Test sample 1, also part of class A, falls outside the classification boundary of class A. On the other hand, classifier B covers more feature space than its training samples. This reduces the empirical risk, but increases the open space risk. Unknown samples, such as test sample 2, will be classified as class B[9].

Open set risk

The main difficulty in open world classification is distinguishing the known classes from the unknown classes. In [60] the *open space risk* is introduced. The open space risk ($R_{\mathcal{O}}$) defines the risk of misidentifying an unknown class from $\mathcal{C}^{\mathcal{U}}$ as a class from $\mathcal{C}^{\mathcal{K}}$. At the other end of the spectrum lies the empirical risk ($R_{\mathcal{E}}$) [60], the risk that a class from $\mathcal{C}^{\mathcal{K}}$ is misclassified. An illustration is given in Figure 2.1. This figure shows a feature representation trained to cluster samples from the same class together. These classes have been learned during training from the available data. The cluster of class B extends far from the training samples in the feature space. This increases open space risk, the risk of classifying an unknown class as class B. Moving further away from the training samples increases the chance of encountering a sample from another class. On the other hand, the open space risk for class A is much

lower. The border of the cluster is tightly bound around the training samples. This, however, does increase the empirical risk, the risk that a new sample from a class is located outside the bounds of the cluster and thus will be misclassified. The goal of open set recognition is to optimize the open set risk (Equation (2.1)), which combines open space risk and empirical risk [60], with regularization constant λ_r and arbitrary classification function f.

$$\underset{f \in \mathcal{H}}{\operatorname{argmin}} \{R_{\mathcal{O}}(f) + \lambda_r R_{\mathcal{E}}(f)\} \tag{2.1}$$

Open world recognition
Open world recognition is an extension of open set recognition. Besides the ability to acknowledge the unknown, an open world algorithm should be able to extend its own knowledge [6]. When presented with enough information or samples from a specific object class from $\mathcal{C}^{\mathcal{U}}$, the agent should add this single class incrementally to its known classes, increasing the number of classes in $\mathcal{C}^{\mathcal{K}}$. Extending knowledge is known as *lifelong learning* and is essential for open world recognition. It allows an agent to learn from experience gathered during deployment.

On-the-job learning
The ability to learn from experiences during deployment is what sets apart open world recognition from open set recognition. However, it overlooks an important problem, pointed out by [39]. Many existing open world recognition studies assume that for each new task labeled data is available to learn from. However, being able to learn during deployment is useless if no data is gathered of past experiences. Ideally this data is gathered without losing autonomy of the agent. In [39] *on-the-job* learning is introduced, which tries to overcome this data limitation of open world recognition. Besides rejecting unknown classes and incrementally learn new classes, an agent capable of on-the-job recognition should also gather data autonomously during deployment.

Lifelong learning
The concept of lifelong learning in robotics is to be able to keep learning outside the training phase, during deployment. In [10] two important requirements of lifelong learning are stated. First, an agent should improve its performance on existing tasks during deployment. Secondly, the agent should be able to detect new tasks during deployment. Translating this to the field of object recognition, such a task is to classify an object class. The existing tasks are the classifications of the classes from the set of $\mathcal{C}^{\mathcal{K}}$. During deployment the agent should improve its classification on classes of $\mathcal{C}^{\mathcal{K}}$ based on new encountered samples from this set of classes. Besides classifying known classes, it should also learn to recognize new tasks, or new object classes. When these new classes are learned, they will be moved from the set of $\mathcal{C}^{\mathcal{U}}$ to $\mathcal{C}^{\mathcal{K}}$. These known tasks are stored in the *knowledge base* (KB). The knowledge base enables the agent to remember past tasks [10]. Examples of the knowledge base in object recognition are the parameters of a neural network or a memory containing image samples. When the existing tasks of in the KB gets updated, this is called *incremental learning*. The addition of an extra task to the KB has multiple names in the literature, *scalable learning* [6], [14], *cumulative learning* [18], [10] or *class-incremental learning* [63], [54]. In the remainder of the report class-incremental learning will be used.

Catastrophic forgetting
An unavoidable consequence of lifelong learning is *catastrophic forgetting* [42]. Learning new tasks will eventually degrade the performance on previously learned tasks and is also known as the plasticity-stability dilemma [1]. Plasticity of a model denotes the tendency to update the weights of the model during the training process. A very plastic model learns quickly, but is also quick to forget past tasks when new ones are learned. At the opposite end is stability. Models with high stability are less likely to update their weights. This results in a slower learning process, but also preventing fast performance loss of past tasks.

2.2. Deep learning

The most popular method for object recognition is deep learning [22]. The past decade, scientific advancements in the field of deep learning have significantly improved object recognition tasks. Deep learning models are neural networks that are constructed of multiple layers, hence the name deep learning, in contrast with shallow neural networks that only contain one or two layers [22]. The study of deep learning has been around for some time. Halfway the twentieth century the first concept of what we know as neural networks was introduced by Rosenblatt, the *perceptron* [55], to study and model the brain [43]. Soon it became clear that perceptron-based models were not only useful to mimic the human brain. These models could also be used to solve different problems, such as object recognition, that were deemed to be too difficult for computers at that time. However it was not until around 2010 that deep learning really started to take off. Before this time, deep learning approaches were computationally too exhaustive, training time would be too long and there were no suitable datasets available [2]. This changed because of the improving hardware such as GPUs and TPUs and network architectures that were much more efficient for specific application instead of only fully connected networks.

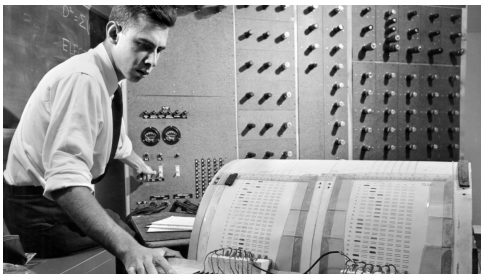


Figure 2.2: Rosenblatt working on the perceptron, the first concept of a neural network, introduced in 1958 [37].

One of the reasons for the success of deep learning is the strategy that is used to solve problems. In more conventional model-based approaches a model is created to simulate the world, a set of rules is conducted. These rules are used to generate answers from input. However, in deep learning, the inputs and answers are presented to the deep neural network and it will try to find the rules that correspond to the relation between input and answers [12]. The goal is to find rules that are generalizable enough such that the model is also applicable to unseen input of the same format.

Creating this set of rules, or model, is done by training. The training procedure aims to find the set of parameters that gives the best performance on a given task. In object recognition this performance is often indicated by the accuracy of a recognition model to correctly classify images. Finding these parameters is done based on an iterative optimization-based approach. An objective function or loss function is formulated and with the help of the backpropagation algorithm [56] the gradient of each parameter with respect to the loss function is calculated. With a gradient-based optimization algorithm the set of parameter that give the best performance is found. Optimization in neural networks slightly differs from classical optimization [22]. First of all, the performance metric of object recognition, for example accuracy, is only optimized indirectly. Accuracy it is not continuously differentiable, a requirement for backpropagation [50]. A surrogate loss function has to be used [50], which is continuously differentiable and changes even with small changes in the parameters. Popular functions in object recognition are cross-entropy and binary cross-entropy [12]. Secondly, it is not the performance on the training data that is optimized, but the performance of the model on test data. Furthermore, because of the large number of trainable parameters, finding the true global minimum is a difficult non-convex optimization problem as there are numerous local minima. Fortunately, most neural networks do not require to reach this global minimum as long as the loss value of the local minimum has decreased enough, which is often the case [22].

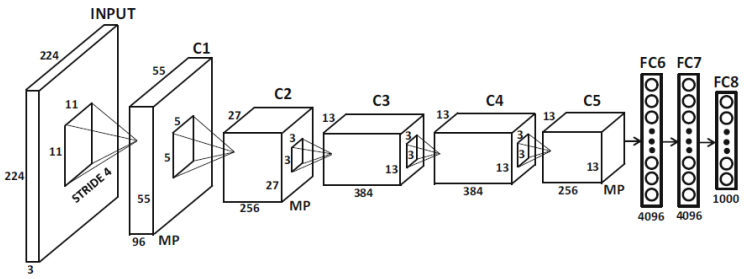


Figure 2.3: The model architecture of *AlexNet* [34]. AlexNet requires an input image size of 224×224, which is passed through five convolutional layers. Then its passed through two fully connected layers and finally to a soft-max classification with 1000 classes. The penultimate layer, FC7, is used for feature extraction[2].

Deep learning models in object recognition

In this report several deep learning models are used for feature extraction. The first model, *AlexNet* (AN) [34], was introduced in 2012 and is named after the author Alex Krizhevsky. This model was designed for the 2012 ILSVRC ImageNet challenge [58], and was the winner of that year [2]. It is the first deep network that used multiple CNNs. The AlexNet architecture consists of five convolutional layers and three fully connected layers, as displayed in Figure 2.3. The penultimate fully connected layer (FC7) is used for feature extraction. Although feature extraction was a known technique at that time, the introduction of AlexNet made it more mainstream, which is why the penultimate layer of a CNN, the layer used for feature extraction, are often called *FC7 layers* [2]. Furthermore, a lot of design choices in AlexNet became standard after publication of the study [2] and is what led to choice of AlexNet as an encoder.

The second and third recognition models are *ResNet-50* (RN50) and *ResNet-152* (RN152) [24], developed for the ImageNet challenge of 2015, and are the first deep residual networks. Where AlexNet consisted of 8 layers, ResNet-50 and ResNet-152 consist of 50 and 152 layers respectively. The design of residual networks proved to be very efficient for creating deeper neural networks. Training time of networks increases with depth, resulting in very slow convergence for deep networks [24]. This was reduced in [24] by introducing *shortcut connections*. A shortcut connection connects a layer *i* with another layer, that is not directly its consecutive layer. This idea behind this is that not all features require the same depth for correct representation. The shortcut connections allows the model to learn the required depth for each feature and improves convergence [24]. In this study ResNet-50 and ResNet-152 are chosen because of their depth. While ResNet-152 has even more layers than ResNet-50 this is at the expense of model size and efficiency.

The final model used as encoder is *EfficientNet* (EN) [64]. Where the residual networks were designed to increase depth of the network, EfficientNet is developed to increase depth, width and image resolution. The authors of [64] show there exists a relationship between image resolution, width and depth and finding the right balance results in better performance of the network. This relation is used to create a new set of models, EfficientNets. The first baseline model, *EfficientNet-b0*, is used in this project, because it is more efficient than the ResNet architectures, contains less parameters, while achieving similar performance.

Meta-learning

Despite the success of deep learning, traditional deep learning models are not perfectly suited for open world recognition tasks [53]. Training deep networks takes a lot of time and a lot of samples. This is not always available in an open world setting, where new tasks should be learned during deployment. Another disadvantage is that a deep learning model requires retraining for each incrementally learned class. *Meta-learning* tries to overcome these problems. Meta-learning, also considered *learning to learn*, creates a machine learning model capable of performing similar but different tasks without retraining. An example of meta-learning in object recognition is *few-shot* learning [53]. Few-shot models are capable of accurate object recognition with training on only a few examples. These models are often referred to as n-way k-shot models. A classification is made from n classes based on k samples from each class.

2.3. Learning to accept classes (L2AC)

In [67] a novel method for open world text classification is introduced, Learning To Accept Classes (L2AC). This method is a meta-learning approach based on deep learning and works with a memory containing samples of known classes. The memory is used to classify known classes and reject unknown classes. If enough samples exist, such an unknown class can be added to the memory without the need to retrain the model. This allows for efficient use during deployment as no down-time is needed for class-incremental learning.

Model framework

In Figure 2.4 the framework of the L2AC algorithm is shown [67]. The algorithm works according to a top-*k* top-*n* principle, similar to few-shot learning. The model predicts a probability of similarity between an input sample and a class, based on k images from this class. This is done for n classes, resulting in n probabilities. The input sample will be assigned to the class with the highest probability. If all n outputs of the meta classifier are below the classification boundary, h, the input sample is classified as unknown.

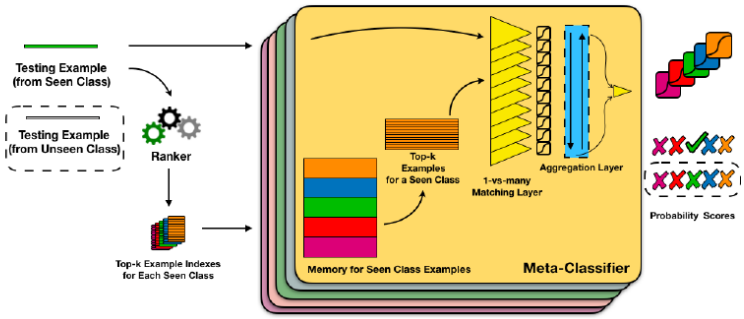


Figure 2.4: The L2AC framework. Assume the seen class set $\mathcal{C}^{\mathcal{K}}$ has 5 classes and their samples are indicated by 5 different colors. L2AC has two components: a ranker and a meta-classifier. Given a (green) testing example from a seen class, the ranker first retrieves the top-k most similar samples from each seen class. Then the meta-classifier takes both the test sample and the top-k most similar samples for a seen class to produce a probability score for that class. The meta-classifier is applied 5 times (indicated by 5 rounded rectangles) over these 5 seen classes and yields 5 probability scores, where the 3rd (green) class attends the maximum score as the final class (green) prediction. However, if the test example (grey) is from an unseen class (as indicated by the dashed box), none of those probability scores from the seen classes will predict positive, leading to rejection.[67]

First the input sample is passed through an feature extractor, or encoder. The encoder uses the NLTK as a tokenizer, a GloVe embedding and a BiLSTM to translate input text to feature vectors. Then, a ranker will compare the input feature vector x_i with the feature vectors of known classes from the memory. This is done with cosine similarity. First the top n similar classes are found by calculating the similarity between the input vector and the mean feature representation of each class. From each of the top n classes the top k feature vectors most similar to the input sample are passed through the meta-classifier.

The meta-classifier consists of the matching and the aggregation layer. First the matching layer will calculate a similarity score between the input and each of the top k feature vectors $x_{a_{1:k}}$. This is done using the similarity function f_{sim} (Equation (2.2)), where $\oplus$ denotes the concatenation operation. The similarity function is passed on through a neural net that gives k similarity scores $r_{1:k}$.

$$f_{sim} = |x_i - x_{a_i}| \oplus |x_i + x_{a_i}| \tag{2.2}$$

The similarity scores are passed on to the aggregation layer, that aggregates all scores and gives a final probability score that represents how likely the input sample belongs to the class. This aggregation layer functions as a one-vs-all binary classification [60]. A final probability is predicted for the n classes. The final classification of the sample is done by selecting the highest probability score of one of the n classes. If all classes have a score lower than h, the input samples is classified unknown. The aggregation layer contains an LSTM. This is a common meta-learning method to automate the optimization process [3]. In Figure 2.5 the full neural network design of L2AC is shown.

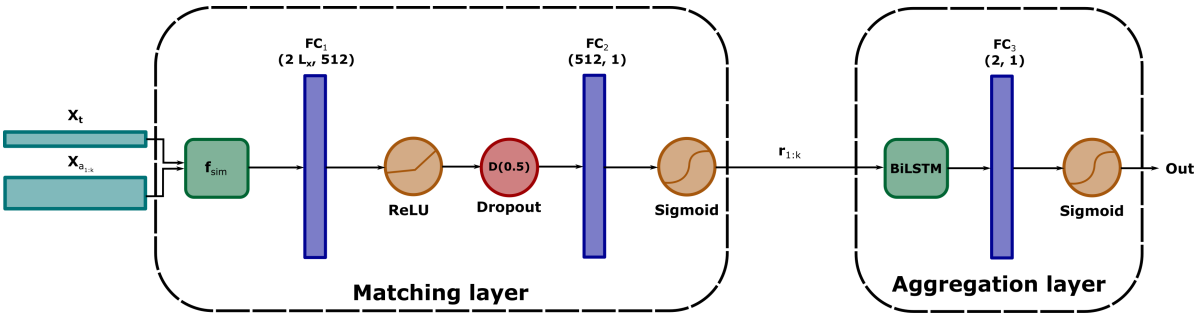


Figure 2.5: The architecture of the neural network of L2AC. The input sample x_t and the samples from the memory x_{a_i} are passed through the matching layer, with L_x the lenght of the feature vectors. Using the similarity function f_{sim}, the matching layer outputs k similarity scores $r_{1:k}$. These are aggregated by the aggregation layer to predict the final output.[67]

The combination of the meta-classifier and the memory makes l2AC an effective open world classifier. Similar to few-shot learning the model can classify input based on just a few samples and generalize well enough to correctly classify new classes as well. Furthermore, the memory of L2AC is interchangeable, meaning that new classes can be added anytime, without the need to retrain the whole model.

2.4. 3D reconstruction

What distinguishes on-the-job recognition from open world recognition is the ability to add new classes to the knowledge base during deployment. In order to perform well on the classification of these new classes. This often requires a lot of data from this specific class. In this section is described how an agent can acquire this data by reconstructing a 3D model of the novel object. Research on 3D reconstruction has been gaining attention since the introduction of Iterative Closest Point (ICP), in 1992 by [7]. ICP is an optimization method to reconstruct a 3D model from multiple partial observations. However, it required a lot of computational power for that time making it a slow process. Furthermore, the RGB-D data provided by the RGB-D cameras was not of high quality. It was not until the introduction of the Microsoft Kinect [45] that the research on 3D reconstruction made big steps. Together with improved computational power, this lead to real-time high-quality reconstruction methods, starting with KinectFusion [46] in 2011. In Section 2.4.1 different techniques are presented for an agent to autonomously capture 3D data. In Section 2.4.2 Voxblox++ is introduced, a method to reconstruct objects from 3D data.

2.4.1. Acquisition systems

As stated in Section 2.4 open world recognition requires the ability to add new classes to the knowledge base. The gathering of this data should be done by the agent during deployment, and preferably requires no human supervision. Since the introduction of Microsoft Kinect [45], RGB-D cameras have become more compact, light-weight and affordable. These cameras have become a standard sensor in the field of robotics. The latest iteration of RGB-D cameras often use time-of-flight (ToF) to accurately measure depth [16]. Time-of-Flight is based on the time it takes for a light pulse to reach an object, reflects and gets captured by a sensor. By using the property of light, traveling at a constant speed, the distance between the object and the camera can be calculated. This is illustrated in Figure 2.6. In combination with a regular RGB camera this technique can acquire high quality RGB-D images.

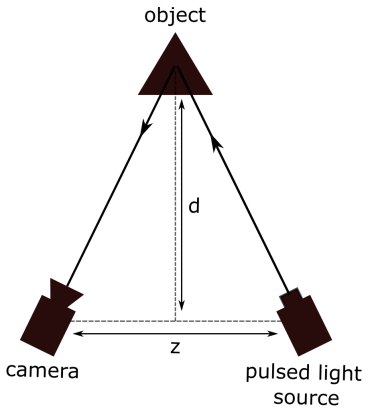


Figure 2.6: The time-of-flight principle. A light source, often a laser, emits a pulsed light. This light pulse travels a certain distance before it hits an object and reflects back. This reflection is captured by a camera. The distance traveled by the light pulse can be inferred by the time it takes between the emitting of light and the moment it is captured by the camera [16].

Equipped with an RGB-D camera the agent should be able to make a full reconstruction of the object. This is often done using a robotic manipulator with the camera attached to the end-point, known as *eye-in-hand* [32] and is shown in Figure 2.7. The eye-in-hand technique mimics a human using a camera in hand. Attaching this robotic manipulator on a mobile base will give a large range of motion, enabling the agent to capture the object from multiple angles. This does, however, increase the complexity of motion planning for the agent. This complex problem is outside of the scope of this report.

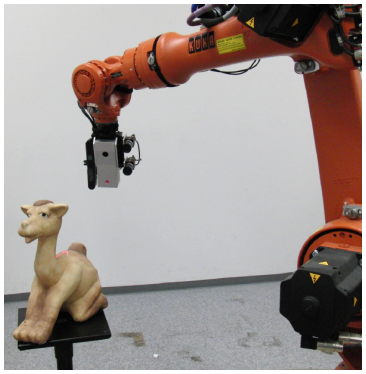


Figure 2.7: A robotic arm with a RGB-D camera attached to the end effector [32]. The robotic arm moves around the object presented on the plateau.

2.4.2. Voxblox++

When the agent has captured enough RGB-D data from all angles of the object, this data needs to be combined to retrieve a complete 3D model. This is done with 3D reconstruction methods such as Voxblox++[23] [19] [47] [48]. This method is able to automatically segment objects from ground surfaces for each individual frame and create separate meshes for each object in an online fashion. The pipeline is visualized in Figure 2.8. Voxblox++ requires a pose of the RGB-D camera for each RGB-D frame. An advantage of Voxblox++ is that it does not require predefined selection of objects that can be reconstructed [19], taking into consideration the open-set condition of the real world.

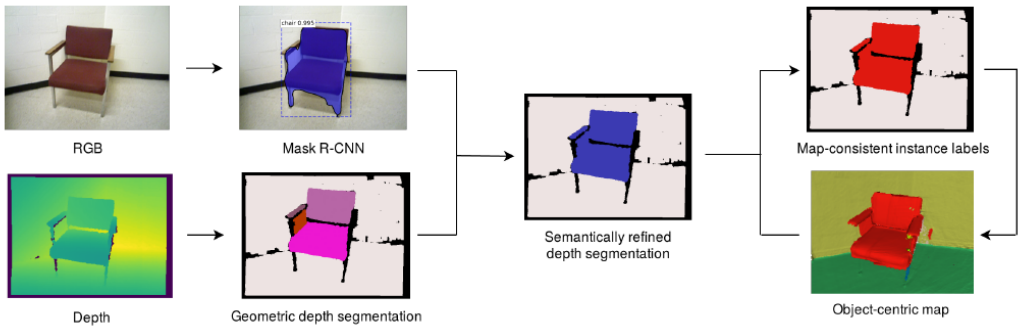


Figure 2.8: A visualization of the pipeline of Voxblox++. The depth and color frame are both individually segmented using geometric and semantic segmentation respectively. Both results are combined for a more accurate segmentation of the frame. The segmented frame is integrated in the Global Segmentation Map[23].

Segmentation

Each RGB-D frame t that Voxblox++ retrieves from the camera gets segmented. First, geometric depth segmentation is performed on the depth image [19]. From the depth image the normal vectors of the surface are calculated and used to determine the local convexity of each pixel. This is combined with a depth discontinuity filter that detects large edges. Together, this combination segments the closed forms and contours from the depth image. The geometric depth segmentation is object independent, allowing for segmentation of unseen objects. The geometric segmentation will form a set of 2D geometric segments, $\mathcal{R}_t$, located in the pixel space, specific to that frame. Secondly, a semantic instance-aware segmentation is performed [23] on the RGB image. This is done using the object detection model Mask R-CNN [25], also performed in pixel space. The semantic segments, or masks, are stored in the set of semantic masks $\mathcal{M}_t$ of the specific frame. The overlap ratio between geometric segment r_i and the semantic mask M_k is calculated according to Equation (2.3). The geometric segment is assigned the semantic label o_k of the mask with the highest overlap ratio p_i. If this highest overlap ratio does not exceed the rejection threshold τ_p, the semantic class will be rejected and no semantic label exists for this object.

$$p_{i,k} = \frac{|r_i \cap M_k|}{|r_i|} \tag{2.3}$$

$$\begin{aligned} p_i &= \max_k p_{i,k} \\ \hat{k}_i &= \arg\max_k p_{i,k} \end{aligned} \tag{2.4}$$

The 2D set of segments $\mathcal{R}$, can be mapped to the set of 3D segments $\mathcal{S}_t$ by utilizing the depth information of the RGB-D image. Each segment s_i is assigned a unique geometric label $l_j \in \mathcal{L}$ and, if available, a semantic object label $o_k \in \mathcal{O}$ These labeled segments are then placed in the Global Segmentation Map (GSM).

Truncated signed distance function

The GSM is based on Voxblox from [48], a real-time volumetric TSDF surface representation framework. The Truncated signed distance function (TSDF), introduced by [13], is a method to efficiently capture surfaces. This data can then be converted to a 3D mesh using ray-casting. The idea of TSDF is to split a volume up in three dimensional pixels, called voxels. Each voxel contains distance value D, the distance from the voxel to the closest surface and has voxel size v. Voxels inside a closed volume contain a negative value, while outside voxels have a positive value, as illustrated in Figure 2.9. A zero crossing between two voxels denotes a surface. Voxels that are a distance further than the truncation distance δ from a surface will be set to either 1 or -1 depending on their position with respect to the surface. This allows for much more efficient volumetric computation. The TSDF is an implicit method to store surface information [31], which is efficient and low in memory.

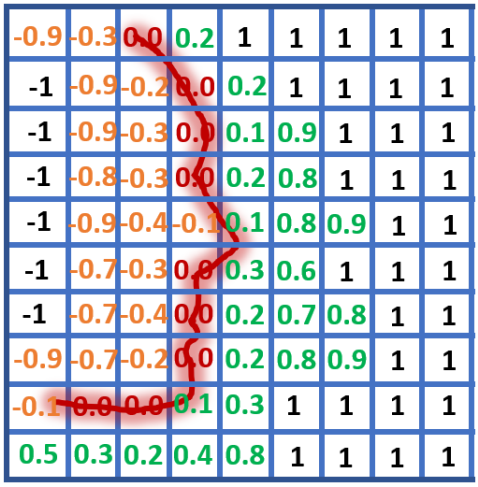


Figure 2.9: A visualization of the truncated signed distance function TSDF. A surface is represented by a voxel grid. The value of each voxel indicates the distance to the nearest surface. For computational efficiency distance value are truncated after the truncation distance δ. This figure uses $\delta = 1$. [11]

$$
\begin{aligned}
d(\mathrm{x}, \mathbf{p}, \mathrm{s}) &= \|\mathbf{p} - \mathrm{x}\| \operatorname{sign}((\mathbf{p} - \mathrm{x}) \bullet (\mathrm{p} - \mathrm{s})) \\
D_{i+1}(\mathrm{x}, \mathbf{p}, \mathrm{s}) &= \frac{W_i(\mathrm{x}) D_i(\mathrm{x}) + w(\mathrm{x}, \mathbf{p}) d(\mathrm{x}, \mathbf{p}, \mathrm{s})}{W_i(\mathrm{x}) + w(\mathrm{x}, \mathbf{p})} \\
W_{i+1}(\mathrm{x}, \mathbf{p}) &= \min\left(W_i(\mathrm{x}) + w(\mathrm{x}, \mathbf{p}), W_{\max}\right) \\
w(\mathbf{x}, \mathbf{p}) &= \begin{cases} \frac{1}{z^2} & -\epsilon < d \\ \frac{1}{z^2} \frac{1}{\delta - \epsilon}(d + \delta) & -\delta < d < -\epsilon \\ 0 & d < -\delta \end{cases}
\end{aligned}
\tag{2.5}
$$

The integration of each new segment in the TSDF is done using raycasting [13]. A ray is cast from the sensor origin $\mathbf{s}$ to the position of a point $\mathbf{p}$ from the new captured depth frame. Each voxel along this ray gets updated according to the set of Equations (2.5) [48]. Consider a voxel with position $\mathbf{x}$, distance value D and weight value W. This voxel gets updated with a point with position $\mathbf{p}$. First the distance d of this voxel with respect to the nearest surface along this ray is calculated. The distance of this particular measurement is given a weight w according to Equation (2.5). The weight is determined by the truncation distance δ, intermediate truncation distance ϵ and the depth measurement z from the camera frame to the point. If the distance between the voxel and surface is larger than the truncation distance, the weight of the voxel will be set to zero. The point will not be used to update the voxel. Specific to Voxblox++ [19] the TSDF is extended with a label map. Besides the distance and weight, voxels also contain a geometric and semantic label. As stated in Section 2.4.2 the incoming frame is segmented and each point is assigned a geometric label $l_i \in \mathcal{L}$ and if present, a semantic label $o_k \in \mathcal{O}$. The labels of the points that are present inside a voxel are inherited by the voxel. Instead of saving a single label per voxel, a label count is saved. This count is updated by each new point located in the voxel. The final assigned label of the voxel is the label with the largest count. This ensures more robust updating between frames [19].

2.5. Synthetic datasets

Performance of image recognition models is heavily correlated to the number of training samples available [62]. Many open world algorithms, such as [30] or [6], assume there is enough data available of new classes to apply class-incremental learning. The problem of gathering data during deployment is often not tackled in studies of open world recognition [39], but is crucial for an autonomous agent capable of on-the-job recognition. However, during deployment, the amount of data of a new object class is often limited to a single object instance in a single environment. Creating a dataset from this single instance is an interesting challenge due to domain shift [29].

Figure 2.10: Style augmentation [29]: Existing images are altered by changing the general style of the image.

Data augmentation

A simple, yet effective way to increase available training samples from scarce images is data augmentation [22]. By randomly cropping, rotating or flipping images before passing them to a classification model during training increases the number of unique samples and can be greatly improve robustness of image recognition models. However, these simple techniques do only increase the number of training samples and do not account for domain shift [29]. A dataset of single object instance will still be heavily biased towards this object and the environment the object is encountered. More advanced augmentation methods exist that alter the contents of the image, such as Random Shadows and Highlights (RHS) [41]. This method generates artificial shadows and highlights that can make recognition models more robust against light perturbations. Another method, Style Augmentation [29], only preserves the shape of the original object but randomizes texture, color and contrast (Figure 2.10). Data augmentation methods however, only alter an existing image, making it difficult in simulating different environments. In the study of [4] is shown that background, object rotation and viewpoints are heavily influencing the bias of datasets. They introduce ObjectNet, a dataset that contains objects in non-standard surroundings and orientations. They show that state-of-art recognition models trained on standard datasets, such as ImageNet [58], had a severe performance drop when applied to their ObjectNet dataset (see Figure 2.11). This leads to the conclusion that to achieve robust models, background surroundings and viewpoints should be strongly varied in order to reduce bias

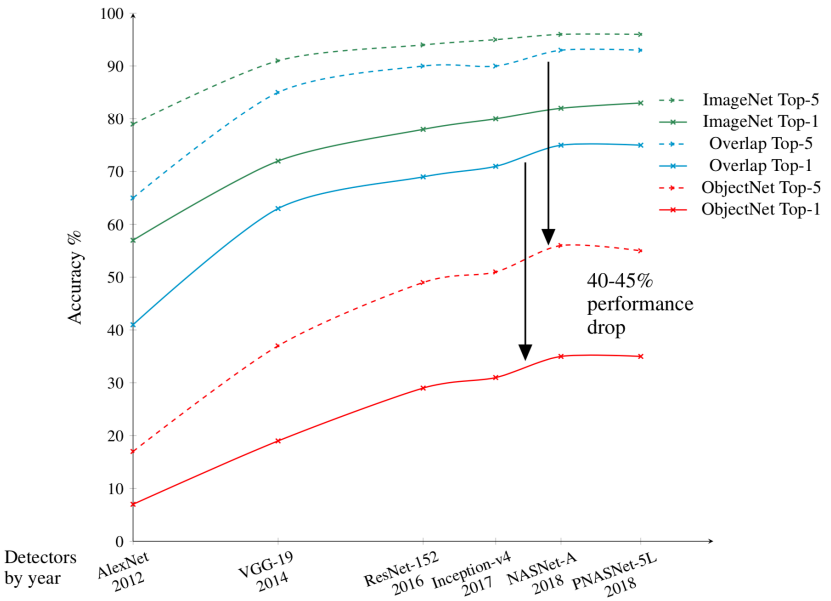


Figure 2.11: Performance of state-of-the-art object detectors trained on ImageNet. Their performance on all objects of ImageNet is shown in green. Performance on ImageNet, only selecting classes that overlap with ObjectNet, are shown in blue and performance on ObjectNet is shown in red. A drop of 40-45% can be seen between samples of ImageNet and ObjectNet. Solid lines denote the top-1 performance (correct in 1 guess) and dashed lines the top-5 performance (correct in 5 guesses).[4]

Domain randomization

ObjectNet uses the idea of domain randomization to cover a large part of all possible object configurations and backgrounds. However, acquiring these images is a very costly and time-consuming process. Synthetic data is a way to create a large amount of data efficiently. Furthermore, it is not restricted by requirement of the physical objects being present in non-standard environments. The usage of synthetic data unfortunately introduces another problem, the synthetic to real domain gap [51]. Training a model on synthetic data does not guarantee the model will perform as well on real data. This is found in the studies of [17] and [21]. Synthetic data is created by pasting 2D object images on a background (Figure 2.12). The results show that using only synthetic data will reduce performance due to the domain gap. A combination of synthetic and real data did increase lead to an increase of performance.

Figure 2.12: Four synthetic image created by the method of [17]. Objects are randomly placed on backgrounds.

However, in [26] is shown that using proper domain randomization techniques can create datasets that achieve similar performance compared to when using real data. In this study the images are created using 3D CAD models. The use of the 3D models allows for images of objects from different views, without the time consuming process that is inherent to placing real objects. Furthermore, environments and light conditions are easily interchangeable. Furthermore, the paper of [26] delivers some interesting conclusions. First, as already mentioned, their ordered way of creating a randomized dataset is significantly more effective than a pure random strategy. During training they gradually reduce the scale of the objects. The training starts with larger versions of the objects. As the training continues, it gets more difficult as smaller objects have to be detected. Secondly, Pure synthetic backgrounds deliver a better performance than a mix of synthetic and real backgrounds. Finally, the use of backgrounds with a lot of clutter improves the performance.

2.6. Summary

In this chapter a brief overview is given of the background necessary for the remainder of the report. First, in Section 2.1, open world recognition is introduced. It is discovered that for true autonomy open world recognition is not sufficient and on-the-job learning is required. Then, in Section 2.2, four deep learning models are discussed that will be used in Chapter 3. Section 2.3 gives a summary of L2AC, which forms the base for the open world recognition model introduced in Chapter 3. Section 2.4 introduced Voxblox++, the 3D reconstruction method that is used in Chapter 4. Finally, in Section 2.5 was explored how to overcome the domain gap between synthetic data and real data.

3

Learning to Accept Image Classes (L2AIC)

In this chapter Learning to Accept Image Classes (L2AIC) is introduced. This open world recognition algorithm is derived from L2AC [67] (Section 2.3), an open world classifier based on text. Similar to L2AC, L2AIC consists of five building blocks. The encoder, the ranker, the memory, the matching layer and the aggregation layer (Figure 3.1). When an input sample is presented to the model, it will be translated to a feature vector by the encoder. The ranker will find the k most similar feature vectors for the n most similar classes from the memory. This is passed through the meta-classifier, that will determine if an input sample belongs to a class or not. The L2AIC has been changed compared to L2AC in order to make it compatible with image classification. First, in Section 3.1, the change in encoder is described. In Section 3.2 is explained how the ranker will rank the memory to input samples. Finally, in Section 3.3 the meta-classifier, consisting of the matching layer and the aggregation layer, is elaborated. Here several changes have been made to enhance performance on image classification.

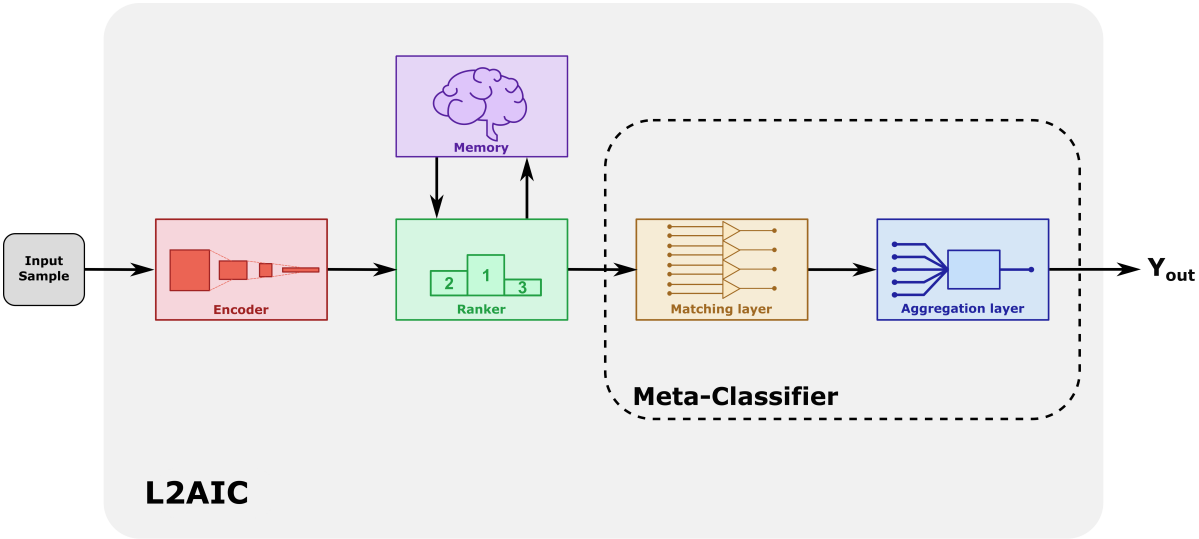


Figure 3.1: The pipeline of the open world classifier L2AIC. Features are extracted from an image sample with the encoder. Using the ranker k samples for n classes are selected from the memory for in depth comparison. The meta-classifier either classifies the image sample as one of the n classes or as a unknown class. [67]

3.1. Encoder

The encoder in the original L2AC algorithm (Section 2.3) consists of a Bidirectional LSTM with a GloVe embedding [67]. The goal of this deep learning encoder, designed for text classification, is to create a feature representation that captures relevant features from the text using representation learning. Similarly, L2AIC requires an encoder that is able to capture relevant features from an image and translate this to a feature vector or feature representation. The ideal feature extractor is able to extract all relevant features of an image. This would require a lot of training. A common approach for feature extraction from images is to use existing image recognition models pretrained on ImageNet [58]. The ImageNet dataset consists of 1000 classes with millions of images. The feature representations learned by the pretrained models are often very well suited to use in other image classification tasks [2]. The early layers of the model capture very well generalizable features that are inherent to any image. These features pass through later layers that construct features more specific to the object of the classification task. A deeper network generally leads to better performance and better feature representation.
In this project the feature representations of four object recognition models are selected, introduced in Section 2.2. The models have been pretrained on ImageNet and the final layers are fine-tuned on the dataset used in the project. In table 3.1 each encoder and its properties are listed. Feature vector length varies between encoders, resulting in different input layers of the meta-classifier. For each encoder the *FC7 layer*, the penultimate layer for feature extraction is displayed as well a the earliest layer that is trained in the fine-tuning process. After extraction all feature vectors are normalized, to improve generalization [22].

Table 3.1: Encoders and their properties.

Encoder	Feature size	Layer depth	Parameters	Fine-tune layers
AlexNet	4096	8	60M	Conv 5 and up
ResNet50	2048	50	26M	Residual block 14 and up
ResNet152	2048	152	60M	Residual block 14 and up
EfficientNet	1280	18	5.3M	Block 6 and up

3.2. Ranking memory

When L2AIC receives an input image it is passed through the encoder translated into a feature vector x_t. For each class the input vector is compared to k samples from this class. Finding these k feature vectors for n classes is done using the ranker and the memory (Figure 3.2). The memory $\mathcal{M}$ contains samples from a set of classes, $c \in \mathcal{C}^{\mathcal{M}}$, as well as the average feature vector from each class $\bar{x}^c$. The ranker works as a simple preselection process to reduce computations needed. Preselection is done using cosine similarity $f_{cos}(a, b)$ (Equation (3.1)). First the cosine similarity between the input and average class vectors is calculated $f_{cos}(x_t, \bar{x}^c), c \in \mathcal{C}^{\mathcal{M}}$. The n most similar classes are selected for further classification. For each of these n classes the cosine similarity between the input sample and each sample of the class is calculated. The k most similar samples $x_{a_{1:k}}$ are selected and passed on to the meta-classifier.

$$f_{cos}\left(a, b\right) = \frac{a \cdot b}{|a|_2 \, |b|_2} \tag{3.1}$$

3.3. Meta-classifier

The meta-classifier works according to an n-way k-shot principle. Each input sample is compared to n classes, with k images per class, selected by the ranker. The goal of this meta-classifier is to predict if the input sample belongs to one of these n classes. The output is a binary classification for each of the n classes. A positive classification denotes that an input sample belongs to a class, a negative classification means the input is not part of the class.

Matching layer

The k feature vectors $x_{a_{1:k}}$, together with the input feature vector x_t are passed through the matching layer. The matching layer, a neural network, consists of a similarity function and fully connected layers. The similarity function f_{sim} (Equation 2.2) uses the concatenation operation $\oplus$ to transform both vectors to similarity space. From this similarity space the matching layer generates a similarity score for each of the k feature vectors, $r_{1:k}$. The aggregation layer gathers these scores and assigns a final classification label to the class.

Similarity function

To give more insight in the similarity function of Equation (2.2), the output is visualized in Figure 3.2. The ResNet152 encoder has been used to extract features from an image. Only the first 100 entries of this feature vector are taken instead of the full length for visualization purposes. The goal of f_{sim} is to translate two vectors to a similarity space where an easier distinction can be made between similar vectors and dissimilar vectors. The similarity function consists of two parts, both with special properties. The first half of the equation, $|x_t - x_{a_i}|$, calculates the index-wise distance of the two vectors. In the case of two identical vectors, $x_t = x_{a_i}$, this results in zero, as can be seen in the figure. This, however, does not make a distinction between important features (high values) and less important features (low values). The second part, $x_t + x_{a_i}$, points out which features are most important to both vectors. If the output of a feature of index m is low at the first half of the concatenation and a high at the second half, this indicates the feature of vectors x_i and x_{a_i} are similar and for both vectors this feature is important. Figure 3.2 shows that two identical vectors give a clear distinguishable output. However seeing the difference between two vectors of the same class, $x_t, x_{a_i} \in C^j$, or two vectors from a different class, $x_t \in C^j; x_{a_i} \notin C^j$, is more difficult for the human eye. In the matching layer fully connected layers are used for accurate prediction.

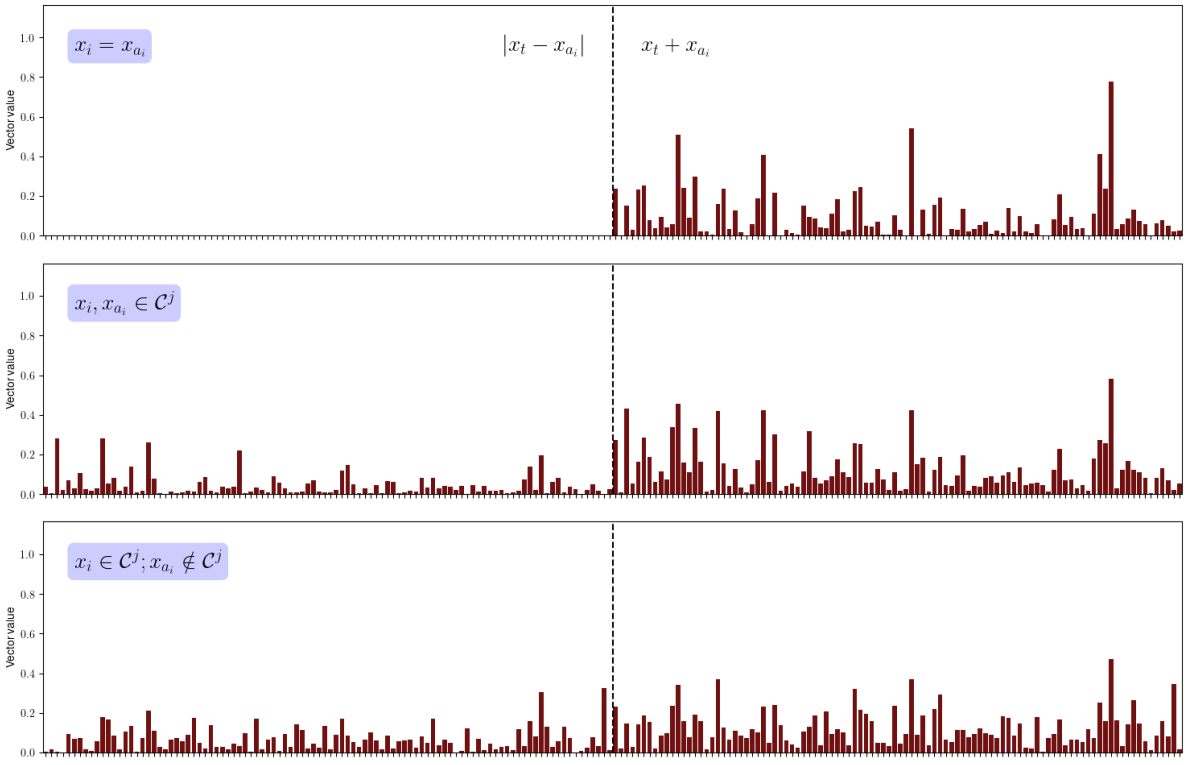


Figure 3.2: The output of the similarity function f_{sim} of Equation 2.2, on two feature vectors. The left side of the dashed line shows the subtraction of both vectors. The right side shows the sum of both vectors. After concatenation this is the output of the similarity function. The upper figure shows the output of two identical vectors, $x_t = x_{a_i}$. The middle figure shows the output of two vectors from the same object class, $x_t, x_{a_i} \in C^j$. The bottom figure shows the output for two vectors from different object classes, $x_t \in C^j; x_{a_i} \notin C^j$. ResNet152 is used as encoder and only the first 100 values of the feature vector are used instead of the full length for visualization purposes.

General changes

Although the meta-classifier is designed to learn to recognize similarities between feature vectors, a direct transfer to image-based feature vectors results in a significant drop in performance. A possible explanation is the difference in signal to noise ratio between images and text samples [12]. Commonly, a piece of text contains mostly words that are relevant to the meaning of the text. This is different for an image, consisting of many individual pixels and not every pixel is relevant for classification of the object. Background pixels, for example, can be seen as noise as they have, in most cases, no relation with the actual object. The meta-classifier of L2AIC has undergone some adjustments from L2AC for compatibility with images. First, an extra dropout layer ($p = 0.5$) is added to the input. This dropout increase the generalization ability of the meta-classifier by adding more randomness in the incoming feature vectors [61]. By using dropout as the first layer it acts as a way of data augmentation. Each iteration the input vectors passed on to the model in a slightly different form. The second adjustment is to replace the ReLU activation function with a Leaky ReLU activation function. This is to prevent the problem of *dying neurons*, sometimes occurring with ReLU functions [2]. The third addition is the use of weight initialization. This can improve convergence time and performance [22]. Initialization is done according to best practices of [50], with Xavier normalization for fully connected layers and orthogonal initialization for the LSTM. The fourth addition is changing the size of the second fully connected layer in the matching layer from 512 to the feature size of the input vectors, increasing the capacity of the model based on the input vector size. The final adjustment is extending the hidden size of LSTM from 1 to k, the number of samples compared per class. This is also done to increase the capacity of the model [22]. The full architectural design is shown in Figure 3.3.

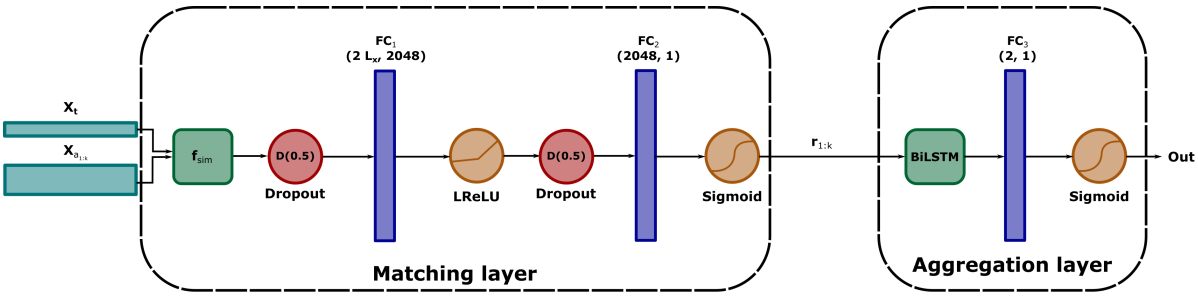


Figure 3.3: The architecture of the neural network of L2AIC. The input sample x_t and the samples from the memory x_{a_i} are passed through the matching layer, with L_x the length of the feature vector. Using the similarity function f_{sim}, the matching layer outputs k similarity scores $r_{1:k}$. These are aggregated by the aggregation layer to predict the final output. In Section 3.3 the design changes with respect to L2AC are explained.

Meta-classifier architecture

Besides the basic adjustments mentioned previously, several versions of the meta-classifier have been created and are listed in below. These variations have been created to study the influence of design choices in the matching and aggregation layer. All variations contain the same base-changes as *L2AIC-default*, the version described in previous in previous paragraph. In appendix A the exact architectures of each model can be found.

- **L2AIC-default**: This version is a copy of the L2AC algorithm, but with the adjustments mentioned in Section 3.3.
- **L2AIC-cosine**: The matching layer has been replaced with the cosine similarity function (Equation (3.1)). This effectively removes the whole matching layer and the aggregation layer will predict a final score based on output values of the ranker. This variation puts the influence of the matching layer to the test.
- **L2AIC-no-lstm**: The LSTM in the aggregation layer is replaced with multiple fully connected layers. Besides aggregation the LSTM also functions as optimizer for the matching layer (Section 2.3). This variation tests the influence of the LSTM on performance of the L2AIC model.
- **L2AIC-abssub**: The similarity function (Equation (2.2)) has been reduced to only the first part of the concatenation, the absolute subtraction of both vectors. As shown in Figure 3.2, this part shows if two vectors are close together in similarity space.

- **L2AIC-concat**: The similarity function is removed with the idea that the meta-classifier learns a similarity function on its own. Both feature vectors are concatenated and passed through the fully connected layers.
- **L2AIC-extended-similarity**: A copy of L2AIC-concat, where the similarity function is removed, but now the matching layer has been extended with multiple fully connected layers. This give the matching layer more capacity for finding a similarity function.
- **L2AIC-smaller-fc**: Both feature vectors are reduced in size by a fully connected layer before being passed on to the similarity function. This fully connected layer might help in the selection of most relevant features before being translated to similarity space.

3.4. Summary

In this chapter L2AIC is introduced. This open world recognition model is adapted from L2AC, which is text-based. To adapt to an image-based classifier, object recognition models are used for feature extraction. Four encoders have been selected for further testing. AlexNet, ResNet50, ResNet152 and EfficientNet. Besides the encoder, the meta-classifier has undergone some changes as well to adapt to the open world recognition task. Finally, several different architectures designs for L2AIC are introduced, containing unique changes. The performance of these designs, as well as the encoders, are evaluated in Chapter 6.

4

Autonomous data collection

This chapter will introduce the autonomous data collection process to create a synthetic dataset for a novel object. The requirement for this protocol is that can be done autonomously by a robotic agent, without the help of human supervision. The protocol can be split into two main steps. First, a 3D model of the novel object is created. This is done with an RGB-D camera and the reconstruction algorithm Voxblox++ and will be explained in Section 4.1. For the second step, a protocol is described to create a synthetic dataset from this 3D model, elaborated in Section 4.2.

4.1. 3D reconstruction

In section 2.4.1 the eye-in-hand capturing technique was introduced. This is an efficient way for an agent to capture images from multiple angles. Using a robot for data-capturing is beyond the scope this project and the eye-in-hand is instead done by a human. Following the findings of the authors of KinectFusion [46] the object will be circled multiple times while recording, as shown in Figure 4.1. Using multiple loops will improve loop closure and reduce inconsistencies in the reconstructed object. The RGB-D camera used during reconstruction is the L515 lidar from the *Intel Realsense* series [28]. This lidar has a high depth accuracy and has a built-in IMU that can be used for pose estimation. For pose estimation the SLAM algorithm RTAB-Map [35] is used. RTAB-Map combines the IMU data, the RGB image and depth image for a robust pose estimation at each time frame.

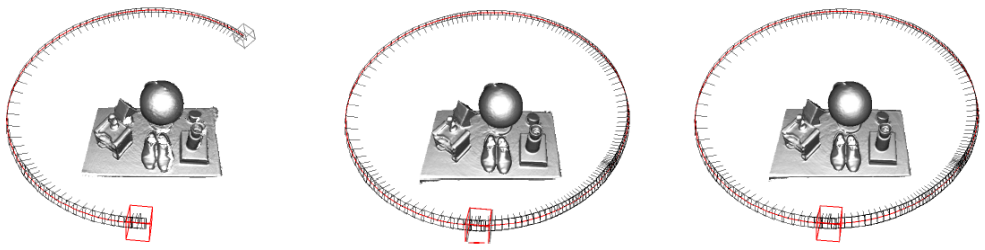


Figure 4.1: The reconstruction of a scene with the KinectFusion reconstruction algorithm [46]. The trajectory of the camera is shown by the red line and the axis of each camera frame. The red cube denotes the starting frame and the gray cube the final frame. *left:* A reconstruction based on a partial loop, some of the surfaces remain incomplete. *middle:* Reconstruction from one completed loop around the scene. The start and end frame are identical, but the red and gray cube have a slight mismatch. This is due to loop closure problems, resulting in some error in reconstruction in the cup. *right:* A reconstruction using multiple loops. The start and end frame have perfect overlap, reducing the loop closure artifacts.

4.1.1. Voxblox++

The Voxblox++ reconstruction algorithm, introduced in section 2.4.2, takes as input the RGB-D image and the pose estimation of the camera. At each time step t these inputs are processed. First, the depth image is segmented with the geometric depth segmentation. This results in the set of geometric segments $\mathcal{S}_t$. At the same time, the RGB image is segmented with Mask R-CNN [25]. The Mask R-CNN model has been loaded with 80 classes of the Microsoft COCO dataset [38]. With the segmented masks the geometric segments are refined and merged according to Equations (2.3) and (2.4). Using the pose estimation corresponding to the image frames the segments are placed into the global segmentation map (GSM). In the GSM they are assigned a unique geometric label l_k and, if available, a semantic label o_k. After scanning individual objects can be retrieved from the GSM and saved as a 3D model.

4.2. Synthetic data generation

To create realistic images of the 3D reconstructed model, Webots [44] is used. This is a robotic simulator capable of generating realistic environments for robotics, both in the physical sense and visual sense. For the application of this project a realistic visual representation is important. The object model is placed in an environment and,by using a virtual camera, images are captured of the object in various orientations and environments. This is further elaborated in the next section. In Section 4.2.2 several domain randomization techniques are listed to overcome the synthetic to real domain gap inherent in synthetic images.

Figure 4.2: An example of the robotic simulator Webots [44], with the TiaGo robot of PAL Robotics [49] deployed in a factory environment.

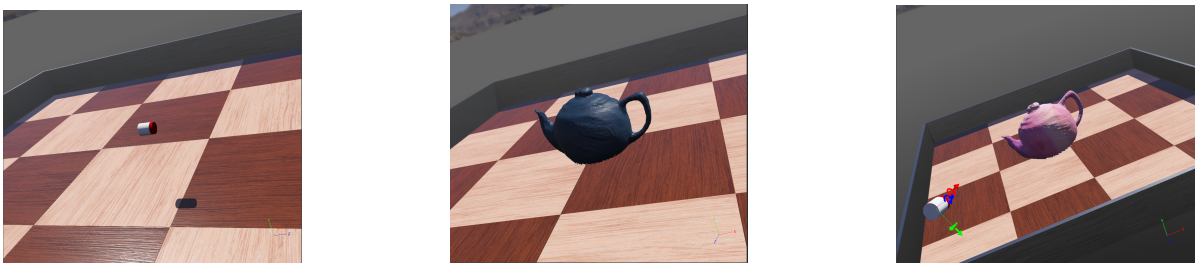
4.2.1. Camera placement

When capturing images with a virtual camera (Figure 4.3a) in Webots, it is has to be ensured the object is within the view frustum of the camera. The position of the camera with respect to the object is parameterized in spherical coordinates $(\rho_c, \theta_c, \phi_c)$. These spherical coordinates are transformed to Cartesian coordinates giving the cameras position with respect to the object $\mathbf{r}_c = x\,\hat{\mathbf{e}}_{o,x} + y\,\hat{\mathbf{e}}_{o,y} + z\,\hat{\mathbf{e}}_{o,z}$. To orient the cameras viewpoint towards the object, the unit z-axis of the camera $\hat{\mathbf{e}}_{c,z}$, is rotated towards the camera position vector, noted as R_1, shown in Figure 4.3c. To ensure the camera is positioned upright in the coordinate frame of the environment the camera is rotated along the z-axis according to Equation (4.1), where $\hat{\mathbf{e}}_Y$ is the y-axis of the coordinate frame of the environment and $\hat{\mathbf{x}}_c$ is the x unit vector of the camera in the coordinate frame of the environment.

$$\theta = \operatorname{acos}\left(\frac{(\mathbf{r}_c \times \hat{\mathbf{e}}_Y)\cdot\hat{\mathbf{x}}_c}{\|(\mathbf{r}_c \times \hat{\mathbf{e}}_Y)\|\,\|\hat{\mathbf{x}}_c\|}\right) \tag{4.1}$$

To prevent images being captured where the object is completely obstructed, for example, due to a wall, the segmentation module of the Webots camera is used. Besides an RGB image the camera also captures a segmentation mask. If the object is not present in the mask this indicates the object is obstructed and a new position is used.

A limitation of 3D scanning in combination with the eye-in-hand method is the bottom of the object cannot be scanned. To prevent the missing bottom of an object from being visible a final check is done. The angle between the normal vector of the bottom surface and the view direction of the camera is computed using the inner product. An angle below $\frac{\pi}{2}$ indicates the bottom is visible and the object is placed somewhere else.



(a) (b) (c)

Figure 4.3: *a:* The virtual camera of Webots. Frames are captured with the red side of the cylinder. *b:* An object with the *PRBAppereance* module applied to generate texture. *c:* The camera is automatically pointed towards the object. This is done using the protocol of Section 4.2.1.

4.2.2. Domain randomization

As introduced in [65] ,[26] and [29] domain randomization significantly improves the recognition performance when trained on synthetic data and can reduce the effects of reality gap. In the Webots environment several conditions are varied. First, the position $\mathbf{r}_o$ and the orientation $\mathbf{q}_o$ of the object are varied, where the orientation is given by roll, pitch and yaw values $(\alpha_o, \beta_o, \gamma_o)$. To acquire a realistic image the physics engine of Webots is used to let the object fall from the random position until it hits a surface. This prevents the creation of dataset with floating objects only. The physics engine assumes a spherical bounding box with radius of 0.1 m. When this bounding box hits a surface the camera will be place. The camera is placed relative to the object using polar coordinates $(\rho_c, \theta_c, \phi_c)$. As explained in the previous section this will create images from different viewpoints of the object. Furthermore, the texture of the object is varied, similar to [65] and [29]. Webots is capable of mimicking material textures such as metal, wood or textile using the *PRBAppearance* module, displayed in Figure 4.3b. To increase generalization of the object different colors and patterns are used for the object. These patterns are obtained from the internet using Google image search. The object, with random texture, is both placed in indoor and outdoor environments of Webots, using different lighting conditions. The complete procedure is listed in Algorithm 1.

Algorithm 1: Randomized synthetic data algorithm

```
Data: World, Object, Camera, N.
Result: N object images
begin
    World → loadWorld()
    for n ∈ N do
        World → setBackground()
        World → setLighting()
        Object → setRandomPosition()
        Object → setRandomOrientation()
        Object → setRandomTexture()
        Camera → setViewtoObject()
        if Camera → containsObjectSegmentation() then
            if Camera → objectBottomNotVisible() then
                Camera → captureImage()
```

4.3. Summary

In this chapter an overview is given of the autonomous data collection process. First, an object is scanned with the L515 lidar. For optimal loop closure the object is encircled multiple times. Using a robot for this process is out of the scope of this report. A human actor is used instead. With Voxblox++ the data gathered with the lidar is combined in a 3D reconstruction of the environment, the GSM. From the GSM a segmented 3D model of the object is extracted. The 3D model is placed in various environments of the Webots robotics simulator. The position, orientation and texture are varied while a simulated camera saves images of the object. This process autonomously creates a dataset containing a randomized images of the object.

5

Experimental setup

This chapter will elaborate on the experimental setup used for testing the L2AIC and autonomous data gathering protocol described in Chapters 3 and 4. First, the image datasets that are used for evaluation of the open world performance of L2AIC are introduced. This performance is evaluated based on several metrics, listed in Section 5.2. In Section 5.3 an overview of the experiments conducted on L2AIC is given. Chapter 3 introduced several designs of L2AIC and these experiment will test their performance in different settings. Section 5.4 describes the steps necessary to autonomously create a synthetic dataset from an object. Finally, the open world algorithm and the dataset creation are combined to design an on-the-job learning task in the last section.

5.1. Datasets

Image recognition is an extensive studied field of computer vision and many datasets have been designed for evaluation of recognition models. This report is focused on open world recognition and a suitable dataset should consist of more than a few classes because different classes are used in the train, validation and test set. This requirement excludes popular datasets such as MNIST and CIFAR-10.

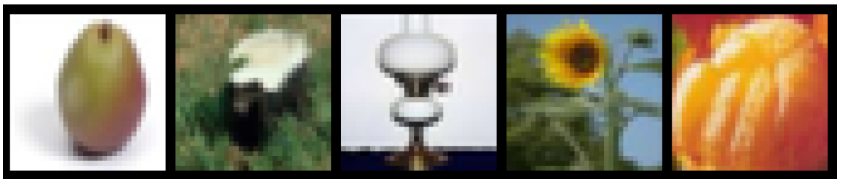


Figure 5.1: Example images of the CIFAR-100 dataset. Each image has a size of 32×32 pixels.

CIFAR-100

The first dataset used in this report is CIFAR-100 [33] and contains images from 100 different classes. The dataset is named after the Canadian Institute for Advance Research (CIFAR). These images are gathered from the internet using search engines such as Google and Flicker. Duplicates, drawings and cartoons have been removed and an additional check is done to verify the labels. Each class has 500 training images, or samples, and 100 samples for testing Resulting in a total of 60.000 images divided over 100 classes. The images have a size of 32×32 pixels with RGB color information. A few examples are shown in Figure 5.1. The 100 labels can be grouped in 20 superclasses with 5 subclasses per superclass. For example, the superclass *reptile*, with subclasses crocodile, dinosaur, lizard, turtle and snake [33]. The idea is that subclasses of a single superclass are more difficult to distinguish, making the dataset more challenging.

TinyImageNet
The TinyImageNet dataset [36] is a subset of the larger ImageNet dataset. It has been created for the Tiny ImageNet VisualRecognition Challenge and is considered to be simpler version of the ImageNet challenge. Instead of 1000 classes, TinyImageNet contains 200 classes. Each class has 500 training images, 50 validation images and 50 test images. The images are downsampled to a size of 64×64 pixels with RGB color information, shown in Figure 5.2. Downsampling the images will cause information loss, which increases classification difficulty [52].

Figure 5.2: Example images of TinyImageNet. Each image has a size of 64×64 pixels.

5.2. Evaluation metrics

To evaluate the performance of L2AIC in an open world setting several evaluation metrics are chosen. Because an open world setting is evaluated the traditional evaluation metric accuracy is not sufficient. In a true open world the number of object classes can be considered infinite. Since the number of known classes K is finite, the number of unknown classes U is also infinite [15]. This makes open world recognition a very imbalanced classification task. In this project the *F-1 score*, the *Wilderness Impact* (WI), and unknown and known recognition errors (ϵ_U and ϵ_U) are used to evaluate the performance of L2AIC.

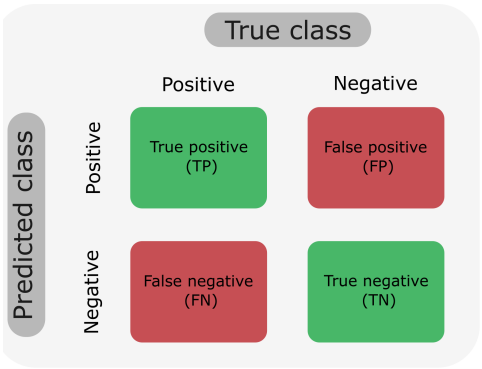


Figure 5.3: The confusion matrix presents true values and the predictions of a classifier. It gives a better understanding on how the classifier performs [50].

F1-score
The F1-score is the harmonic mean of the precision and recall of a model and is given by Equation 5.1 [22]. Give the confusion matrix in Figure 5.3, the precision can be defined as the ratio between the number of true positives and the number of all predicted positives (TP + FP). The recall is defined as the ratio between the number of true positives and the number of actual positives (TP + FN). Individually these metrics can cause bias. For example, a high recall can be achieved when all classes are classified positive, but at the cost of a large number of false positives. F1-score combines both measures to get a more reliable evaluation metric, particularly for imbalanced datasets.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.1}$$

Similar to the study of [67] the *weighted F1-score* (Equation (5.2)) is used for model evaluation. Weighted F1-score takes into account the number of samples for each class, compensating for a very large size of the unseen class set.

$$F1^{weighted} = \sum_{c \in C} \frac{N_c}{\sum_{c \in C} N_c} \cdot F1_c \tag{5.2}$$

Wilderness Impact
In [15] the concept of *Wilderness* is introduced. The *Wilderness Ratio* of a dataset is defined as the ratio between known (N_K) and unknown (N_U) samples, as shown in Equation (5.3). A dataset under closed condition has a Wilderness Ratio of zero. Adding samples of unknown objects will increase the Wilderness and make classification more difficult. The *Wilderness Impact*, (Equation (5.4)), describes the change in precision between an open and closed world setting. Where FP_o is the number of unknown samples incorrectly classified as known, FP_c the known samples misclassified as another class and TP_c as the correct classified samples. When the number of unknown samples increases the probability of classifying these unknown samples as a known class increases and the Wilderness impact will increase. A good classification model has little increase in WI as the Wilderness Ratio grows.

$$Wilderness\ Ratio = \frac{N_U}{N_K} \tag{5.3}$$

$$\begin{aligned} WI &= \frac{Precision\ in\ closed\ set}{Precision\ in\ open\ set} - 1 \\ &= \frac{TP_c}{TP_c + FP_c} \Big/ \frac{TP_c}{TP_c + FP_c + FP_o} - 1 \\ &= \frac{FP_o}{TP_c + FP_c} \end{aligned} \tag{5.4}$$

Classification errors
In [6] the open world error ϵ_{OW} is introduced. This error combines the multi-class classification error in closed set, ϵ_K, with prediction error on unknown samples ϵ_U. As explained in Section 2.1 the two main challenges of open world recognition are rejecting unknowns and classifying known objects. Although the open world error gives an overall indication of the performance on these challenges it does cannot give performance of the individual challenges. In this report has been chosen to use both ϵ_K and ϵ_U. Recall that open set risk, Section 2.1, consists of two component, the open space risk and empirical risk. The metrics ϵ_K and ϵ_U can give an indication of performance with respect to the empirical risk or open space risk respectively.

$$\epsilon_{OW} = \epsilon_K + \epsilon_U = \frac{1}{N_K} \sum_{i=1}^{N_K} [\hat{y}_i \neq y_i] + \frac{1}{N_U} \sum_{j=N_K+1}^{N_U} [\hat{y}_i \neq unkown] \tag{5.5}$$

5.3. L2AIC

The training procedure of the L2AIC algorithm is a bit more complex than regular object recognition models and consists of several steps. First, in Section 5.3, the data is split in encoder data $\mathcal{D}^{]}$ and data for the meta-classifier of L2AIC D^m. Then, the pretrained encoder is fine-tuned on the samples from $\mathcal{D}^{]}$ (Section 5.3). Finally, in Sections 5.3 and 5.3 the training procedure of the meta-classifier is described. Results can be found in Section 6.1.

Data split

Before training can start the dataset is split into data for fine-tuning the encoder $\mathcal{D}^e$ and data for used by the L2AC meta-classifier $\mathcal{D}^m$. The classes of the encoder and classes of the meta-classifier are mutually exclusive, $\mathcal{C}^{fn} \cap \mathcal{C}^{\mathcal{M}} = \varnothing$. The encoder data is split in a train set $\mathcal{D}^e_{trn}$ and a validation set $\mathcal{D}^e_{val}$.
The meta-classifier data is divided in three parts, a train, validation and test set. During each training phase, the class set is distributed over a memory $\mathcal{C}^{\mathcal{M}}$ and a set of input samples $\mathcal{S}$. The memory can consist of two sets of classes. The first set, $\mathcal{C}^{\mathcal{K}}$, is the set of known classes that has been used for training. The second set, $\mathcal{C}^{\mathcal{N}}$, is the set of new classes that are added to the memory after training. The latter class set, the option to incrementally learn new classes, is what makes the L2AIC an open world algorithm. The set of input samples contains besides $\mathcal{C}^{\mathcal{K}}$ and $\mathcal{C}^{\mathcal{N}}$ a set of unknown classes $\mathcal{C}^{\mathcal{U}}$. In the original paper [67] the algorithm is solely evaluated on the performance on $\mathcal{C}^{\mathcal{N}}$. This is to prove that the meta-classifier is able to generalize over all features instead of class-specific features from the training set and is suitable for open world recognition. Although this is an important property of the meta-classifier, in this project also the performance on $\mathcal{C}^{\mathcal{K}}$ is evaluated. The motivation of this approach is that it is not desirable to start deployment with an entire new memory. Using the same memory for each training phase is more intuitive. In Table 5.1 an overview of the classes for each training phase is given.

Table 5.1: Classes in the memory of $L2AIC$ during training, validation and testing, for the different datasets

Dataset	Phase	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{U}}$
CIFAR-100	**Train**	50	40
	Validation	50	10
	Test	50	20
TinyImageNet	**Train**	80	80
	Validation	80	20
	Test	80	100

Encoder fine-tuning

In the first training step the encoder is fine-tuned using the training data of $\mathcal{D}^e_{train}$. The fine-tuning of the encoder is done on a model pretrained on ImageNet for 50 epochs, with a batch size of 100. An SGD optimizer with a learning rate of 0.001 is used and this is only applied to the final few layers of the model. All image samples are upscaled to 224×224 pixels. As this fine-tuning is a regular multi-class classification task the cross-entropy function is used as a loss function. The validation data $\mathcal{D}^e_{val}$ is used to track generalization performance of the encoder. The best model is selected using early stopping based on the F1-score of the validation set. After finishing the fine-tuning the encoder is used to create the feature vectors required for the memory and input samples for each training phase. The training images have been rescaled to 224×224 pixels and each feature vector is normalized between 0 and 1. The results of this experiment are listed in Section 6.1.

Meta-classifier training
After having created the required memory and input samples for all training phases and finetuning of the encoder, the meta-classifier is trained. The meta-classifier will be trained for 400 epochs with a batch size of 256. The optimization algorithm is Adam and a learning rate of 0.0001 is used. Early stopping on the F1-score is used to retrieve the best model. As this is a binary classification task, the binary cross-entropy loss function is used. During training each input sample is compared against n negative classes and 1 positive class. The n negative classes are found using the ranker, as explained in section 5.3. This leads to a very unbalanced training set and is countered by setting the positive weight (p) in the binary cross entropy loss equal to n. During testing the input sample will be compared to the top n similar samples.

Training procedure variations
Besides the default training method, introduced in the previous paragraph, four variations in the training procedure are introduced. Two variations are regarding the training procedure of the meta-classifier and the other two are variations in the ranking procedure. The first variation, *two-stage* training, will train the meta-classifier in *two steps*. The first N epochs only the parameters of the matching layers are trained. Then after N epochs, the best parameter configuration is selected, based on early stopping. In the second step the whole meta-classifier is trained, including both matching layer and the aggregation layer again for N epochs. The idea of this two stage training is that the matching layer is better trained to separate positive classes from negative classes as the loss function is directly applied to its output. The second variation, *freeze-ml*, also requires training in two steps. But after the matching layer has been trained and the best model is selected the parameters of the matching layer are *frozen*, meaning they are not longer used in the optimization process. In the second step, now only the aggregation layer is trained.

To improve generalization of the meta-classifier two variations of the ranker are introduced. These variations change the ranker during the training phase. Given input sample x_i with class label c_i, the the meta-classifier is presented with n classes $c_{1:n}$, where $c_j \in c_{1:n} \neq c_i$, once with class c_j, with $c_j = c_i$ for each input sample. A class c_j, with $c_j \neq c_i$ is denoted as a *negative* class and $c_j = c_i$ is considered a *positive* class. In *ranker-default*, the ranker retrieves the k most similar samples for the n negative classes and the single positive class. The *ranker-reverse* version changes the ranking of the positive class. Instead of finding the k most similar samples, the k least similar samples are considered. The idea behind this variation is that the meta-classifier is more challenged to find similarities between feature vectors and will generalize better. The second version, *ranker-extend*, is to counter the class imbalance in the training dataset. For each positive class n negative classes are present. In this version the ranker will also create n positive class entries by using the $n \times k$ similar samples from the positive class. The first entry consists of the k most similar samples, the next entry consists of the next k similar samples etc. Besides countering class-imbalance this version will also improve generalization by presenting more difficult positive classes. The results of these variations can be found in Section 5.3.

5.4. Self-supervised data gathering

The objects used for reconstruction are listed in Figure 5.4. These objects are a combination of common household items and unique objects. As stated in Section 4.1 the L515 camera is held in hand and moved several times around the object to reduce loop closure problems. The camera has an RGB resolution of 960×540 and depth resolution is 640×480. The reason for this discrepancy is the hardware limits of the camera, where depth and RGB resolution are not the same [28]. Fortunately, this is solved within the software of the camera. The aligned depth and RGB images, together with the pose estimation of the camera are recorded for reconstruction with Voxblox++.

Figure 5.4: The objects selected for reconstruction are a mix between common household items and unique objects.

Voxblox++ has been originally designed for reconstructing large maps containing several objects. In this project the focus is on single objects. To be able to capture enough detail of the scanned object the voxel size v has been decreased from 2 cm, in the original paper [23], to 1 mm. This more detailed voxel grid comes at the cost of more memory requirements, but because of the smaller scene captured this is not significant. To avoid capturing too many background objects, and increase the reconstruction speed, the depth image is clipped between 0.1 and 0.5 meter. The truncation distance δ is set to 5 times the voxel size. Each voxel is updated according to the set of Equations (2.5). The final output of Voxblox++ will be a GSM containing a segment of an object and a ground surface, where the ground surface can be discarded as it is not of interest. Reconstructed models are visually evaluated. Important are the quality of the loop closure and if the object has been correctly segmented from the ground surface.

From the reconstructed objects a synthetic dataset is created with Webots using the domain randomization protocol of Algorithm 1. Table 5.2 the minimum and maximum values for the camera placement $(\rho_c, \theta_c, \phi_c)$ and object orientation $(\alpha_o, \beta_o, \gamma_o)$. Where camera placement is done according to spherical coordinates and object orientation with roll, pitch and yaw. Several combinations of the domain randomization protocol of Algorithm 1 are used to create the dataset to study the influence of individual components. The results of this dataset are shown in Section 6.2.

Table 5.2: Minimum and maximum values for the object and camera placement and orientation for $\mathcal{D}^{syn}$.

Parameter	Min value	Max value
ρ_c	0.4 m	1.0 m
θ_c	$-\frac{1}{4}\pi$	$\frac{1}{4}\pi$
ϕ_c	$-\frac{1}{4}\pi$	$\frac{1}{4}\pi$
α_o	$-\pi$	π
β_o	$-\frac{1}{4}\pi$	$\frac{1}{4}\pi$
γ_o	$-\frac{1}{4}\pi$	$\frac{1}{4}\pi$

5.5. On-the-job recognition

The on-the-job recognition task combines the open world algorithm L2AIC and the self-supervised data gathering. This is done in three steps. First, the L2AIC performs classification on a test set with classes $\mathcal{C}^{\mathcal{M}}$ and $\mathcal{C}^{\mathcal{U}}$. The model has to correctly classify classes from $\mathcal{C}^{\mathcal{M}}$ and reject the unknowns $\mathcal{C}^{\mathcal{U}}$. Secondly, from the set of unknown classes $\mathcal{C}^{\mathcal{U}}$, object class c_o is picked for reconstruction. A synthetic dataset is made from this object class using the self-supervised data gathering method from Section 5.4. Finally, this dataset is added to the memory, now containing $K + 1$ classes. The object class c_o should now be known to the model. The performance of L2AIC is evaluated again, and it should no longer reject images from this class.

At the first step, L2AIC will be trained on TinyImageNet. $\mathcal{C}^{\mathcal{M}}$ will contain 70 classes with images from TinyImageNet. During evaluation the model is tested on images from these 70 classes plus an additional set of unknown classes $\mathcal{C}^{U}$ containing 31 different classes. 30 of these unknown classes contain images from TinyImageNet. The thirty-first class, c_o, contains images taken from an object with a camera. The L2AIC algorithm should reject all 31 classes of $\mathcal{C}^{U}$ and correctly classify all classes from $\mathcal{C}^{\mathcal{M}}$. In the second step, the object from c_o is used to create a synthetic dataset. Using the procedure from Section 5.4 two different datasets are created. The first dataset, $\mathcal{D}^{syn}$ will vary the object distance from 0.4 m to 1.0 m. The second set, $\mathcal{D}^{syn,close}$, will create a dataset where the object is closer to the camera, varying from 0.2 m to 0.7 m. In the final step, this new dataset is added to the memory of L2AIC. The algorithm now has 71 classes stored in memory, the original 70 classes extended with class c_o. The performance of the updated L2AIC is evaluated on the same test set as in step one, but now capable of classifying c_o.

Besides the two synthetic dataset, two additional datasets of c_o are created for addition to the memory. The first set, $\mathcal{D}^{Tiny,memory}$ is the training set of TinyImageNet for this specific class c_o. The second set, $\mathcal{D}^{photos,memory}$ is a set of photos taken with a camera of the object during the 3D reconstruction process. This is to compare performance between different image domains and the influence of the domain gap. Three domains are used in this report. The synthetic domain, with datasets $\mathcal{D}^{syn}$ and $\mathcal{D}^{syn,close}$. The real domain, with datasets $\mathcal{D}^{photos,memory}$ and $\mathcal{D}^{photo,test}$. The last domain is the TinyImageNet domain, containing the dataset of TinyImageNet. The results of the on-the-job recognition task are discussed in Section 6.3. Examples of $\mathcal{D}^{photos,memory}$, $\mathcal{D}^{photos,test}$ and $\mathcal{D}^{Tiny,memory}$ can be found in Appendix B.

6 Results

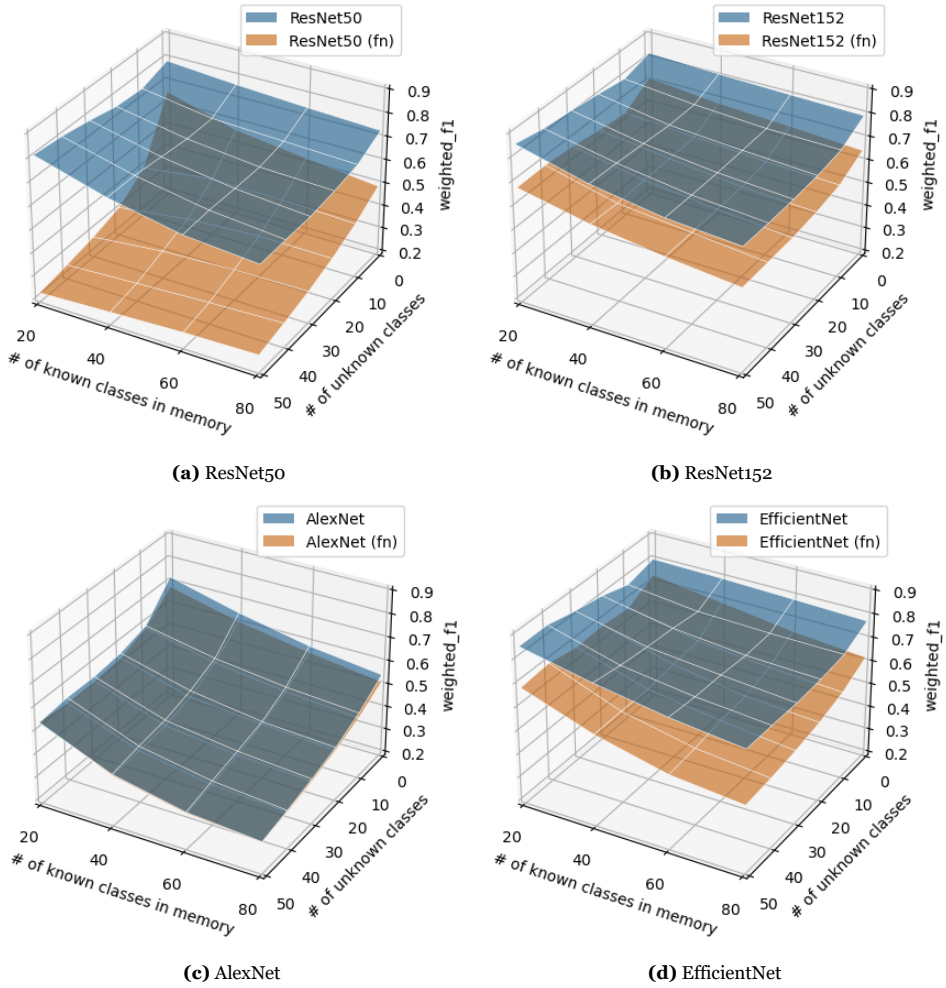
In this chapter the results from the experiments introduced in the previous chapter are evaluated. First, in Section 6.1, the open world classifier L2AIC is tested under varying conditions and variations. Secondly, in Section 6.2 the results of the autonomous data generation is presented. Thirdly, L2AIC is tested in the on-the-job recognition task. Using a synthetic dataset of a 3D model of a novel object, the performance of the classifier is evaluated on this new object. The chapter concludes with a small summary of the results.

6.1. L2AIC

In this section the best performing model of the L2AIC classifier is evaluated. This evaluation is done by assessing the open world capabilities of the classifier based on the metrics of Section 5.2. By varying the number of known and unknown classes, the rejection and classification capabilities are tested. Each experiment shows the average of 10 runs with random initialization. First, a comparison of different encoders is done, as well as the influence of additional fine-tuning. Secondly, using the best performing encoder, different model architectures of the meta-classifier are compared. Third, the influence of the ranking hyperparameters n and k are evaluated. Then, in Section 5.3 different variations of the training procedure are evaluated. Next, the best performing L2AIC model is compared to the original model of the authors of [67] in Section 6.1. This section concludes with a brief summary of the findings in Section 6.1

Encoders

In Figure 6.1 an overview is given of the performance of L2AIC with different encoders. The weighted F1-score is calculated while varying the number of known classes in memory and the number of unknown classes presented to the algorithm. The figure shows L2AIC using the ResNet50, ResNet152, AlexNet and EfficientNet encoders. The *fn* label denotes fine-tuning on 50 separate classes of the TinyImageNet dataset.



(a) ResNet50

(b) ResNet152

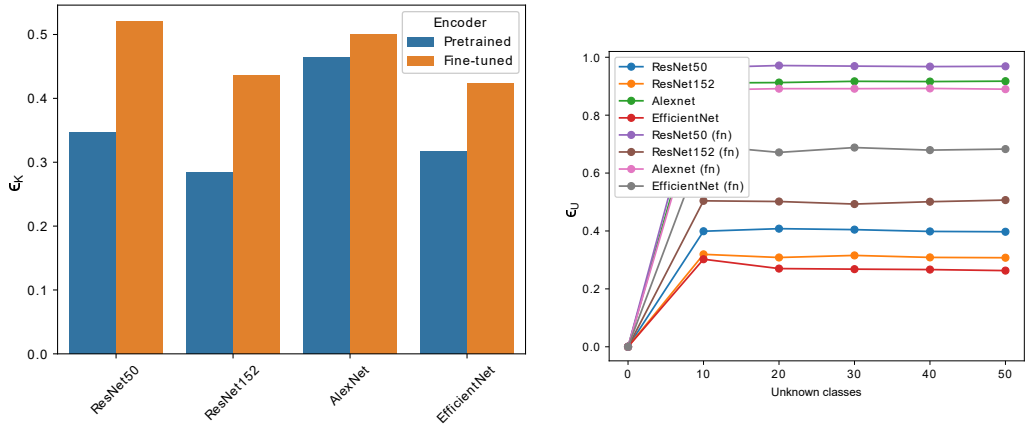
(c) AlexNet

(d) EfficientNet

Figure 6.1: In this graph the weighted F1-score is plotted as a function of the number of known classes present in memory and the number of unknown classes presented to the algorithm. Each subplot shows the performance of L2AIC with a different encoder. A pretrained encoder with weights of ImageNet is used. The *fn* label denotes the encoder has been additionally fine-tuned on 50 classes from the TinyImageNet dataset.

Figure 6.1 shows that fine-tuning significantly reduces the F1-score for all encoders. This decrease in performance can be explained by overfitting of the encoder on $\mathcal{C}^{fn}$, the set of classes used for fine-tuning. Because $\mathcal{C}^{fn} \notin \mathcal{C}^{\mathcal{M}}$, fine-tuning might cause the encoder to extract features only important to $\mathcal{C}^{fn}$. This compromises the advantage of an encoder pretrained on ImageNet, which functions as a general feature extractor. Furthermore, AlexNet performs significantly worse than the other three encoders. This is not surprising, given that AlexNet was one of the first deep learning classifiers. Although some design choices from AlexNet have become standard in deep neural networks [2], much improvements have been made on model design since the introduction of AlexNet in 2012.

In Figure 6.2 the known and unknown prediction error for different encoders are given. Similar to Figure 6.1 this shows that fine-tuning decreases performance. Furthermore, in general AlexNet performs worse than the other encoders. EfficientNet and ResNet152, pretrained only, perform the best. ResNet152, pretrained, has the lowest ϵ_K, and thus the best classification performance. However, pretrained EfficientNet is better suited for rejecting unknown classes, given by the lowest ϵ_U. This is in line with the theory of open set risk of Section 2.1. A trade-off exists between the empirical risk and open space risk.

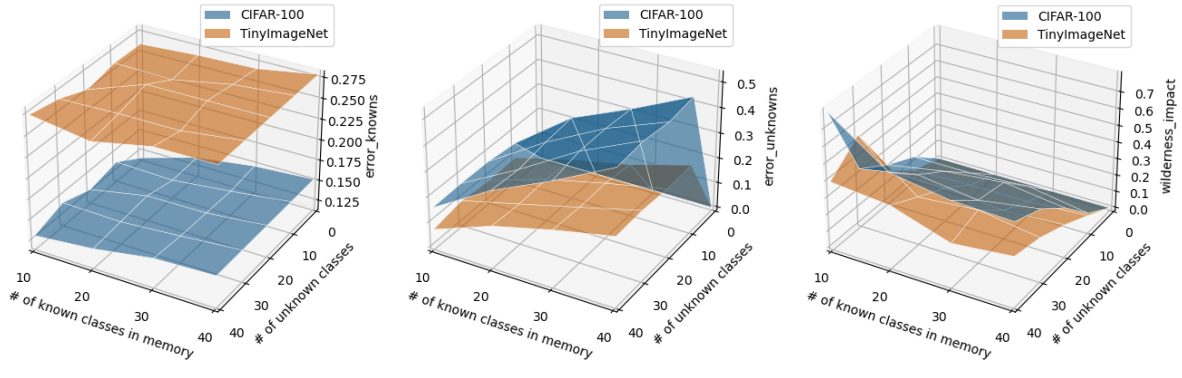


(a) **(b)**

Figure 6.2: (a) The known classification error (ϵ_K) of L2AIC using different encoders. The algorithm has 80 known classes stored in memory. Used encoders are loaded with pretrained weight on ImageNet and some have undergone additional fine-tuning on a subset of classes of TinyImageNet. Note that a lower error is means better performance. **(b)** The unknown classification error (ϵ_U) of L2AIC using different encoders All encoders have been pretrained on ImageNet. The label (fn) denotes if the encoder has been fine-tuned on a subset of classes of the TinyImageNet dataset. The model has 80 classes in memory and the number of unknown classes is varied.

As stated in the previous paragraph, using an encoder pretrained on ImageNet allows the model to extract general features. However, this might cause a bias towards the performance on TinyImageNet as this is a subset of ImageNet. To evaluate this bias the performance of L2AIC on both CIFAR-100 and TinyImageNet is compared, shown in Figure 6.3. For this comparison a pretrained ResNet152 encoder is used, without additional fine-tuning. For both datasets the L2AIC algorithm has been trained on 40 classes. Figure 6.3 shows ϵ_U and ϵ_K and the Wilderness Impact for varying numbers of known and unknown classes. From Figure 6.3a and 6.3b it becomes clear that ϵ_K on CIFAR-100 is lower, but the L2AIC algorithm is less capable of rejecting unknown classes as the ϵ_U is higher compared to TinyImageNet. Figure 6.3c shows that by presenting more unknown classes, the Wilderness Impact on the CIFAR-100 dataset increases faster compared to TinyImageNet, a result of the higher ϵ_U. Unknown samples will have a larger negative impact on classification of CIFAR-100.

From these results can be concluded that fine-tuning has an adverse effect on the performance, for further experiments, only pretrained encoders are considered. Furthermore, overall performance of AlexNet is significantly worse and AlexNet will not be considered in further experiments.



(a) **(b)** **(c)**

Figure 6.3: A surface plot of performance of the L2AIC algorithm on the datasets CIFAR-100 and TinyImageNet. **(a)** The known prediction error ϵ_K. **(b)** The unknown prediction error ϵ_U. **(c)** The Wilderness Impact.

Table 6.1: The F1-score, unknown classification error and the known classification error for L2AIC using different meta-classifier architectures and encoders. The L2AIC has 40 classes stored in memory and is tested on an additional 40 unknown classes. The performance is given for both $\mathcal{C}^{\mathcal{K}}$ and $\mathcal{C}^{\mathcal{N}}$. The encoders are loaded with pretrained weights only. The five best performing models have been printed in bold font.

Meta-classifier architecture	Encoder	Weighted F1		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
L2AIC-default	*RN50*	0.684	0.584	0.348	0.541	0.304	0.299
	RN152	0.470	0.653	0.828	0.403	0.18	0.299
	EN	**0.737**	**0.617**	**0.107**	**0.348**	**0.399**	**0.420**
L2AIC-cosine	*RN50*	0.677	0.557	0.403	0.599	0.278	0.291
	RN152	**0.746**	**0.597**	**0.239**	**0.563**	**0.285**	**0.245**
	EN	0.680	0.511	0.440	0.701	0.245	0.254
L2AIC-no-lstm	*RN50*	0.687	0.569	0.352	0.580	0.296	0.281
	RN152	0.505	0.664	0.778	0.401	0.188	0.282
	EN	**0.740**	**0.639**	**0.068**	**0.203**	**0.426**	**0.487**
L2AIC-extended-similarity	*RN50*	0.610	0.049	0.503	0.984	0.297	0.907
	RN152	0.557	0.035	0.666	1.000	0.235	0.903
	EN	0.625	0.322	0.537	0.845	0.231	0.470
L2AIC-smaller-fc	*RN50*	0.718	0.484	0.238	0.691	0.332	0.331
	RN152	**0.706**	**0.588**	**0.349**	**0.537**	**0.261**	**0.293**
	EN	**0.769**	**0.549**	**0.233**	**0.614**	**0.243**	**0.291**
L2AIC-abssub	*RN50*	0.468	0.351	0.751	0.840	0.289	0.403
	RN152	0.295	0.285	0.991	0.924	0.249	0.395
	EN	0.490	0.339	0.748	0.836	0.255	0.450
L2AIC-concat	*RN50*	0.505	0.307	0.732	0.856	0.227	0.484
	RN152	0.454	0.368	0.837	0.812	0.194	0.422
	EN	0.576	0.364	0.631	0.825	0.222	0.399

Meta-classifier architecture

In Table 6.1 the Weighted F1-score, ϵ_U and ϵ_K are listed for the different meta-classifier architectures introduced in Section 3.3. For each architecture the encoders ResNet50, ResNet152 and EfficientNet have been loaded with pretrained weights only. Both the results for $\mathcal{C}^{\mathcal{K}}$ and $\mathcal{C}^{\mathcal{N}}$ have been listed. Performance on samples from $\mathcal{C}^{\mathcal{K}}$ is better than for $\mathcal{C}^{\mathcal{N}}$, which is to be expected. The model has never seen these different classes before during training. This decrease is, for some specific models, not that large. This suggests the meta-classifier is able to learn to correctly recognize similarities between features and able to generalize this knowledge to new feature vectors as well. In Appendix C a more extensive overview of the individual architecture is listed.

What stands out is that the architectures containing changes in the similarity function from the matching layer, L2AIC-extended-similarity, L2AIC-abssub and L2AIC-concat perform significantly worse than the other models. With the exception of L2AIC-cosine, which performs slightly better. This decrease of performance is due to a combination of overfitting of the model on the training data and early stopping. Early stopping ensures the model leading to the highest performance on the validation is eventually saved as the final model after training. Due to overfitting the validation performance will decrease immediately after a few epochs and will save the model without trained weights.
A particular case is the L2AIC-cosine model. The L2ACI-cosine with a ResNet152 encoder has a relatively good performance. Replacing the matching layer with a cosine similarity function removes the trainable parameters of the matching layer. This effectively means the matching layer is left out and the aggregation layer predicts a classification based on direct input of the cosine similarity of the ranker. However, this input, a single cosine similarity value, does not contain enough distinguishable information for the aggregation layer to learn from. As a result, learning the correct weights for the network of L2AIC-cosine is not possible.

Based on Table 6.1 the five best performing models (printed in bold font) have been selected. The output distribution of these models on input samples is displayed in Figure 6.4. The colors denote the distribution of input samples of known and unknown classes. The memory of L2AIC contains 40 classes from $\mathcal{C}^{\mathcal{N}}$, they have not been seen during training. The input images contain an additional 40 unknown classes. Output scores below the classification boundary $h = 0.5$ get classified as unknown. Not all outputs are distributed fully along the range of 0 and 1. This is due to early stopping. The loss function, binary cross-entropy, ideally forces the known and unknown distribution apart by updating the parameters after each epoch. However, this takes several iterations and if the training is stopped before this point the output range is smaller.

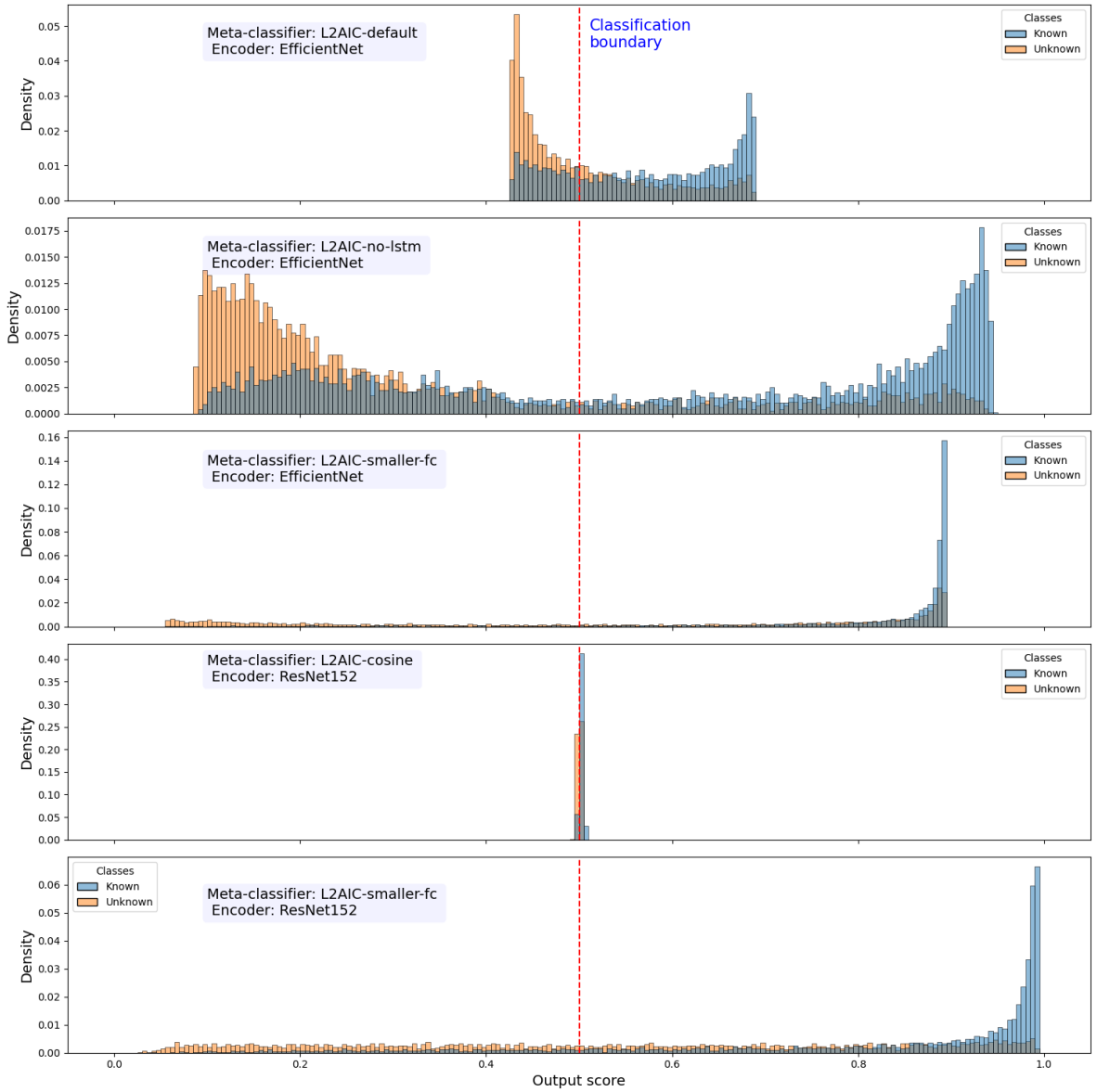


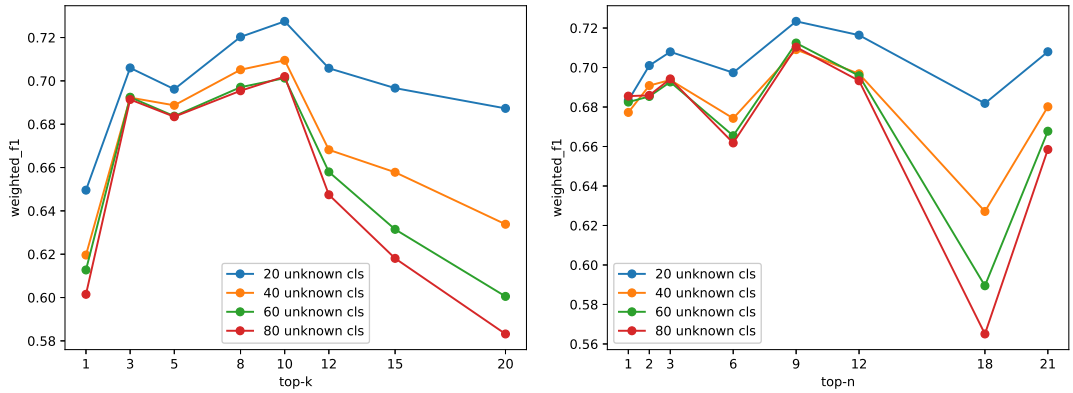
Figure 6.4: The normalized distribution of the output score of different L2AIC models. The memory contains 40 classes and the model is tested on an additional 40 classes. The different class memory is used. Each input sample is assigned a final output score. A score below the classification threshold $h = 0.5$ is classified as unknown. The colors denote the ground truth class of the input samples.

As stated in previously, the L2AIC-cosine was not able to learn to distinguish between known and unknown classes. This is also visible in Figure 6.4 where the output distributions of known and unknown classes are almost identical and no proper distinction between known an unknown can be made. The seemingly good performance can be explained by the aggregation layer. The training the aggregation layer of this model effectively moves the output distribution to a position where the classification boundary will optimally classify the output distribution, without being able to distinguish known and unknown classes.

The other four models are capable of separating the known and unknown score distribution. Again a trade-off exists between rejection and classification performance. The L2AIC-default and L2AIC-no-lstm models are biased towards the rejection of classes. A large part of the known distribution is at the wrong side of h, resulting in more known classes to be wrongfully identified as unknown. The L2AIC-smaller-fc model is biased towards classifying known classes. The known output distribution of this model is almost complete at the correct (right) side of the classification boundary, while unknown output scores are more spread out. This is in line with the theory of open set risk, the trade-off between open space risk and empirical risk. Based on Table 6.1 and Figure 6.4 the model with the best classification performance is L2AIC-smaller-fc with ResNet152. The L2AIC-no-lstm with EfficientNet gives the best rejection performance. These are use for the remainder of the report.

Ranker hyperparameters

The L2AIC model according to the *n-way k-shot* principle. An input sample is compared to k different images from n classes of the memory. As elaborated in Section 3.2 these n classes and k images are retrieved by the ranker using cosine similarity between feature vectors. This section will review the choice of these hyperparameters. From the original study of L2AC [67], the reference values are $n = 9$ and $k = 5$. The hyperparameters define how many samples and classes the ranker passes on to the meta-classifier.



(a) **(b)**

Figure 6.5: a The weighted F1-score as a function of k. The L2AIC model has 80 classes in memory and the number of unknown classes is varied. **b** The weighted F1-score as a function of n. The L2AIC model has 80 classes in memory and the number of unknown classes is varied.

Figure 6.5 shows the weighted F1-score for as function of k and n respectively, while varying the number of unknown classes. The L2AIC-smaller-fc model with pretrained ResNet152 encoder is used. Selecting the optimal hyperparameters comes down to a trade-off between rejection and classification. A small value for k, the number of samples for each class, will lead to a poor classification performance. When selecting $k = 1$, the matching layer of the meta-classifier generates only one similarity score, based on a single image per class. This makes the aggregation layer redundant as it has nothing to aggregate and is prone to error. This single image might be an outlier resulting in an incorrect classification. At the other end, when k becomes larger, the rejection performance will decrease. Figure 6.5a shows that $k = 10$ balances this trade-off leading to an optimal performance.

Along the same line can be argued that $n = 9$ is the optimal value for the number of classes selected by the ranker. The ranker uses the average feature vector of a class to find its similarity to an input sample. When $n = 1$ the ranker might pass on the wrong class to the meta-classifier, making it impossible for the meta-classifier to give a correct prediction. Increasing n will increase the chance that the correct class is passed on to the meta-classifier, increasing the classification performance. However, adding more classes to the meta-classifier will increase the ϵ_U as the possibility an unknown object gets classified as a known class grows.

Combining above results leads to the conclusion that the hyperparameter set of $k = 10$ and $n = 9$ leads to the best performance of L2AIC.

Table 6.2: The performance of different ranking and training variations on the TinyImageNet dataset. The models have 80 known classes in memory and are tested on an additional 80 unknown classes. The classes stored in memory are the same as used for training the model, $\mathcal{C}^{\mathcal{K}}$. For all variations a pretrained ResNet152 encoder has been used in combination with the L2AIC-smaller-fc model architecture.

Setting	**Weighted F1**	ϵ_U	ϵ_K
Default training	0.706	0.349	0.261
Reverse ranking	0.312	0.126	1.000
Extended ranking	0.596	0.589	0.250
Two step training	0.661	0.461	0.254
Freeze matching layer	0.627	0.531	0.253

Training procedure

In Table 6.2 the performance of L2AIC using training procedures is listed as well as the default settings. The L2AIC-smaller-fc model is used in combination with a pretrained ResNet152. The model has 80 classes stored in memory and these classes are the same classes used for training. An additional 80 unknown classes are used for testing the performance. The performance is measured by the weighted F1-score, the unknown classification error and the known classification error. The use of the reverse ranking method during training has a strong negative effect on the performance. With this method the ranker selects the least similar samples from the positive class of each input sample, while it selects the most similar samples from the negative classes. This effectively learns the classifier to label the least similar classes as a known class, which is the opposite of how a good classifier should perform. The model is not able to correctly classify any known classes, given by a ϵ_K of 1.00.

The three remaining new methods, extended ranking, two-step training and freezing the matching layer show better results. However, when compared to the default training method, there is no overall improvement. Although the other methods have a slightly lower ϵ_K, this comes at the cost of an increased ϵ_U and lower F1-score. For further experiments only the default training procedure is considered.

Table 6.3: A comparison of performance of L2AC and L2AIC on the CIFAR-100 dataset. The model has 15 classes stored in memory and is presented with an additional 10 unknown classes.

Model	Encoder	Weighted F1		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
L2AC	*EN*	0.799	0.642	0.295	0.576	0.144	0.201
L2AC	*RN152*	0.795	0.668	0.318	0.432	0.135	0.268
L2AIC-no-lstm	*EN*	0.805	0.681	0.066	0.204	0.281	0.379
L2AIC-smaller-fc	*RN152*	0.818	0.654	0.283	0.527	0.118	0.221

L2AC comparison

Based on the results of previous paragraphs two versions of L2AIC have been selected on their best overall performance. The firs model is the L2AIC-no-lstm model with a pretrained EfficientNet encoder. The hyperparameters are set at $k = 10$ and $n = 5$. No variation in training procedure is used. The second model is the L2AIC-smaller-fc with a pretrained ResNet152 encoder. Again, the hyperparameters are set at $k = 10$ and $n = 5$ and no variation in the training procedure is used. The first model is more biased towards rejecting unknowns, while the second model is more biased towards correctly classifying known objects. A comparison is made with the original L2AC model from [67], with two different pretrained encoders, EfficientNet and ResNet152. The hyperparameters are kept at their original values $k = 5$, $n = 9$. Table 6.3 and 6.4 list the performance on the CIFAR-100 and TinyImageNet dataset respectively. Using CIFAR-100 15 classes are stored in memory and 10 unknown classes are used. For TinyImageNet 40 classes are in memory with an additional 40 unknown classes.

The results show a difference in performance using $\mathcal{C}^K$ and $\mathcal{C}^N$. In the case of $\mathcal{M} = \mathcal{C}^K$, the smaller-fc has the best overall performance for both datasets. However, this is not the case for $\mathcal{M} = \mathcal{C}^N$. Now, the no-lstm models is more favorable in terms of overall performance. This is mainly because of the good rejection performance of unknown classes and comes at the cost of a higher known classification error.

Table 6.4: A comparison of performance of L2AC and L2AIC on the TinyImageNet dataset. The model has 40 classes stored in memory and is presented with an additional 40 unknown classes.

Model	Encoder	Weighted F1		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
L2AC	*EN*	0.691	0.622	0.41	0.481	0.233	0.286
L2AC	*RN152*	0.696	0.583	0.378	0.555	0.251	0.288
L2AIC-no-lstm	*EN*	0.727	0.642	0.057	0.12	0.458	0.545
L2AIC-smaller-fc	*RN152*	0.767	0.625	0.232	0.469	0.247	0.293

Section summary

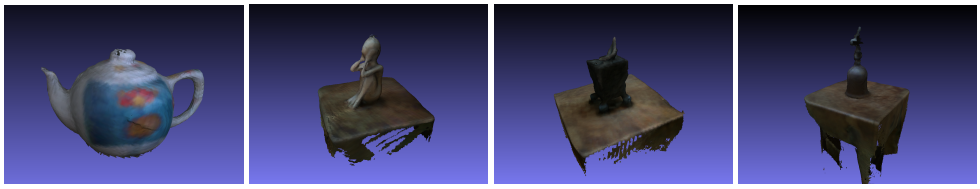
In this section the performance of L2AIC has been evaluated. It was found that additional fine-tuning on the encoder has an adverse effect on the performance. The optimal set of hyperparameters are $k = 10$ and $n = 9$. Changing the training procedure has a negative on the overall performance. The best performing models were found to be L2AIC-no-lstm with pretrained EfficientNet encoder and L2AIC-smaller-fc with pretrained ResNet152 encoder. These models are compared to the original L2AC algorithm and show an increase in performance.

6.2. Autonomous data gathering

In 5.4 six objects have been introduced for autonomous data gathering. From these six objects, two objects,were dropped because VoxBlox++ was unable to reconstruct these. Another three were dropped, despite a good reconstruction. VoxBlox++ was unable to separate the ground surface from the object. From only one object, the teapot, 3D reconstruction was successful. A synthetic dataset has been made from this object.

3D reconstruction

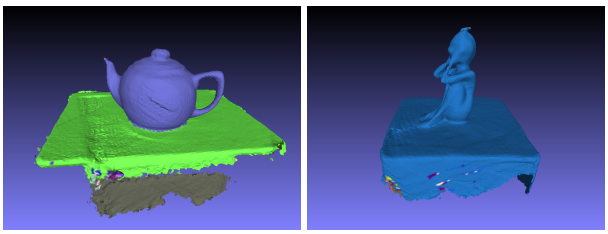
In Figure 6.6 four reconstructed objects of the original six objects of Figure 5.4 are shown. Two objects have been dropped because reconstruction failed. The first case, the aluminum teapot, failed because of the reflection of the surface. The ToF camera is unable to capture these reflective surfaces. The other object, the coffee cup failed because of form symmetry. Simple, symmetric forms are difficult to reconstruct as reconstruction algorithms are unable to provide accurate loop closure.



(a) **(b)** **(c)** **(d)**

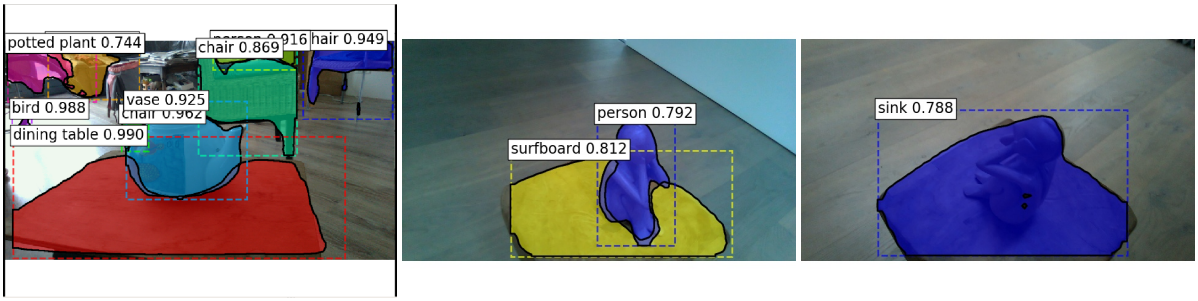
Figure 6.6: 3D models of the four objects used in for reconstruction. **(a)** The teapot is successfully segmented from the ground surface. **(b)**, **(c)** **(d)** The objects are reconstructed but separation from the ground surface has failed.

From the four objects from Figure 6.6, three objects still include the ground surface. These surface artifacts make the models unfit for the creation of synthetic datasets. More details of correct and incorrect segmentation can be found in Figures 6.7 and 6.8. Figure 6.7 shows the GSM of the teapot and the statue. Colors denote a different segment. The ground surface of the teapot has been marked as a different segment, while the statue and its ground surface are one segment. Figure 6.8 shows the semantic segmentation of the teapot and statue using Mask-RCNN during the reconstruction process. Although the predicted labels are wrong, the masks correctly covers the objects in the Figure 6.8a and 6.8b. However, Image 6.8c shows that during the reconstruction process the statue also gets masked together with the ground surface. As stated in Section 2.4.2 the final label of the segments is based on the final label count. If an object is masked more often together with its ground surface this will lead to incorrect results.



(a) **(b)**

Figure 6.7: The global segmentation map of the teapot and the statue after the reconstruction process. Different segments are marked by different colors. The segmentation of the teapot from the ground surface has been successful. The statue and its ground surface are considered one segment.

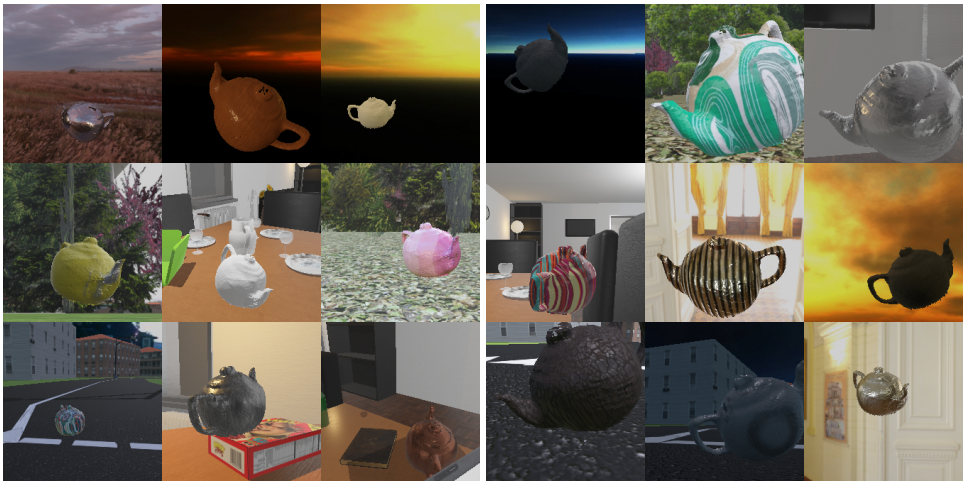


(a) **(b)** **(c)**

Figure 6.8: The semantic segmentation of Voxblox++ using Mask R-CNN during the reconstruction process. Although the semantic labels are incorrect, the created masks segment the objects correctly. **(a)** The teapot, labeled as vase is correctly masked. **(b)** A statue of a person, labeled correctly. **(c)** The same statue incorrectly labeled as a sink, the mask is also incorrect and merges the table surface and the object.

Synthetic data generation

With the 3D model of the teapot (Figure 6.6a), two synthetic dataset have been created, $\mathcal{D}^{syn}$ and $\mathcal{D}^{syn,close}$. The first dataset contains images of the teapot from a large range of camera distance, while the second set is more focused on the object. Some examples are shown in figure 6.9. Both datasets consists of 450 randomly generated images. The environment and lighting conditions were varied as well as the texture, position and orientation of the object. All images are saved with a resolution of 224×224.



(a) **(b)**

Figure 6.9: **(a)** $\mathcal{C}^{syn}$, the synthetic dataset created with webots, camera distance varies from 0.4 to 1.0 m. **(a)** $\mathcal{C}^{syn,close}$, the synthetic dataset with more close-up shots. The camera distance varies from 0.2 to 0.7 m.

6.3. On-the-job recognition

Based on the results of 6.1 two different versions of L2AIC have been selected for the on-the-job recognition task. The L2AIC-no-lstm with a pretrained EfficientNet encoder and the L2IAC-smaller-fc with pretrained ResNet152. Hyperparameters are set to $k = 10$ and $n = 9$. The models are trained on TinyImageNet and have a memory containing 70 classes. The test set consists of the 70 memory classes extended with 30 unknown classes. An additional class is added to the test set, c_{teapot}. Two different datasets are used for this new class. The first set, $\mathcal{D}^{photos,test}$ are images from the teapot instance from Section 6.2. The teapot is placed in various environments and lighting conditions. The second test set for c_{teapot} consists of test set of the teapot class of TinyImagenet, $\mathcal{D}^{Tiny,test}$. Their evaluation performance is listed in Tables 6.6 and 6.7 respectively. Both the performance on the specific new class c_{teapot} and all classes c_{all} are evaluated. Example images of $\mathcal{D}^{photos,test}$, $\mathcal{D}^{Tiny,test}$ and $\mathcal{D}^{photos,memory}$ can be found in Appendix B.

L2AIC-no-lstm

In Section 6.1 was found that the L2AIC-no-lstm with EfficientNet encoder was biased towards rejecting test samples. In Table 6.5 the ϵ_K of this model on $\mathcal{D}^{photos,test}$ are listed. The rejection bias causes the model to misclassify almost all samples from c_{teapot}, resulting in a very high ϵ_K, much larger than the average error of all classes.

The two datasets from the synthetic domain have no almost no correct classifications on $\mathcal{D}^{photos,test}$. This could be explained by the domain gap. The test dataset of $\mathcal{D}^{photos,test}$ is of the real domain. The L2AIC-no-lstm appears to be unable to match the test data with the memory of these domains. Using the dataset in the TinyImageNet domain, $\mathcal{D}^{Tiny,memory}$ gives better results. Because training of L2AIC is also done in the TinyImageNet domain the model is slightly better in overcoming the domain gap. The same holds for using a memory of the real domain, $\mathcal{D}^{photos,memory}$. Now both the memory and the input of class c_teapot are in the same domain, resulting in slightly better results. However, the error is still much larger than the average error for all classes and L2AIC appears to be unable to match the input to the memory of these domains. Full results of L2AIC-no-lstm model are listed in Appendix E.

Table 6.5: The ϵ_K performance of L2AIC-no-lstm with EfficientNet encoder on $\mathcal{D}^{photos,test}$. L2AIC has 70 classes saved in memory and the test set consists consists of an additional 30 unknown classes. The class c_{teapot} is added to the memory as different datasets, listed under **Memory addition**. Performance is both evaluated on the single teapot class as well as all test classes.

Memory addition	**Classes**	ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
None	c_{teapot}	-	-
	c_{all}	0.437	0.5
$\mathcal{D}^{Tiny,memory}$	c_{teapot}	0.79	0.98
	c_{all}	0.443	0.507
$\mathcal{D}^{photos,memory}$	c_{teapot}	0.86	0.92
	c_{all}	0.445	0.507
$\mathcal{D}^{syn}$	c_{teapot}	0.99	1.0
	c_{all}	0.445	0.507
$\mathcal{D}^{syn,close}$	c_{teapot}	1.0	1.0
	c_{all}	0.445	0.507

L2AIC-smaller-fc

In Table 6.6 the results of the L2AIC-smaller-fc with ResNet152 encoder are listed. The model is tested on $\mathcal{D}^{photos,test}$. In the case of no additional memory, $c_{teapot} \in \mathcal{C}^{\mathcal{U}}$, the model is able to reject the images of the teapot, ϵ_U is low. It performs better than the average of all unknown samples. This is the first step in the on-the-job recognition task: Rejecting unknown samples. After the second step, generating a new dataset for unknown classes, these datasets can be added to L2AIC. In the third step the performance on the new object is evaluated now with extended memory of L2AIC. The table shows that for all four different datasets the model is able to correctly classify the newly memorized object class c_{teapot}. Based on the weighted F1-score and ϵ_K the datasets $\mathcal{D}^{photos,memory}$ and $\mathcal{D}^{syn,close}$ perform best on the new object class. This can be explained by the fact that both datasets are very similar to $\mathcal{D}^{photos,test}$. They contain the same teapot instance and are close-up images of the object, making it easier to classify.

Looking at ϵ_U, the addition of $\mathcal{D}^{Tiny,memory}$ to the memory will increase the error on all unknown classes. This increase is larger compared to the other three datasets. A possible explanation for this is that $\mathcal{D}^{Tiny,memory}$ contains multiple instances of a teapot with different forms and colors. This might cause some test samples to be misclassified as a teapot. The other datasets contains images from one instance, the reconstructed teapot. Another possible explanation is that because the test set of the unknown classes are also of the TinyImageNet domain they get misclassified more easily.

Table 6.6: The performance of L2AIC-smaller-fc with ResNet152 encoder on $\mathcal{D}^{photos,test}$. L2AIC has 70 classes saved in memory and the test set consists consists of an additional 30 unknown classes. The class c_{teapot} is added to the memory as different datasets, listed under **Memory addition**. Performance is both evaluated on the single teapot class as well as all test classes.

Memory addition	**Classes**	**Weighted F1**		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
None	c_{teapot}	-	-	0.22	0.35	-	-
	c_{all}	0.699	0.632	0.378	0.458	0.261	0.331
$\mathcal{D}^{Tiny,memory}$	c_{teapot}	0.942	0.87	-	-	0.11	0.23
	c_{all}	0.69	0.625	0.396	0.482	0.259	0.331
$\mathcal{D}^{photos,memory}$	c_{teapot}	0.99	0.974	-	-	0.02	0.05
	c_{all}	0.699	0.634	0.381	0.463	0.257	0.327
$\mathcal{D}^{syn}$	c_{teapot}	0.942	0.844	-	-	0.11	0.27
	c_{all}	0.7	0.633	0.379	0.461	0.259	0.331
$\mathcal{D}^{syn,close}$	c_{teapot}	0.995	0.958	-	-	0.01	0.08
	c_{all}	0.7	0.634	0.38	0.461	0.257	0.328

In Table 6.7 the performance of L2AIC-smaller-fc on $\mathcal{D}^{Tiny,test}$ is listed. The addition of $\mathcal{D}^{Tiny,memory}$ to the memory is essentially repeating the experiments of Section 6.1. Only images of the TinyImageNet domain are used. An important difference between $\mathcal{D}^{Tiny,test}$ and $\mathcal{D}^{photos,test}$ is that the latter test set only contains one instance of a teapot. The first test set contains multiple instances making the recognition task more difficult. This is shown in Table 6.7 as the datasets consisting of only one instance of a teapot, $\mathcal{D}^{photos,memory}$, $\mathcal{D}^{syn}$ and $\mathcal{D}^{syn,close}$ have a much larger classification error on c_{teapot}. However, despite knowing only one instance of a teapot, they are able to classify a substantial part of the test set.

Table 6.7: The performance of L2AIC-smaller-fc with ResNet152 encoder on $\mathcal{D}^{Tiny,test}$. L2AIC has 70 classes saved in memory and the test set consists consists of an additional 30 unknown classes. The class c_{teapot} is added to the memory as different datasets, listed under **Memory addition**. Performance is both evaluated on the single teapot class as well as all test classes.

Memory addition	**Classes**	**Weighted F1**		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
None	c_{teapot}	-	-	0.18	0.18	-	-
	c_{all}	0.699	0.633	0.378	0.452	0.261	0.331
$\mathcal{D}^{Tiny,memory}$	c_{teapot}	0.942	0.99	-	-	0.11	0.02
	c_{all}	0.69	0.627	0.396	0.482	0.259	0.328
$\mathcal{D}^{photos,memory}$	c_{teapot}	0.561	0.507	-	-	0.61	0.66
	c_{all}	0.695	0.627	0.381	0.463	0.266	0.336
$\mathcal{D}^{syn}$	c_{teapot}	0.361	0.246	-	-	0.78	0.86
	c_{all}	0.695	0.625	0.379	0.461	0.268	0.339
$\mathcal{D}^{syn,close}$	c_{teapot}	0.561	0.45	-	-	0.61	0.71
	c_{all}	0.696	0.627	0.38	0.461	0.266	0.337

6.4. Summary

This chapter has covered the results of the experiments of this report. First, an extensive study has been done on L2AIC regarding design choices, hyperparameters and training procedure. Two designs stand out based on their overall performance, the L2AIC-no-lstm with pretrained EfficientNet encoder and the l2AIC-smaller-fc with pretrained ResNet152 encoder. The first model has better unknown rejection while the latter model is better at classifying known classes. This is in line with the theory of open set risk. A trade-off exists between the open space risk and the empirical risk. Secondly, the results of the autonomous data generation method have been reviewed. Six objects were initially selected for data generation, but for only one object, the teapot, a synthetic dataset was successfully created. Finally, the on-the-job recognition task has been conducted using the synthetic dataset of the teapot and the two best performing versions of L2AIC. L2AIC-no-lstm was unable to perform the on-the-job recognition task. However, L2AIC-smaller-fc model was able perform the task and learned to recognize a previously unknown object by adding a synthetic dataset to the memory.

7 Discussion

This chapter discusses the results from Chapter 6. First, the L2AIC is covered, followed by the autonomous data collection process. Then the on-the-job recognition task is reviewed and some suggestions for future work is given. The chapter concludes with a small summary of the findings.

Learning to Accept Image Classes
Following the results of Section 6.1 two designs of the L2AIC algorithm have the best overall performance. Comparison to the L2AC algorithm shows that a small improvement on the overall performance is achieved. The first model, L2AIC-no-lstm with pretrained EfficientNet encoder, has a bias towards rejecting unknown classes. The second model, L2AIC-smaller-fc with pretrained ResNet152 encoder, has a bias towards correct classification of classes. Open world performance of the classifier has shown to be a trade-off between two key characteristics of open world learning, the ability to reject unknown classes and the ability to recognize the classes that are known. An algorithm more eager to reject unknowns will also reject the more difficult samples from known object classes. On the other hand, an algorithm that is more eager to classify its known classes also easily classifies a similar but unknown object as known. This is in line with the theory of open set risk of Section 2.1.
The choice of encoder is also very important for the performance of L2AIC. It is found that encoders loaded with pretrained weights on ImageNet perform better than using a pretrained encoder that has undergone fine-tuning on the used dataset. This fine-tuning is done on a separate set of classes and might cause the encoder to overfit on this particular set of classes and their features.
Because TinyImageNet is a subset of ImageNet, the pretrained encoder is biased towards this dataset. A performance comparison with CIFAR-100 shows that L2AIC is better at rejecting unknowns from the TinyImageNet dataset, but performs better at classifying known classes at the CIFAR-100 dataset. This shows that the L2AIC is at least partially able to counter this bias.
The optimal set of hyperparameters for the ranker have been found to be $k = 10$ and $n = 9$. Lower hyperparameters would favor better unknown rejection but at the cost of higher known classification error. In a similar fashion, increasing the hyperparameters improves classification of known classes but at the cost of a higher unknown classification error.
Changing training procedures of the L2AIC algorithm did not improve results. Variations in the ranking procedure and training in multiple steps gave a small decrease of known classification error. However, this comes at the cost of worse class rejection decreasing the overall open world performance of the algorithm.

Autonomous data collection
During the process of autonomous data collection it became clear that 3D reconstruction is still a difficult problem to solve. Only one out of six object was successfully reconstructed and separated from the ground surface. With the reconstructed object a synthetic dataset is created. This is done in autonomous fashion using the robotics simulator Webots. Randomized images are created from the object in different environments and lighting conditions with varying textures, positions and orientations.

To my knowledge, this is the first synthetic dataset that uses 3D models of real objects in a 3D environment. By creating an autonomous process the labour-expensive and time-consuming process of dataset creation can be simplified.
The autonomous data collection process has some limitations. Foremost, the process has not been fully automated in this report. Because the use of robotics was outside of the scope of this project the 3D camera has been handled by a human actor instead of a robot. Furthermore, Webots cannot import the original texture of the 3D model. Object casting shadows is not possible for 3D models that contain too many vertices, which is often the case with 3D scanned objects.

On-the-job recognition
The on-the-job recognition task, learning to recognize a teapot, has been performed with the L2AIC-no-lstm model with pretrained EfficientNet and L2AIC-smaller-fc model with ResNet152. The l2AIC-no-lstm failed the task. It was unable to correctly recognize samples from the teapot due to its bias towards class rejection. Almost all teapot images were misclassified as unknown. This could also have been made worse due to the domain gap between the real and synthetic domain.
The L2AIC-smaller-fc did succeed at the on-the-job recognition task. Adding synthetic data of the teapot to its memory enabled the model to correctly classify test images of the teapot. Almost all test images were classified correctly. This might be an indication that the model takes more into account than just the object features. Because the model is a meta-classifier it uses comparison of features to decide similarity between an image and object classes. The almost perfect classification might be an indication the meta-classifier compares domain specific features as well.
Two synthetic datasets have been created, varying the camera distance to the object. Results show that the synthetic dataset consisting of more close-up images of the object achieves better performance. Besides being easier to classify, the fact that the test set also consists of close-up images of the object might play a role in this.
Beside synthetic datasets the performance of L2AIC has also been evaluated with a dataset consisting of images from the reconstruction process. Performance has shown to be almost identical to synthetic datasets. This suggests that only 2D images of a camera are enough for successful classification, making the extensive reconstruction process redundant. However, as mentioned earlier, the synthetic dataset does not contain the original texture of the object. This proves that using a synthetic dataset of a single object can enable recognition of the different colored objects of the same class. This hypothesis is strengthened by classifying image samples from the teapot class of TinyImageNet. This set of images contain many instances of teapots with different shapes and colors. The L2AIC-smaller-fc can correctly classify a part of these images using with the help of a memory containing just a single instance of the object. Again, the 2D teapot images and synthetic dataset show similar results, but the synthetic dataset has still a lot of room for improvement, leaving the possibility for better performance in the future.

Future research
This report contains some interesting findings, but also some limitations. A list of proposals for future research is given that could lead to better understanding and results with regards to open world recognition, autonomous data collection and on-the-job recognition.

- To decrease the effect of the domain gap between the synthetic and real domain the L2AIC model could be trained with a memory consisting of only synthetic images, while giving real images as input. This way the model can adapt the synthetic domain to the real domain.
- To test the true open world capability of L2AIC it is proposed to train on the full ImageNet dataset. This would allow for testing on a much larger set of classes and is a more close simulation of real world classification task.
- The robotics simulator Webots had limitation such as casting shadows or loading original texture. These are software specific limitations and could be overcome by using different software.
- A synthetic dataset could also be created using a combination of 3D models and Generative Adversarial Networks (GANs). This could improve the domain randomization of the dataset.
- The autonomous data collection process was only partially autonomous as the camera was still handled by a human actor. Future work could completely automate the process using a robot.
- This report only focused on open world recognition. Expanding open world learning to object detection, *open world detection*, is one of the next steps to be taken.

Summary

Based on their performance the L2AIC-no-lstm with EfficientNet encoder and L2AIC-smaller-fc with ResNet152 proved to have the best overall open world performance. They still have their limitations as they are either biased towards unknown class rejection or known class classification. This is in line with the theory of open space risk. Results show that they outperform the original open-wold classifier L2AC. The autonomous data collection was successful for one of the six selected objects. Two datasets have been created with varying distance of the camera from the object. To my knowledge, this is the first study of creating synthetic datasets consisting of 3D reconstructed objects in 3D environments. The ability of automating the process can replace the traditional labour-expensive and time-consuming dataset creation process. From the two best performing L2AIC models, only L2AIC-smaller-fc succeeded in the on-the-job recognition task. The model has been tested by using two synthetic datasets with varying camera distances stored in memory. Results show that close-up images of the object give better performance. Beside the synthetic dataset a dataset of a 2D camera is during testing. A memory with these camera images gave a similar performance as with the synthetic set. Beside a single instance test set, the model is also tested on the multi-instance test set of TinyImageNet. Results show that a memory, consisting of only one instance from a class, is sufficient to correctly classify a substantial part of the multi-instance test set.

References

[1] Wickliffe C. Abraham and Anthony Robins. "Memory retention – the synaptic stability versus plasticity dilemma". In: *Trends in Neurosciences* 28.2 (Feb. 2005), pp. 73–78. ISSN: 01662236. DOI: 10.1016/j.tins.2004.12.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166223604003704 (visited on 10/04/2021).

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-94462-3 978-3-319-94463-0. DOI: 10.1007/978-3-319-94463-0. URL: http://link.springer.com/10.1007/978-3-319-94463-0 (visited on 08/05/2021).

[3] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *arXiv:1606.04474 [cs]* (Nov. 30, 2016). arXiv: 1606.04474. URL: http://arxiv.org/abs/1606.04474 (visited on 10/04/2021).

[4] Andrei Barbu et al. "ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models". In: (2019), p. 11.

[5] Abhijit Bendale and Terrance Boult. "Towards Open Set Deep Networks". In: *arXiv:1511.06233 [cs]* (Nov. 19, 2015). arXiv: 1511.06233. URL: http://arxiv.org/abs/1511.06233 (visited on 03/26/2021).

[6] Abhijit Bendale and Terrance Boult. "Towards Open World Recognition". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, MA, USA: IEEE, June 2015, pp. 1893–1902. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298799. URL: http://ieeexplore.ieee.org/document/7298799/ (visited on 01/05/2021).

[7] P.J. Besl and Neil D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/34.121791. URL: http://ieeexplore.ieee.org/document/121791/ (visited on 10/27/2020).

[8] Boston Dynamics. *Atlas*. Boston Dynamics. 2021. URL: https://www.bostondynamics.com/atlas (visited on 09/04/2021).

[9] T. E. Boult et al. "Learning and the Unknown: Surveying Steps toward Open World Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 17, 2019), pp. 9801–9807. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33019801. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5054 (visited on 03/30/2021).

[10] Zhiyuan Chen and Bing Liu. "Lifelong Machine Learning, Second Edition". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12.3 (Aug. 14, 2018), pp. 1–207. ISSN: 1939-4608, 1939-4616. DOI: 10.2200/S00832ED1V01Y201802AIM037. URL: https://www.morganclaypool.com/doi/10.2200/S00832ED1V01Y201802AIM037 (visited on 08/16/2021).

[11] Ta-Ying Cheng. *Understanding Real Time 3D Reconstruction and KinectFusion*. Medium. Mar. 7, 2020. URL: https://itnext.io/understanding-real-time-3d-reconstruction-and-kinectfusion-33d61d1cd402 (visited on 12/27/2020).

[12] Francois Chollet. *Deep learning with Python*. OCLC: ocn982650571. Shelter Island, New York: Manning Publications Co, 2018. 361 pp. ISBN: 978-1-61729-443-3.

[13] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*. the 23rd annual conference. Not Known: ACM Press, 1996, pp. 303–312. ISBN: 978-0-89791-746-9. DOI: 10.1145/237170.237269. URL: http://portal.acm.org/citation.cfm?doid=237170.237269 (visited on 10/29/2020).

[14] Rocco De Rosa, Thomas Mensink, and Barbara Caputo. “Online Open World Recognition”. In: *arXiv:1604.02275 [cs, stat]* (Apr. 8, 2016). version: 1. arXiv: 1604.02275. URL: http://arxiv.org/abs/1604.02275 (visited on 04/06/2021).

[15] Akshay Raj Dhamija et al. “The Overlooked Elephant of Object Detection: Open Set”. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2020 IEEE Winter Conference on Applications of Computer Vision (WACV). Snowmass Village, CO, USA: IEEE, Mar. 2020, pp. 1010–1019. ISBN: 978-1-72816-553-0. DOI: 10.1109/WACV45572.2020.9093355. URL: https://ieeexplore.ieee.org/document/9093355/ (visited on 05/11/2021).

[16] Marc-Antoine Drouin and Lama Seoud. “Consumer-Grade RGB-D Cameras”. In: *3D Imaging, Analysis and Applications*. Ed. by Yonghuai Liu et al. Cham: Springer International Publishing, 2020, pp. 215–264. ISBN: 978-3-030-44070-1. DOI: 10.1007/978-3-030-44070-1_5. URL: https://doi.org/10.1007/978-3-030-44070-1_5 (visited on 03/19/2021).

[17] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. “Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection”. In: *arXiv:1708.01642 [cs]* (Aug. 4, 2017). arXiv: 1708.01642. URL: http://arxiv.org/abs/1708.01642 (visited on 10/15/2020).

[18] Geli Fei, Shuai Wang, and Bing Liu. “Learning Cumulatively to Become More Knowledgeable”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, Aug. 13, 2016, pp. 1565–1574. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939835. URL: https://dl.acm.org/doi/10.1145/2939672.2939835 (visited on 08/17/2021).

[19] Fadri Furrer et al. “Incremental Object Database: Building 3D Models from Multiple Partial Observations”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid: IEEE, Oct. 2018, pp. 6835–6842. ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8594391. URL: https://ieeexplore.ieee.org/document/8594391/ (visited on 02/23/2021).

[20] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. “Recent Advances in Open Set Recognition: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.2981604. arXiv: 1811.08581. URL: http://arxiv.org/abs/1811.08581 (visited on 03/26/2021).

[21] Georgios Georgakis et al. “Synthesizing Training Data for Object Detection in Indoor Scenes”. In: (2017), p. 9.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. 775 pp. ISBN: 978-0-262-03561-3.

[23] Margarita Grinvald et al. “Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery”. In: *IEEE Robotics and Automation Letters* 4.3 (July 2019), pp. 3037–3044. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2019.2923960. arXiv: 1903.00268. URL: http://arxiv.org/abs/1903.00268 (visited on 02/23/2021).

[24] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[25] Kaiming He et al. “Mask R-CNN”. In: (2017).

[26] Stefan Hinterstoisser et al. “An Annotation Saved is an Annotation Earned: Using Fully Synthetic Training for Object Detection”. In: (2019), p. 10.

[27] Jie Hu et al. “Squeeze-and-Excitation Networks”. In: *arXiv:1709.01507 [cs]* (May 16, 2019). arXiv: 1709.01507. URL: http://arxiv.org/abs/1709.01507 (visited on 09/09/2021).

[28] Intel. *LiDAR – Intel® RealSense™ Technology*. Intel® RealSense™ Depth and Tracking Cameras. 2021. URL: https://www.intelrealsense.com/lidar/ (visited on 03/25/2021).

[29] Philip T Jackson et al. “Style Augmentation: Data Augmentation via Style Randomization”. In: (2018), p. 10.

[30] K. J. Joseph et al. “Towards Open World Object Detection”. In: *arXiv:2103.02603 [cs]* (May 9, 2021). arXiv: 2103.02603. URL: http://arxiv.org/abs/2103.02603 (visited on 05/11/2021).

[31] Alireza Khatemian and Hamid Arabnia. “Survey on 3D Surface Reconstruction”. In: (2016).

[32] Simon Kriegel. “Autonomous 3D Modeling of Unknown Objects for Active Scene Exploration”. In: (2015), p. 175.

[33] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: (2009), p. 60.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 08/13/2021).

[35] Mathieu Labbé and François Michaud. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation”. In: *Journal of Field Robotics* 36.2 (Mar. 2019), pp. 416–446. ISSN: 15564959. DOI: 10.1002/rob.21831. URL: http://doi.wiley.com/10.1002/rob.21831 (visited on 11/02/2020).

[36] Ya Le and Xuan Yang. “Tiny ImageNet Visual Recognition Challenge”. In: (2015), p. 6.

[37] Melanie Lefkowitz. *Professor’s perceptron paved the way for AI – 60 years too soon*. Cornell Chronicle. 2019. URL: https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon (visited on 10/06/2021).

[38] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Vol. 8693. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10601-4 978-3-319-10602-1. DOI: 10.1007/978-3-319-10602-1_48. URL: http://link.springer.com/10.1007/978-3-319-10602-1_48 (visited on 08/24/2021).

[39] Bing Liu. “Learning on the Job: Online Lifelong and Continual Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.9 (Apr. 3, 2020), pp. 13544–13549. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i09.7079. URL: https://aaai.org/ojs/index.php/AAAI/article/view/7079 (visited on 05/11/2021).

[40] M. Mancini et al. “Knowledge is Never Enough: Towards Web Aided Deep Open World Recognition”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019 International Conference on Robotics and Automation (ICRA). ISSN: 2577-087X. May 2019, pp. 9537–9543. DOI: 10.1109/ICRA.2019.8793803.

[41] Osama Mazhar and Jens Kober. “Random Shadows and Highlights: A new data augmentation method for extreme lighting conditions”. In: *arXiv:2101.05361 [cs]* (Jan. 18, 2021). arXiv: 2101.05361. URL: http://arxiv.org/abs/2101.05361 (visited on 08/19/2021).

[42] Michael McCloskey and Neal J. Cohen. “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem”. In: *Psychology of Learning and Motivation*. Vol. 24. Elsevier, 1989, pp. 109–165. ISBN: 978-0-12-543324-2. DOI: 10.1016/S0079-7421(08)60536-8. URL: https://linkinghub.elsevier.com/retrieve/pii/S0079742108605368 (visited on 04/08/2021).

[43] Warren S Mcculloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: (1943), p. 17.

[44] Olivier Michel. “Webots™: Professional Mobile Robot Simulation”. In: *International Journal of Advanced Robotic Systems* 1.1 (Mar. 1, 2004). Publisher: SAGE Publications, p. 5. ISSN: 1729-8814. DOI: 10.5772/5618. URL: https://doi.org/10.5772/5618 (visited on 08/31/2021).

[45] Microsoft. *Kinect - Windows app development*. 2010. URL: https://developer.microsoft.com/en-us/windows/kinect/ (visited on 03/25/2021).

[46] Richard A Newcombe et al. “KinectFusion: Real-Time Dense Surface Mapping and Tracking”. In: (2011), p. 10.

[47] Helen Oleynikova et al. “Voxblox: Building 3D Signed Distance Fields for Planning”. In: (2016). Artwork Size: 8 p. Medium: application/pdf Publisher: ETH Zurich, 8 p. DOI: 10.3929/ETHZ-A-010820353. URL: http://hdl.handle.net/20.500.11850/128028 (visited on 03/10/2021).

[48] Helen Oleynikova et al. “Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vancouver, BC: IEEE, Sept. 2017, pp. 1366–1373. ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8202315. URL: http://ieeexplore.ieee.org/document/8202315/ (visited on 02/23/2021).

[49] PAL Robotics. *PAL Robotics : Leading company in service robotics*. PAL Robotics. 2021. URL: https://pal-robotics.com/ (visited on 08/31/2021).

[50] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. First edition. OCLC: ocn902657832. Sebastopol, CA: O'Reilly, 2017. 507 pp. ISBN: 978-1-4919-1425-0.

[51] Xingchao Peng et al. “Syn2Real: A New Benchmark forSynthetic-to-Real Visual Domain Adaptation”. In: *arXiv:1806.09755 [cs]* (June 25, 2018). arXiv: 1806.09755. URL: http://arxiv.org/abs/1806.09755 (visited on 12/15/2020).

[52] Hadi Pouransari and Saman Ghili. “Tiny ImageNet Visual Recognition Challenge”. In: (2015), p. 9.

[53] Sudharsan Ravichandiran. *Hands-on meta learning with Python: meta learning using one-shot learning, MAML, Reptile, and Meta-SGD with TensorFlow*. OCLC: 1107492001. Birmingham, UK: Packt Publishing, 2018. ISBN: 978-1-78953-702-4. URL: http://proquestcombo.safaribooksonline.com/9781789534207 (visited on 09/09/2021).

[54] Sylvestre-Alvise Rebuffi et al. “iCaRL: Incremental Classifier and Representation Learning”. In: *arXiv:1611.07725 [cs, stat]* (Apr. 14, 2017). arXiv: 1611.07725. URL: http://arxiv.org/abs/1611.07725 (visited on 04/06/2021).

[55] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519 (visited on 08/09/2021).

[56] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0 (visited on 08/10/2021).

[57] Donald Rumsfeld. *Known and Unknown: A Memoir*. 1st edition. Sentinel, Feb. 8, 2011. 883 pp.

[58] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (Dec. 1, 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y (visited on 08/23/2021).

[59] W. J. Scheirer, L. P. Jain, and T. E. Boult. “Probability Models for Open Set Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (Nov. 2014). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 2317–2324. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2014.2321392.

[60] Walter J Scheirer, Archana Sapkota, and Terrance E Boult. “Towards Open Set Recognition”. In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (2012), p. 17.

[61] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: (2014), p. 30.

[62] Chen Sun et al. “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”. In: *arXiv:1707.02968 [cs]* (Aug. 3, 2017). arXiv: 1707.02968. URL: http://arxiv.org/abs/1707.02968 (visited on 08/19/2021).

[63] Niko Sünderhauf et al. “The Limits and Potentials of Deep Learning for Robotics”. In: *arXiv:1804.06557 [cs]* (Apr. 18, 2018). arXiv: 1804.06557. URL: http://arxiv.org/abs/1804.06557 (visited on 01/05/2021).

[64] Mingxing Tan and Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *arXiv:1905.11946 [cs, stat]* (Sept. 11, 2020). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946 (visited on 08/13/2021).

[65] Josh Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *arXiv:1703.06907 [cs]* (Mar. 20, 2017). arXiv: 1703.06907. URL: http://arxiv.org/abs/1703.06907 (visited on 11/09/2020).

[66] Antonio Torralba and Alexei A. Efros. “Unbiased look at dataset bias”. In: *CVPR 2011*. 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Colorado Springs, CO, USA: IEEE, June 2011, pp. 1521–1528. ISBN: 978-1-4577-0394-2. DOI: 10.1109/CVPR.2011.5995347. URL: http://ieeexplore.ieee.org/document/5995347/ (visited on 01/05/2021).

[67] Hu Xu et al. “Open-world Learning and Application to Product Classification”. In: *arXiv:1809.06004 [cs]* (Mar. 1, 2019). DOI: 10.1145/3308558.3313644. arXiv: 1809.06004. URL: http://arxiv.org/abs/1809.06004 (visited on 04/12/2021).

A

L2AIC architectures

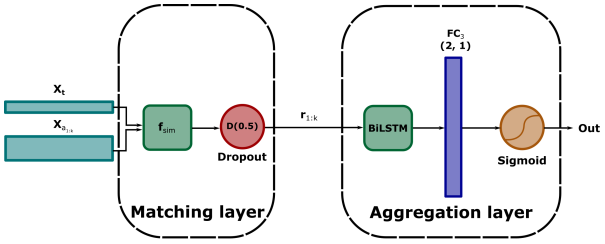


Figure A.1: L2AIC-cosine: The matching layer has been replaced with the cosine similarity function (Equation 3.1). This effectively removes the whole matching layer and the aggregation layer will predict a final score based on output values of the ranker. This variation puts the influence of the matching layer to the test.

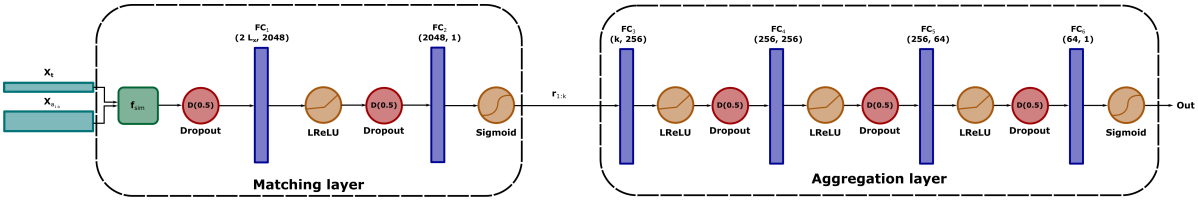


Figure A.2: L2AIC-no-lstm: The LSTM in the aggregation layer is replaced with multiple fully connected layers. Besides aggregation the LSTM also functions as optimizer for the matching layer (Section 2.3). This variation tests the influence of the LSTM on performance of the L2AIC model.

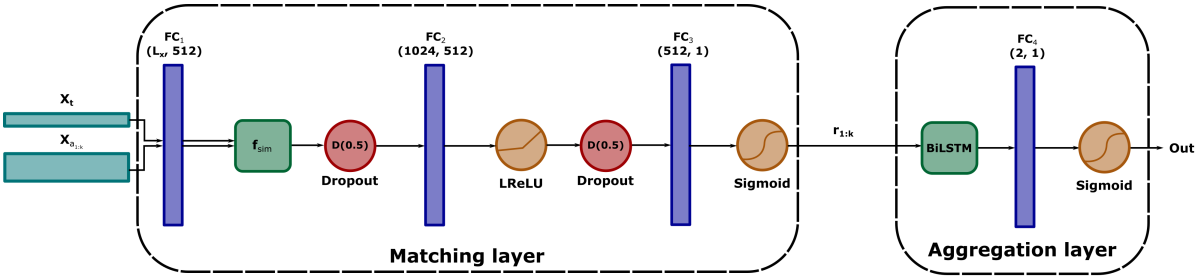


Figure A.3: L2AIC-smaller-fc: Both feature vectors are reduced in size by a fully connected layer before being passed on to the similarity function. This fully connected layer might help in the selection of most relevant features before being translated to similarity space.

]

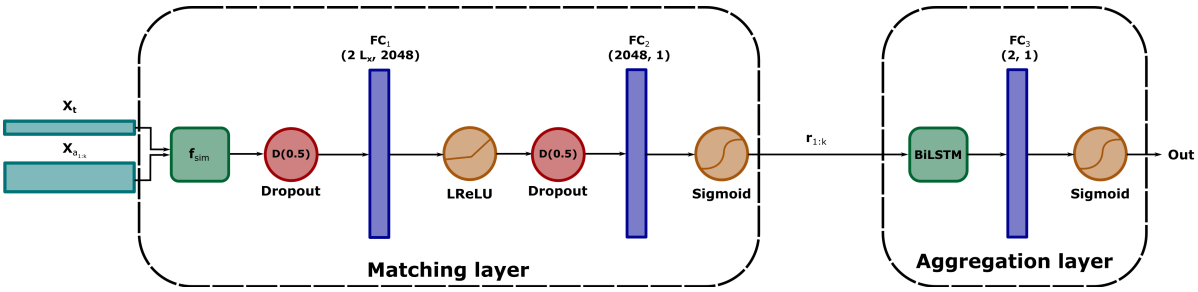


Figure A.4: L2AIC-abssub: The similarity function (Equation 2.2) has been reduced to only the first part of the concatenation, the absolute subtraction of both vectors $f_{sim} = |x_t - x_{a_i}|$.

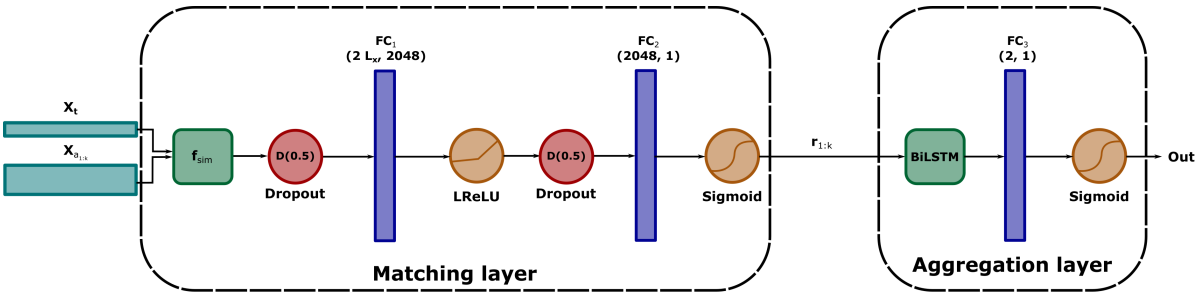


Figure A.5: L2AIC-concat: The similarity function is removed with the idea that the meta-classifier learns a similarity function on its own . Both feature vectors are concatenated and passed through the fully connected layers $f_{sim} = x_t \oplus x_{a_i}$.

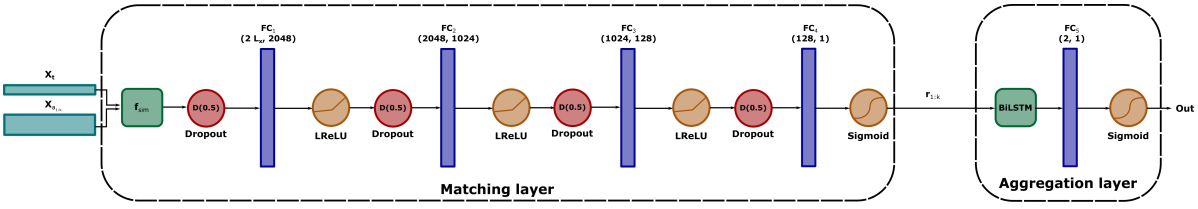


Figure A.6: L2AIC-extended-similarity:A copy of L2AIC-concat, where the similarity function is removed ($f_{sim} = x_t \oplus x_{a_i}$), but now the matching layer has been extended with multiple fully connected layers. This give the matching layer more capacity for finding a similarity function.

B

Extra datasets

Figure B.1: $\mathcal{D}^{photos,memory}$: The dataset created from snapshots during the reconstruction process. This dataset is used as memory for the L2AIC algorithm.

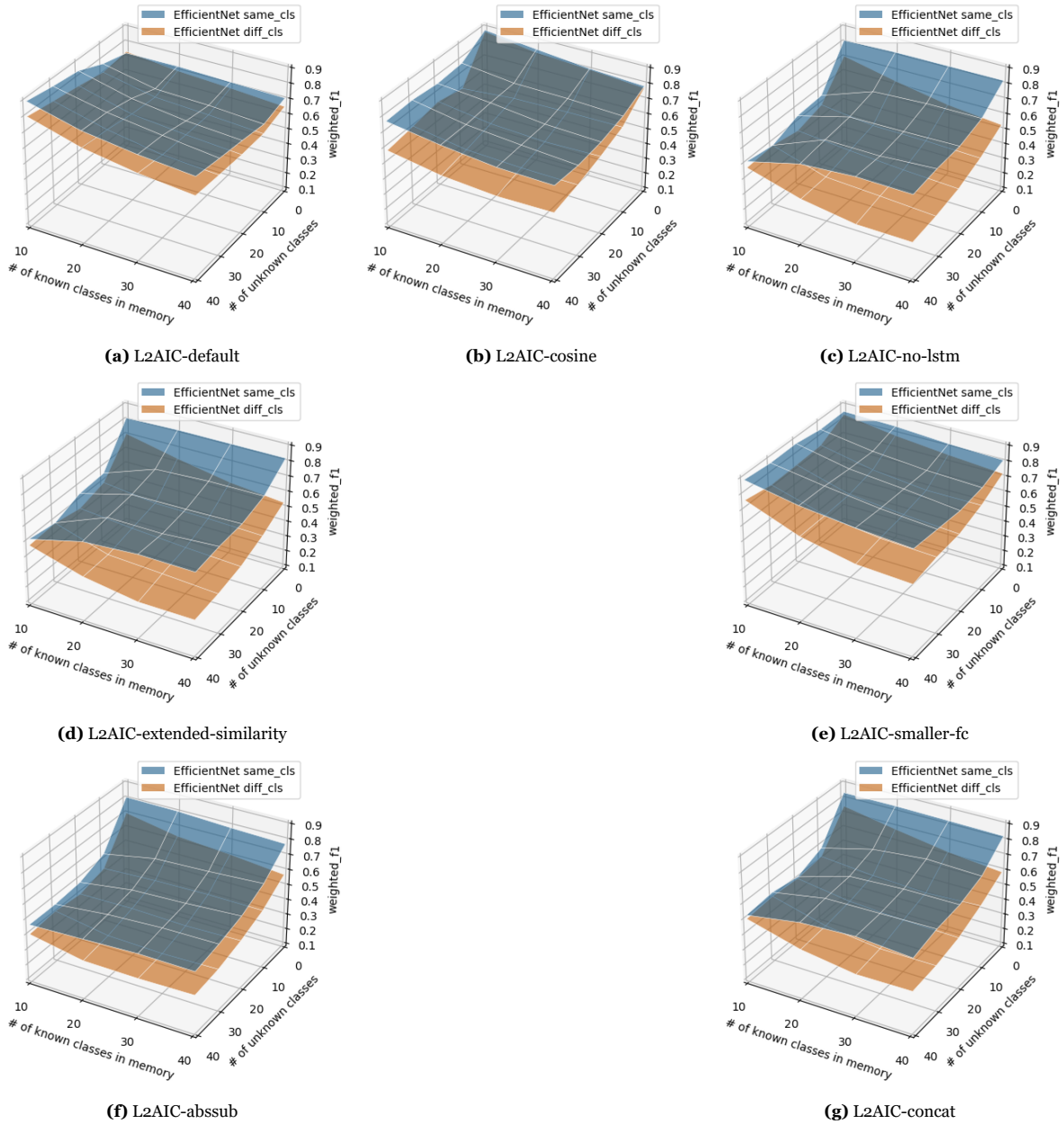
Figure B.2: $\mathcal{D}^{photos,test}$: The dataset created for testing the performance of the L2AIC algorithm.

Figure B.3: $\mathcal{D}^{Tiny,test}$: The teapot class of the TinyImageNet dataset. This dataset is used for testing the performance of the L2AIC algorithm.

C

Meta-classifier extra results

C.1. Weighted F1-score for architectures



(a) L2AIC-default

(b) L2AIC-cosine

(c) L2AIC-no-lstm

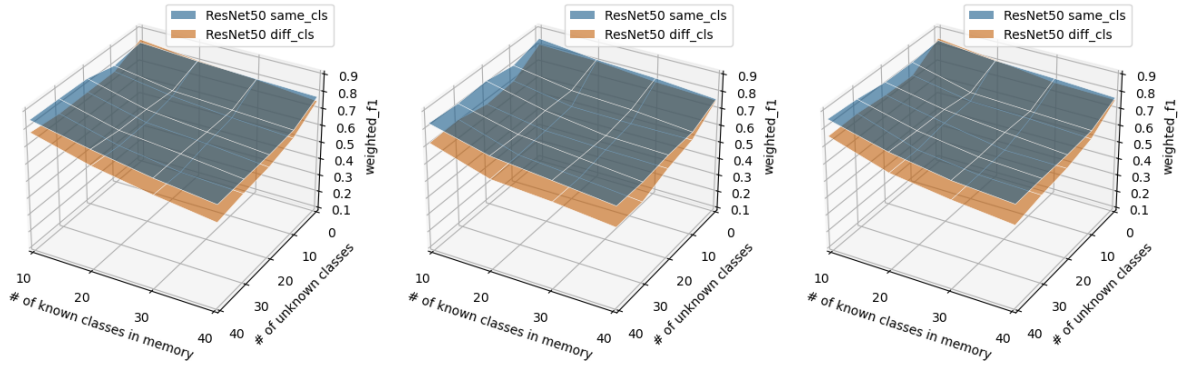
(d) L2AIC-extended-similarity

(e) L2AIC-smaller-fc

(f) L2AIC-abssub

(g) L2AIC-concat

Figure C.1: Weighted F1-score for L2AIC with EfficientNet per architecture

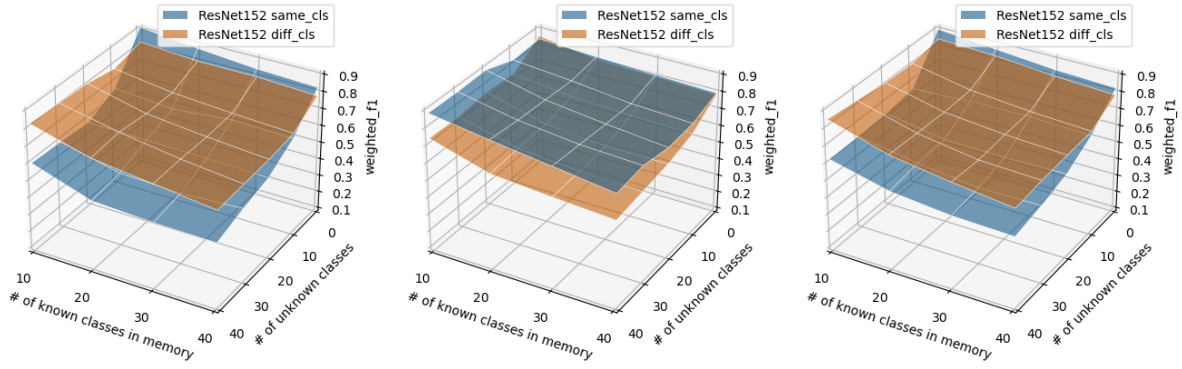


(a) L2AIC-default **(b)** L2AIC-cosine **(c)** L2AIC-no-lstm

(d) L2AIC-extended-similarity **(e)** L2AIC-smaller-fc

(f) L2AIC-abssub **(g)** L2AIC-concat

Figure C.2: Weighted F1-score for L2AIC with ResNet50 per architecture



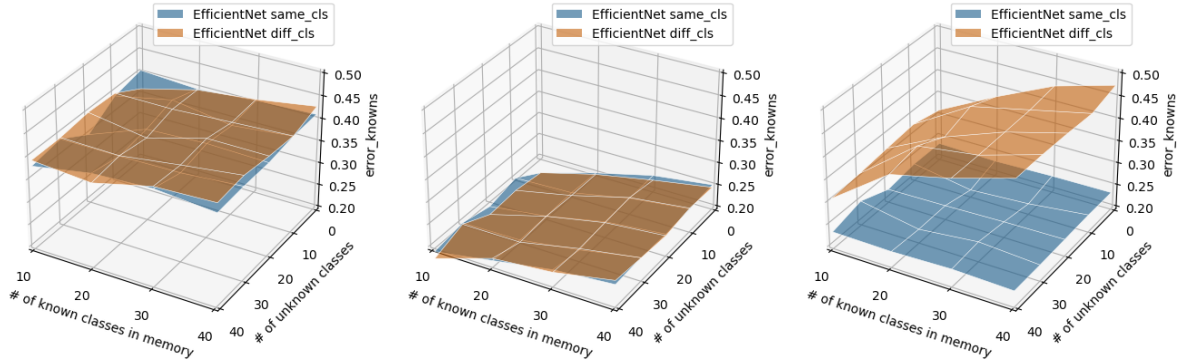
(a) L2AIC-default **(b)** L2AIC-cosine **(c)** L2AIC-no-lstm

(d) L2AIC-extended-similarity **(e)** L2AIC-smaller-fc

(f) L2AIC-abssub **(g)** L2AIC-concat

Figure C.3: Weighted F1-score for L2AIC with ResNet152 per architecture

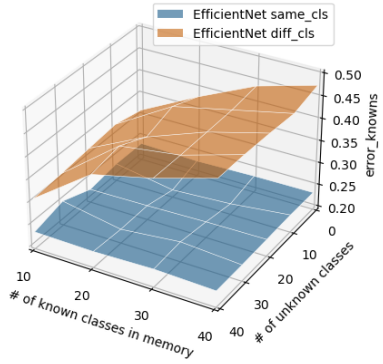
C.2. known error score per architecture



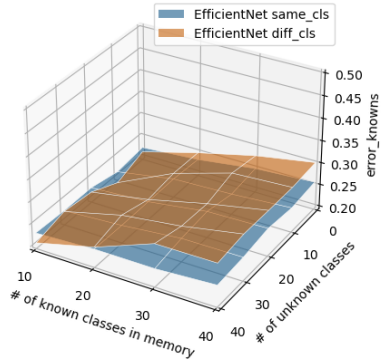
(a) L2AIC-default

(b) L2AIC-cosine

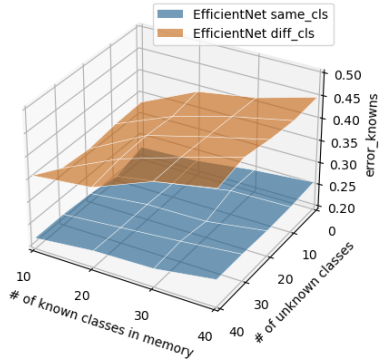
(c) L2AIC-no-lstm



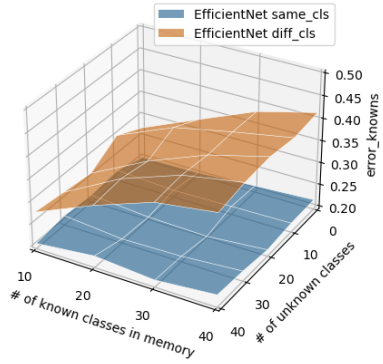
(d) L2AIC-extended-similarity



(e) L2AIC-smaller-fc

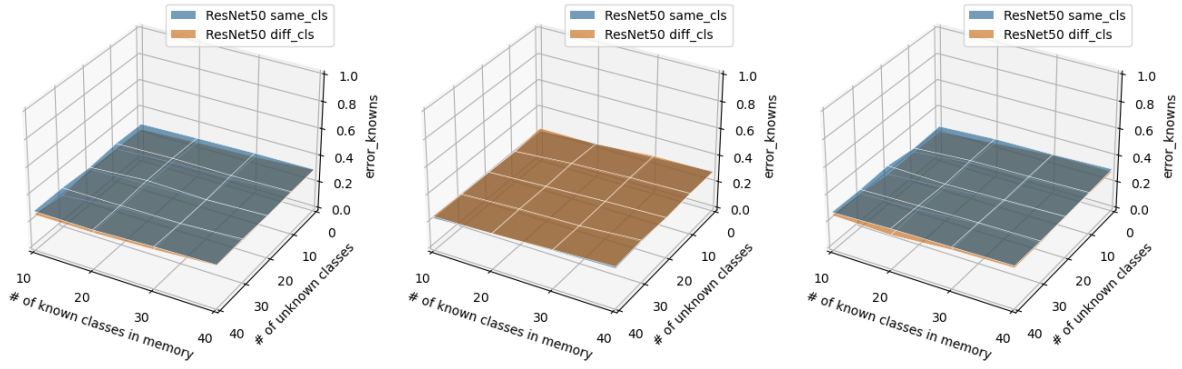


(f) L2AIC-abssub



(g) L2AIC-concat

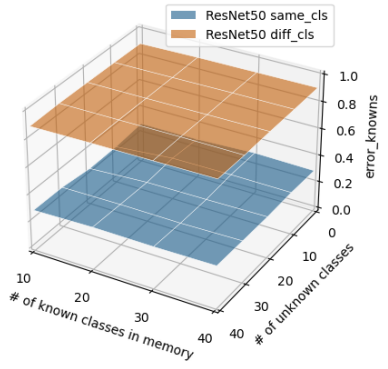
Figure C.4: Known error for L2AIC with EfficientNet per architecture



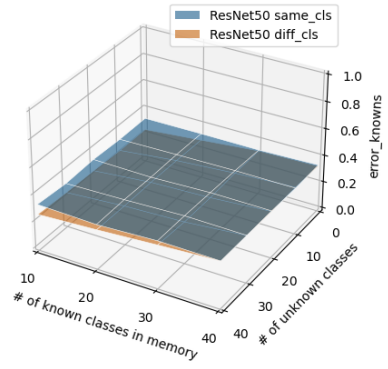
(a) L2AIC-default

(b) L2AIC-cosine

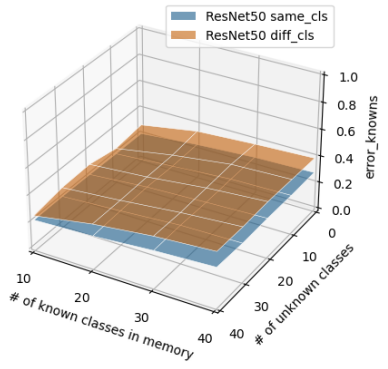
(c) L2AIC-no-lstm



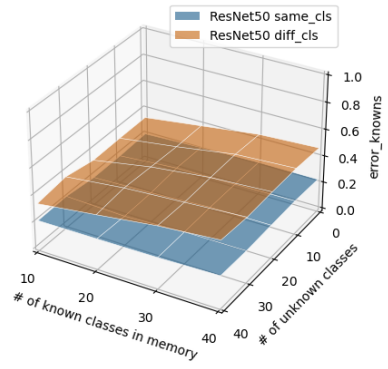
(d) L2AIC-extended-similarity



(e) L2AIC-smaller-fc

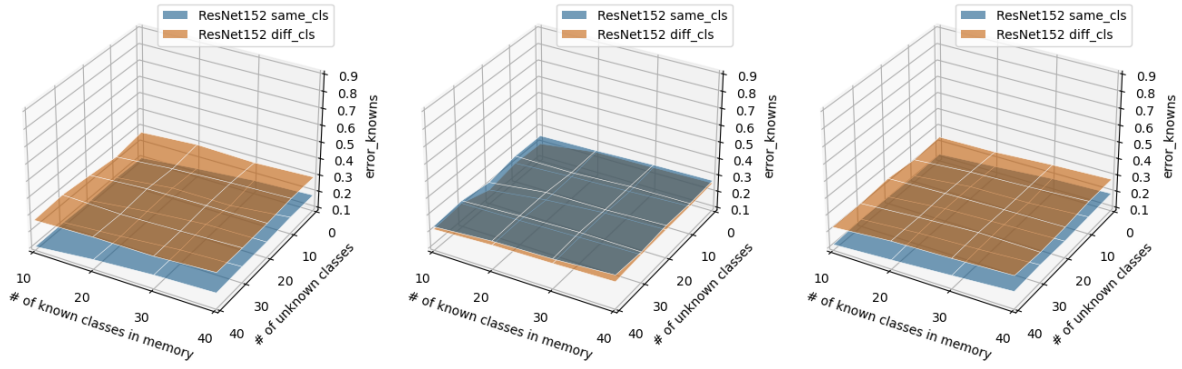


(f) L2AIC-abssub



(g) L2AIC-concat

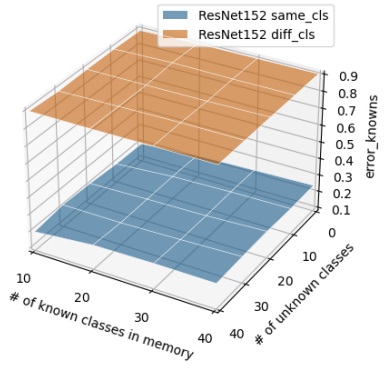
Figure C.5: Known error for L2AIC with ResNet50 per architecture



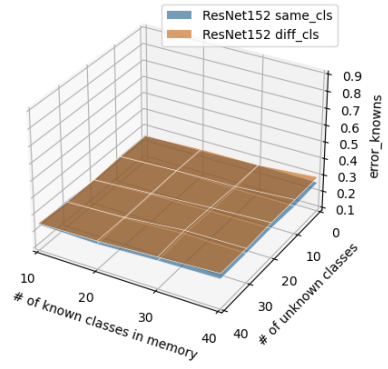
(a) L2AIC-default

(b) L2AIC-cosine

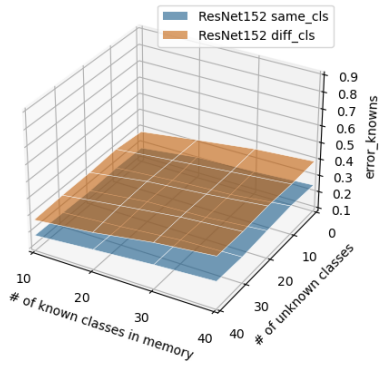
(c) L2AIC-no-lstm



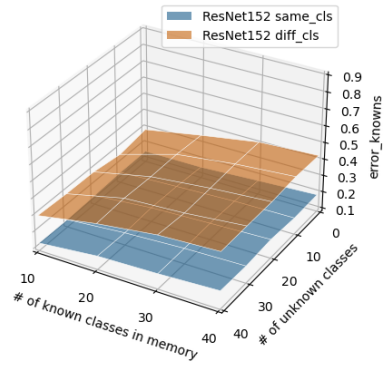
(d) L2AIC-extended-similarity



(e) L2AIC-smaller-fc



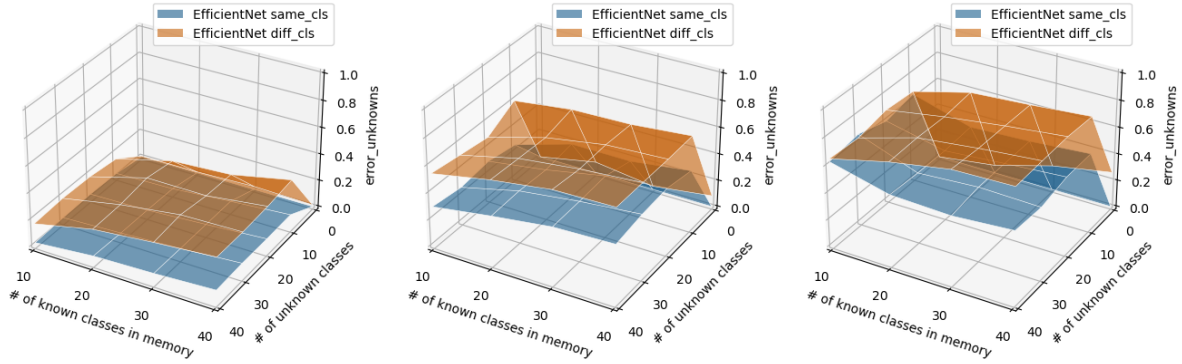
(f) L2AIC-abssub



(g) L2AIC-concat

Figure C.6: Known error for L2AIC with ResNet152 per architecture

C.3. Unknown error per architecture

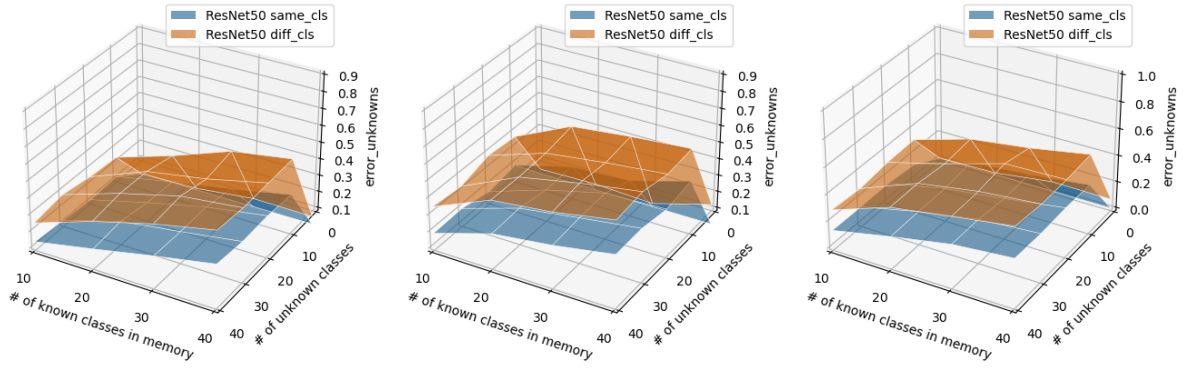


(a) L2AIC-default **(b)** L2AIC-cosine **(c)** L2AIC-no-lstm

(d) L2AIC-extended-similarity **(e)** L2AIC-smaller-fc

(f) L2AIC-abssub **(g)** L2AIC-concat

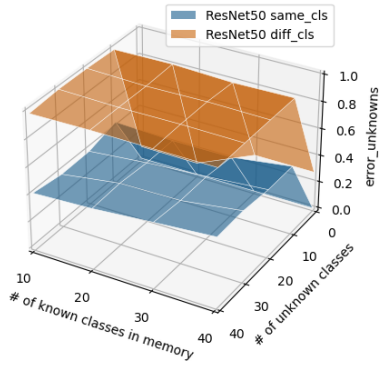
Figure C.7: Unknown error for L2AIC with EfficientNet per architecture



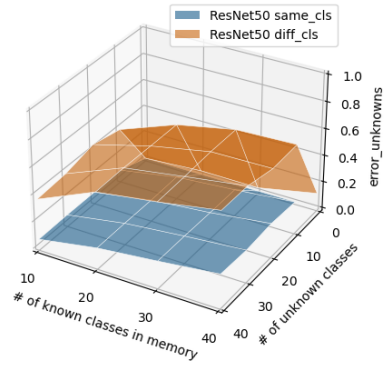
(a) L2AIC-default

(b) L2AIC-cosine

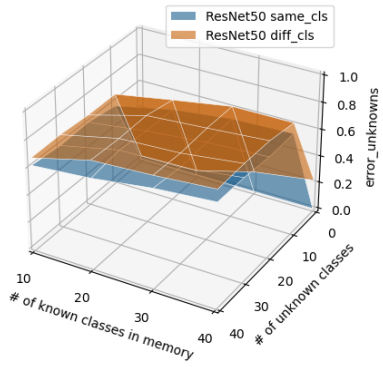
(c) L2AIC-no-lstm



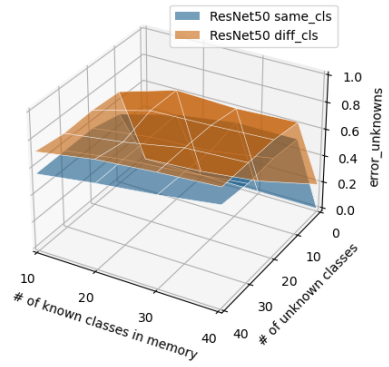
(d) L2AIC-extended-similarity



(e) L2AIC-smaller-fc

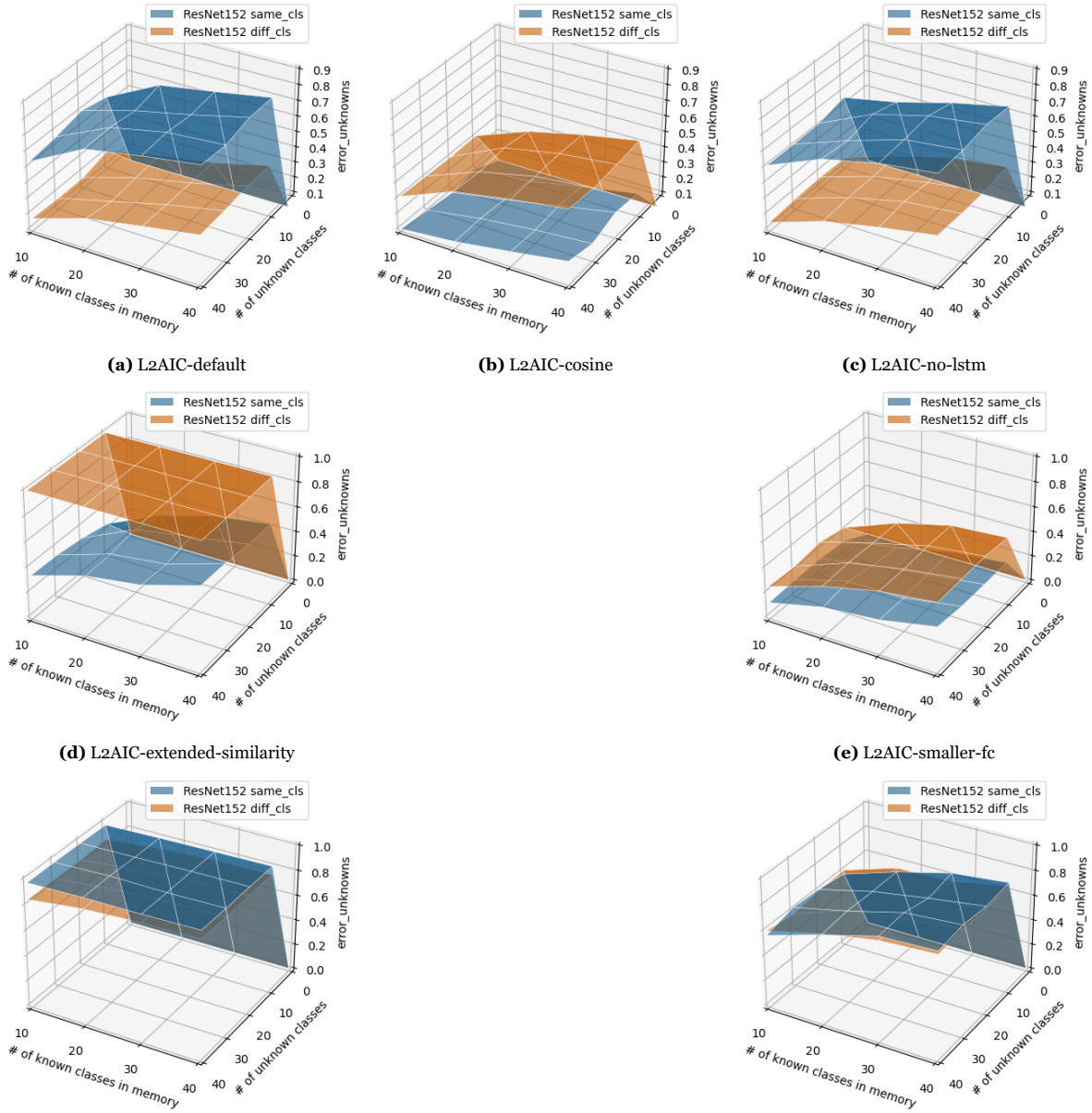


(f) L2AIC-abssub



(g) L2AIC-concat

Figure C.8: Unknown error for L2AIC with ResNet50 per architecture



(a) L2AIC-default

(b) L2AIC-cosine

(c) L2AIC-no-lstm

(d) L2AIC-extended-similarity

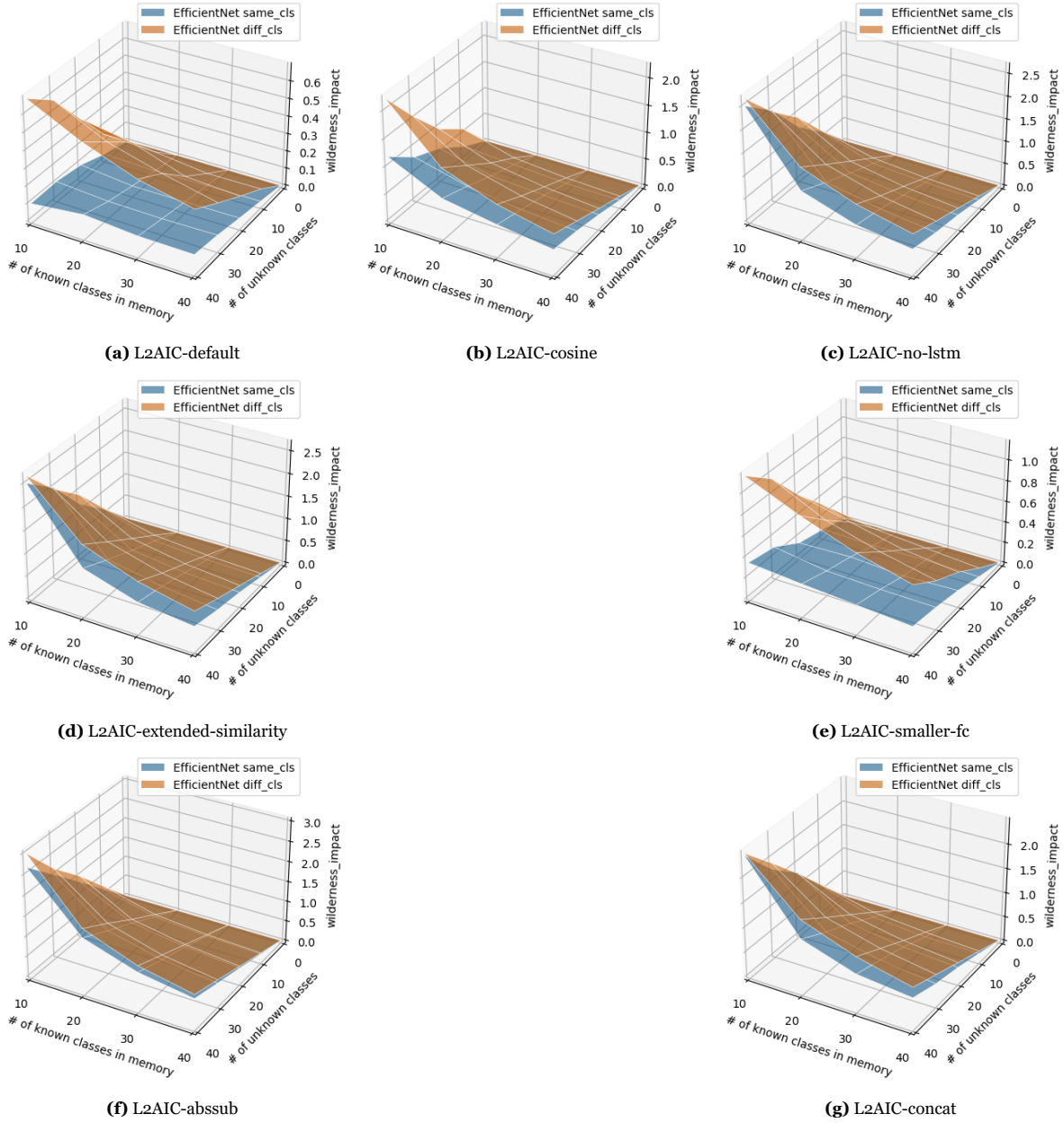
(e) L2AIC-smaller-fc

(f) L2AIC-abssub

(g) L2AIC-concat

Figure C.9: Unknown error for L2AIC with ResNet152 per architecture

C.4. Wilderness impact per architecture

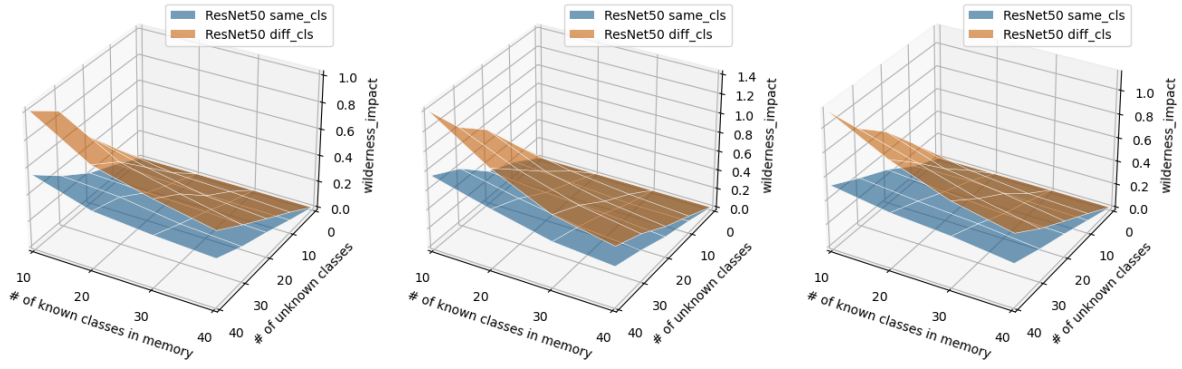


(a) L2AIC-default **(b)** L2AIC-cosine **(c)** L2AIC-no-lstm

(d) L2AIC-extended-similarity **(e)** L2AIC-smaller-fc

(f) L2AIC-abssub **(g)** L2AIC-concat

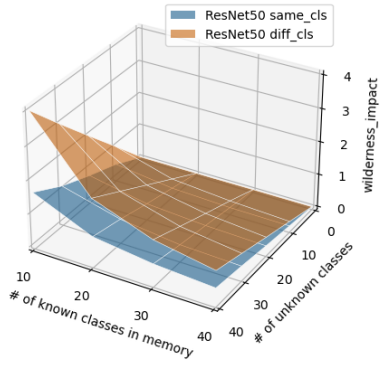
Figure C.10: Wilderness impact for L2AIC with EfficientNet per architecture



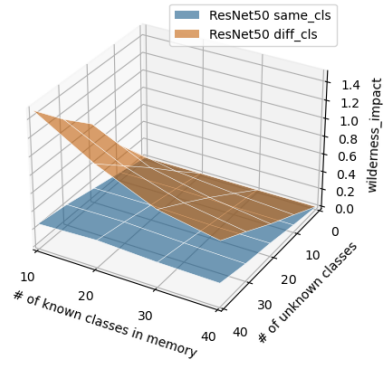
(a) L2AIC-default

(b) L2AIC-cosine

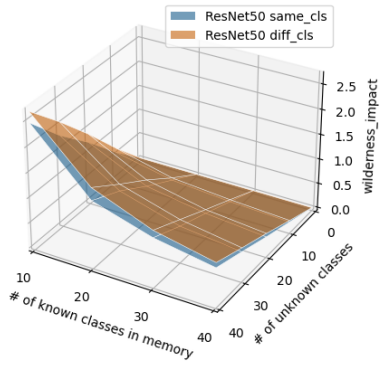
(c) L2AIC-no-lstm



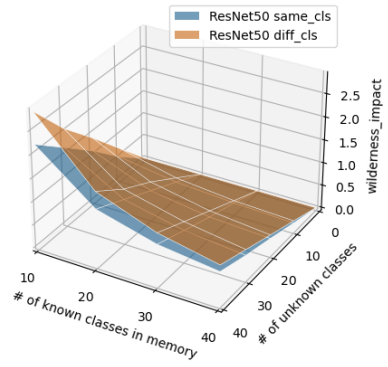
(d) L2AIC-extended-similarity



(e) L2AIC-smaller-fc

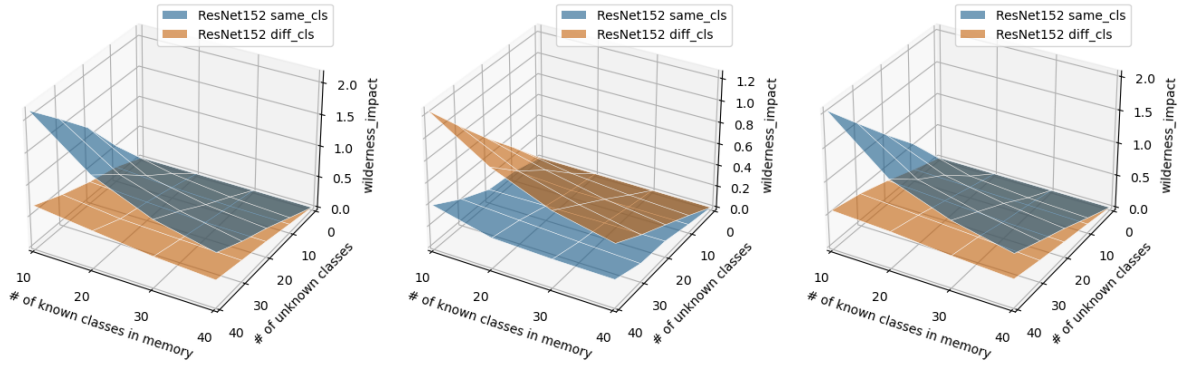


(f) L2AIC-abssub



(g) L2AIC-concat

Figure C.11: Wilderness impact for L2AIC with ResNet50 per architecture



(a) L2AIC-default **(b)** L2AIC-cosine **(c)** L2AIC-no-lstm

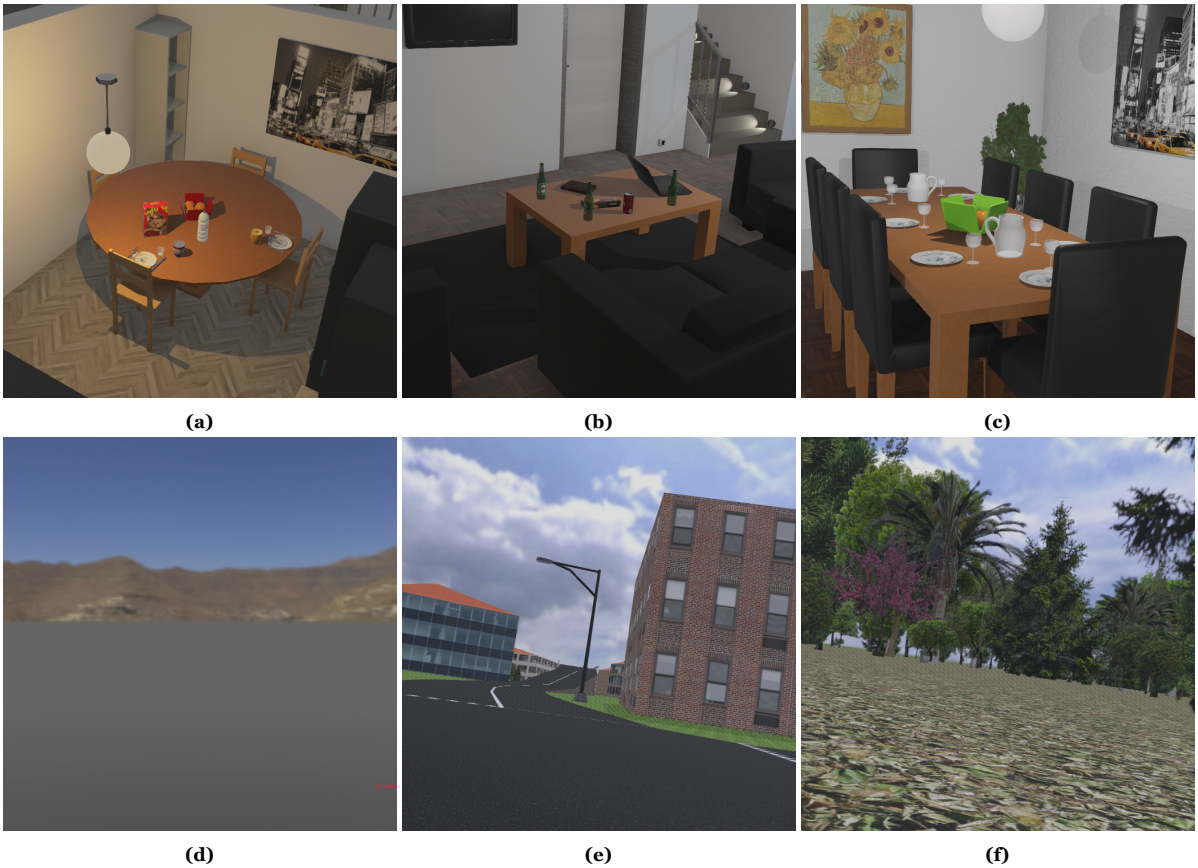
(d) L2AIC-extended-similarity **(e)** L2AIC-smaller-fc

(f) L2AIC-abssub **(g)** L2AIC-concat

Figure C.12: Wilderness impact for L2AIC with ResNet152 per architecture

D

Webots environment



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure D.1: (a) (b) (c) Three different inside environments in an apartment setting. **(d)** An environment only containing a variable background and lighting conditions. **(e) (f)** Two outside environment in a city and a park.

E

On-the-job recognition extra results

Table E.1: The performance of L2AIC-no-lstm with EfficientNet encoder on $\mathcal{D}^{photos,test}$. L2AIC has 70 classes saved in memory and the test set consists consists of an additional 30 unknown classes. The class c_{teapot} is added to the memory as different datasets, listed under **Memory addition**. Performance is both evaluated on the single teapot class as well as all test classes.

Memory addition	**Classes**	**Weighted F1**		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
None	c_{teapot}	-	-	0.0	0.0	-	-
	c_{all}	0.755	0.583	0.132	0.184	0.437	0.5
$\mathcal{D}^{Tiny,memory}$	c_{teapot}	0.347	0.039	-	-	0.79	0.98
	c_{all}	0.748	0.571	0.138	0.203	0.443	0.507
$\mathcal{D}^{photos,memory}$	c_{teapot}	0.246	0.148	-	-	0.86	0.92
	c_{all}	0.749	0.574	0.133	0.19	0.443	0.506
$\mathcal{D}^{syn}$	c_{teapot}	0.02	0.0	-	-	0.99	1.0
	c_{all}	0.748	0.572	0.133	0.19	0.445	0.507
$\mathcal{D}^{syn,close}$	c_{teapot}	0.0	0.0	-	-	1.0	1.0
	c_{all}	0.748	0.572	0.133	0.19	0.445	0.507

Table E.2: The performance of L2AIC-no-lstm with EfficientNet encoder on $\mathcal{D}^{Tiny,test}$. L2AIC has 70 classes saved in memory and the test set consists consists of an additional 30 unknown classes. The class c_{teapot} is added to the memory as different datasets, listed under **Memory addition**. Performance is both evaluated on the single teapot class as well as all test classes.

Memory addition	**Classes**	**Weighted F1**		ϵ_U		ϵ_K	
		$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$	$\mathcal{C}^{\mathcal{K}}$	$\mathcal{C}^{\mathcal{N}}$
None	c_{teapot}	-	-	0.01	0.07	-	-
	c_{all}	0.755	0.583	0.132	0.186	0.437	0.5
$\mathcal{D}^{Tiny,memory}$	c_{teapot}	0.758	0.958	-	-	0.39	0.08
	c_{all}	0.75	0.58	0.138	0.203	0.437	0.495
$\mathcal{D}^{photos,memory}$	c_{teapot}	0.058	0.0	-	-	0.97	1.0
	c_{all}	0.748	0.572	0.133	0.19	0.445	0.507
$\mathcal{D}^{syn}$	c_{teapot}	0.0	0.0	-	-	1.0	1.0
	c_{all}	0.748	0.572	0.133	0.19	0.445	0.507
$\mathcal{D}^{syn,close}$	c_{teapot}	0.0	0.02	-	-	1.0	0.99
	c_{all}	0.748	0.573	0.133	0.19	0.445	0.507