Anxo-Lois Pereira Canovas

Food image recognition by a replay-based continual learning method using uncertainty-driven past-sample selection

MASTER'S THESIS

Supervised by Dr. Eduardo Aguilar and Dr. Petia Radeva

Master's Degree in Biomedical Data Science

Barcelona, June, 2024

I would like to thank everyone who has supported me during the course of this Master's degree. Both my family and my friends. On the other hand, I would also like to thank the supervisors of this project who have helped me to carry out this study, as well as all the people and organizations that have made it possible for me to get here and do this project.

Abstract

In this paper, we propose a novel approach to food image recognition utilizing a replay-based continual learning method with uncertainty-driven past-sample selection. Our method aims to address the challenges of data variability and evolving food databases by selectively retaining and revisiting samples based on their uncertain score. This ensures robust performance and adaptability, improving the accuracy of food classification over time. The proposed system could improve significantly dietary tracking, healthcare applications, and the food industry by improving the benchmarking results of the state-of-the-art sample selection methods. We have experimented and evaluated our proposed methods and other baseline methods with which we compared them on three datasets, including FOOD101, the food classification dataset we used to validate the results. Finally, we have obtained very positive results, as we have largely outperformed the baseline sample selection methods for rehersal on the FOOD101 dataset.

Keywords

Continual learning, Replay, Rehearsal, Uncertainty, Food recognition, Food classification, Evidential Deep Learning

Dr. Eduardo Aguilar , certifies that the student *Anxo-Lois Pereira Canovas* has elaborated the work under his direction and he authorizes the presentation of this Master's Thesis for its evaluation.

Advisor(s) signature:

Firmado por AGUILAR TORRES EDUARDO ANDRES - ****8628* el día 07/06/2024 con un certificado emitido por AC FNMT Usuarios

Contents

1. Introduction

The rapid advancement of artificial intelligence (AI) and machine learning (ML) has opened new frontiers in numerous fields, with one of the most promising being food image recognition. Accurate and efficient food image recognition can facilitate automated dietary tracking, enabling individuals to monitor their nutritional intake more effectively and adhere to healthier eating habits. In healthcare, this technology can aid in managing chronic conditions like diabetes or obesity by providing precise dietary recommendations based on real-time food intake analysis. For the food industry, improved classification systems can streamline inventory management, enhance customer service in automated restaurants, and support sophisticated food delivery systems. However, developing robust AI models for this task presents significant challenges, particularly in the realm of continual learning, where the model must adapt to new data and evolving patterns without forgetting previously learned information.

Continual learning, or lifelong learning, is an approach where an AI system incrementally learns from new data while retaining knowledge from prior learning experiences. Traditional ML models typically require retraining from scratch with the entire dataset whenever new data is introduced, which is both time-consuming and computationally expensive. In contrast, continual learning aims to enable the model to learn continuously from new data streams, making the process more efficient and scalable. However, one of the critical challenges in continual learning is the phenomenon of catastrophic forgetting, where the model loses previously acquired knowledge when exposed to new data.

A promising strategy to mitigate catastrophic forgetting is replay-based continual learning. This method involves selectively replaying past samples during the training of new data to reinforce and preserve existing knowledge. However, the efficiency of this method heavily depends on the selection of past samples. Traditional random sampling approaches often fail to prioritize the most informative and relevant samples, leading to suboptimal performance and increased computational burden.

This thesis proposes an innovative replay-based continual learning method using uncertainty-driven past-sample selection, specifically tailored for food image recognition. Uncertainty allows to infer into the model's ability to give confidence in a prediction given a particular sample. If the model is very confident in the prediction of the sample, the uncertainty measure will be low, meaning that the sample is within the distribution. On the other hand, in cases where the uncertainty is high, it means that the model is not confident in the prediction. By leveraging uncertainty measures, the proposed method aims to identify and prioritize past samples that are most likely to improve the model's performance and robustness. This approach not only enhances the model's ability to retain previously learned information but also ensures that new knowledge is integrated more effectively.

The significance of improving selection methods for continual learning in food image recognition extends beyond academic interest. In practical terms, better food image recognition systems can lead to more accurate dietary tracking and nutritional analysis, aiding individuals in making healthier food choices and managing chronic conditions like diabetes and obesity. Additionally, these systems can assist in food quality control and safety monitoring, reducing the risk of foodborne illnesses and enhancing consumer trust. Moreover, in the culinary industry, improved food recognition can facilitate innovative applications such as automated meal logging, personalized recipe recommendations, and even augmented reality cooking assistants.

In summary, this thesis aims to bridge the gap between theoretical advancements in continual learning and practical applications in food image recognition. By developing a more effective and efficient replay-based continual learning method using uncertainty-driven past-sample selection, this research strives to contribute to the creation of smarter, more reliable AI systems that can continuously improve and adapt, ultimately benefiting society at large.

This thesis is organized as follows. First, the background, motivation and objectives of this study are summarized. Then we discuss the state-of-the-art that has been collected from the current literature. Afterward, we define in detail all the important points of the study, such as the rehearsal strategy followed, the baseline methods and our proposed methods. At the end we show the results of our experiments and, finally, the discussion and conclusions of this project.

1.1 Background and motivation

The global rise in diet-related health issues, such as obesity, diabetes, and cardiovascular diseases, underscores the critical need for effective dietary management. According to the World Health Organization, unhealthy diets are one of the leading causes of noncommunicable diseases, contributing significantly to premature deaths worldwide. Despite widespread awareness of these risks, many individuals struggle to adhere to dietary recommendations due to a lack of practical tools that can assist in making informed food choices.

In recent years, advancements in artificial intelligence (AI) and machine learning have opened new avenues for addressing this challenge. Among these, food image classifiers have emerged as a promising technology. These classifiers can analyze photographs of food and provide detailed nutritional information, offering a convenient and immediate method for monitoring and managing one's diet.

However, not all countries or areas eat the same foods. Due to this, it is possible that in terms of project design, it may be necessary to plan the project in phases where the foods to be trained in each phase (either by location or food groups) are not stable and it is necessary to give results in each of the phases. At the same time, it is possible that the duration of this supposed project would be very long, making the cost of storing and managing the old data for reuse, as well as the cost of re-training at each iteration with all the data would add a lot of computational costs. All of this, can greatly complicate the viability of the project. This is where continual learning comes in.

The continual learning paradigm is gaining more and more momentum in today's world due to the increasing amount of data available at one time compared to storage and computational costs. What mostly defines training in a replay-based continual learning paradigm is the rehearsal strategy used. In class-incremental visual object recognition, this strategy is in charge of deciding which images of the classes that have been trained within a specific experience, or phase, are stored in a buffer to be used in future training with unseen classes in order to avoid catastrophic forgetting of what the model has learned, i.e., to classify the images from the seen classes.

Currently, many strategies have been proposed to choose the best images during rehearsal to maximize the final results of the training. We want to improve these strategies by using uncertainty. The uncertainty implementation we use is based on Evidential Deep Learning (*EDL*), which has shown to be an efficient method to compute the uncertainty. The analysis of uncertainty in the predictions allows us to rank the sample according to what the model currently knows. High values of uncertainty imply data that are under-represented or out of distribution, while low values imply data that the model knows very well. Our **hypothesis** is that if we use this metric or score to select samples, we can get better benchmark results than the ones from baseline methods. This selection can be done in many ways, either by choosing the samples with the lowest uncertainty because the model learns them well, or by choosing the samples with the highest uncertainty and, so, combining all the samples out of the distribution. We also want to explore the combination of these strategies or other more complex ones.

With this, we want to improve the state-of-the-art results of current selection strategies for rehearsal. To compare the methods, an exhaustive analysis was made to evaluate and iteratively develop the various proposed rehearsal strategies. Finally, the results are validated by applying the best strategies on the dataset with food data.

1.2 Objective

The main objective of this project is to develop a rehearsal strategy in a continual learning paradigm based on uncertainty. This strategy should allow differentiating representative images of the classes from the images that the model considers uncertain or undefined. Finally, use this novel strategy to train, in a continual learning paradigm, a classifier that differentiates many different types of foods.

By creating this strategy, we intend to improve the benchmarks of the current rehearsal selection strategies on typical datasets, and then test their ability to generalize on other datasets by confronting it to a dataset with many different classes of foods and checking how much better the proposed strategies are and which of them is better in general.

1.2.1 Specific objectives

1. **Analyze the state-of-the-art in Continual learning and CoreSet rehearsal methods:** Investigate, learn and finally define in this study, in reference to Continual Learning and rehearsal: what it is, how it works and what different ways of implementation exist, to decide which implementation this study follows. In the same way, investigate in the current literature, which methods of sample selection for the rehearsal for Continual learning and CoreSet selection are the best right now, if there are already methods that work based on uncertainty, either the same implementation that we use or a different one and how the frameworks (datasets, metrics, etc) on which these methods are compared are usually defined.

2. **Investigate the Evidential Deep Learning implementation to quantify the uncertainty:** understand the implementation of the method to quantify the uncertainty that we are going to use (*EDL*), as well as its advantages and disadvantages, to be able to implement a sample selection method from this score.

3. **Implement the baseline methods for comparison purposes:** Design, implement and prepare the framework where the experiments are carried out. This includes the design of the training method, design of the rehearsal characteristics, dataset selection and benchmarking preparation, among others. Moreover, we want to implement several state-of-the-art baseline strategies with which to compare our proposed novel methods.

4. **Design and implement the proposed replay-based methods that include the predicted uncertainty in decision-making:** Implement the sample selection methods for uncertainty-based rehearsal. The idea we have about this implementation is quite iterative. We want to implement some methods, experiment with them and depending on the results when comparing them with the baseline methods, change them a little bit or implement different methods.

5. **Perform an experimental validation of the proposed methods quantitatively and qualitatively:** Once the implementation of all the sample selection methods, the baseline and the proposed ones have been finished, their performance is experimentally validated. To do this, multiple scenarios are prepared with various datasets.

2. State-of-the-art

For this project, it is important to consider the methods of selecting samples from a population that can give the best results to use as a baseline and compare them to the results obtained with the method proposed in this research.

First, we would like to highlight that even though the paradigm and scenario designed in this project is based on finding the sample to be used during rehearsal in continual learning, other research lines end up working similarly and, therefore, their methods may also give good results for continual learning paradigm [18], such as *CoreSet* selection.

On the other hand, it is difficult to decide which methods are exactly the best in general in the selection of samples for rehearsal in continual learning, since the results are very dependent on the design of the experiment. In particular, the results vary greatly depending on the percentage of data selected for the sample compared to the population size. Certain methods are better for very small percentage samples, while others are for very large sample sizes [7]. Therefore, this study focuses on the methods that give the best results in experiments with a setup and design similar to the one we have considered for this project, where we select only 10% of the data for rehearsal.

2.1 Rehearsal selection methods for Continual Learning

First of all, it is necessary to comment on the sample selection strategies created directly for rehearsal in the continual learning paradigm that is currently giving the best results.

2.1.1 Random sample selection

Random sample selection is one of the most commonly used methods to compare results within a continual learning paradigm with rehearsal. Specifically, *Reservoir sampling* is often used [3]. *Reservoir sampling* is a family of randomized algorithms for efficiently choosing a simple random sample. This algorithm allows to choose a sample of size $k \leq n$ from a population n by passing through each of the elements of the population only once, which makes it a very good algorithm for random selection in cases of populations with a large number of elements.

Although at first sight the random sample selection may seem to be a good lower bound when compared to other strategies, in continual learning this is not the case. In fact, random sample selection is considered one of the best sample selection algorithms in rehearsal. In many researches based on rehearsal methods in continual learning, random selection ends up being one of the best methods [3]. Specifically, in experiments where the scenario and situation is similar to this project, the random method ends up completely dominating the other methods in terms of final accuracy [3]. In addition, this method is one of the most efficient in terms of computational time, which tends to make it the most used method in this type of problem.

It is also important to note that this method is also used in other research lines, such as *CoreSet selection* [2][18][7], with similar results.

2.1.2 Forgetting Events

Another method is *Forgetting Events* [17], which is an error-based method, where the number of times the model has forgotten an item, i.e., the number of times the model incorrectly predicts an item after having it classified correctly in the previous epoch. This allows get rid of the samples that are never or rarely forgotten and to keep the samples that are more easily forgotten. This strategy also has shown good results in the *CoreSet selection* [7].

2.1.3 Influence

Example influence [16] is conceptually defined as, almost, inversely to the uncertainty. A sample has a high influence if the model learns a lot from that sample. In a dataset each training sample is different, even if it belongs to the same data distribution. Because of this, the samples contribute differently to the learning of the model, i.e., training in a continuous learning paradigm can be improved by acquiring the influence of the samples and selecting samples based on this score.

Some influence techniques have been tested in the continual learning paradigm with good results [16] and generate some interest for this study as they are conceptually contrary to our proposed method.

2.1.4 Multi-criteria subset selection

Another approach that has been followed in recent years is Multi-criteria subset selection [5]. This idea is very simple conceptually but can be applied in many ways. This strategy is based on generating several scores for each sample and calculating a final score, which is the one that is used to select samples by combining them all. These scores could be any score that could be used to select samples for rehearsal and a weighted average is usually used to combine them. In some studies, they use the vector of characteristics and the loss of the samples to sort them into three ranking lists and by combining the results of the rankings, sort the samples to select the most important ones for rehearsal [5].

2.2 CoreSet methods

CoreSet selection[2] [7][18] is a research line where the goal is to select a small percentage of the samples of a dataset, so that the costs of storing the data and the training are way more efficient. This is born out of the need to reduce the time and energy consumed in training by reducing the size of the training dataset for future iterations of model training.

This technique is very similar to the idea behind the replay-based method in a continual learning framework [2], so it is important to consider which selection methods are currently the best for this type of situation. In fact, some recent studies have tested the use of *CoreSet* selection techniques for Continual Learning [18] in various ways, such as as an alternative to rehearsal [2][8].

2.2.1 Contrastive Active learning

Initially created for Active Learning, but used with good results within the CoreSet problem, we have *Contrastive Active learning* [11], or *CAL*. Active Learning has the objective of efficiently acquiring data for annotations from a large unlabeled population. This is to optimize the human labeling effort to the most informative or important data. Although it is a different paradigm, it maintains the central idea of getting the most important small sample from a large population, as in the case of *CoreSet* or the continual learning rehearsal. Therefore, this sampling technique has been used to solve *CoreSet* [7] problems and therefore it is also considered as one of the most promising options for continual learning rehearsal.

This sample selection strategy selects samples whose predictive likelihood diverges most from that of their neighbors. It has given very good results in the *CoreSet* research line, in scenarios similar to this project [7].

2.2.2 Bilevel Optimization

Bilevel Optimization has shown state-of-the-art performance results for the *CoreSet* research line[2], but also for the *Continual learning* paradigm[8]. At its core, bilevel optimization is a special kind of optimization where one problem is embedded, or nested, within another. These two tasks are referred to as upper-level and lower-level optimization tasks. This type of approach, has witnessed and increasing popularity in machine learning in recent years, with applications from meta-learning to hyper-parameter optimization. In *CoreSet*, it has been used a weighted variant of the empirical risk minimization[2], where the goal is to optimize the lower-level optimization task:

$$L(\theta, w) = \sum_{i=1}^{n} w_i l_i(\theta) \tag{1}$$

The result of optimizing this equation is $\theta^*(w) = argmin_\theta L(\theta, w)$, and with this the upper-level, and the real objective to optimize, is $L(\theta^*(w))$. Here, $l_i(\theta) = l_i(x_i, y_i; \theta)$ which is the representation of the loss obtained from the sample x_i in the model with weights θ and w_i is the weight assigned to the sample. The difference between the lower-level and the upper-level in this case is that in the latter, $w_i = 1; \forall i \in [n]$. Note that this is only one example implementation of this strategy, and for other paradigms, other bilevel optimization-based strategies have been implemented have been implemented, like for *Continual Learning*[8].

2.2.3 Graph Cut

Graph Cut[10] is a submodular function. Submodular functions measure the diversity and information of the samples, so they can be used to select important samples. They are defined as $f : 2^V \rightarrow R$, which return a real value for any $U \subset V$. For f to be a modularity function, it must also meet the following conditions, given that $A \subset B \subset V$ and $\forall x \in V \setminus B$:

$$f(A \cup x) - f(A) \geq f(B \cup x - f(B) \tag{2}$$

There are other functions that fulfill these conditions besides *Graph Cut*, however the performance results of *Graph Cut* for the *CoreSet*[7] research line are the most competitive and promising of them all.

2.2.4 K-Center Greedy

K-Center Greedy was proposed as a CoreSet approach to Active Learning[14]. This method is a greedy approximation to the NP-hard problem of minimax facility location. This means, that with the objective of removing very similar data points, k samples are selected as S from the full dataset T such that the largest distance between a data point in $T \setminus S$ and its closest data point in S is minimized. This can be seen in the next equation 3, where $D(\cdot, \cdot)$ is the distance function:

$$min_{S \subset T} \quad max_{x_i \in T \setminus S} \quad min_{x_j \in S} \quad D(x_i, x_j) \tag{3}$$

This method has shown competitive results on the CoreSet research line[7].

2.2.5 Uncertainty methods

As methods close to our proposal, although there are not too many cases where implementation of the idea of uncertainty is used for sample selection in deep learning, there are some. Some of the examples we found are the *Least Confidence*, *Entropy* and *Margin* [6]. These methods were initially created for active learning and *CoreSet*.

It should be noted that the way of defining and calculating the uncertainty of these sample selection strategies is very different from the one used in this project. The definition and explanation of the implementation used in this study can be found in Section 3. At the same time, it is important to note that looking at the results of these techniques, they underperform the other techniques discussed in this section, at least as far as the *CoreSet* selection is concerned [7].

The uncertainty implementation used in this study to design the proposed methods is the quantified uncertainty in *Evidential Deep Learning*[15][1]. There are no existing cases in the literature today where this uncertainty implementation is used in *Continual learning* nor in *CoreSet selection*, so this study is the first one where it is evaluated in a replay-based Continual learning approach.

3. Methodology

In this study, we leverage deep learning methodologies, with a particular focus on *Continual learning* using a rehearsal strategy, to tackle the challenge of learning from a non-stationary data stream. Continual learning, also known as lifelong learning, refers to the ability of a model to learn continuously from a stream of data that evolves over time, without suffering from catastrophic forgetting of previously learned knowledge. This capability is crucial for developing adaptive systems that can operate in dynamic environments and maintain their performance over time.

The specific approach employed in this study involves the integration of a rehearsal strategy within a deep learning framework. The rehearsal strategy is designed to mitigate the issue of catastrophic forgetting by maintaining a memory buffer that stores a subset of past data. This stored data is then used alongside new data to periodically retrain the model, ensuring that it retains knowledge from previous tasks while learning new ones.

In order to do so, exhaustive research has been carried out on continual learning and rehearsal, the best sample selection methods at present, and the implementation of the uncertainty used in this study. Subsequently, all baseline sample selection methods have been implemented within a continual learning framework. The uncertainty-based sample selection methods for rehearsal have been designed and implemented iteratively by comparing their benchmarking results with the baseline results and with each other.

Figure 1 shows the steps followed in this study. Initially, a literature analysis has been done to learn and define Continual Learning, rehearsal, and state-of-the-art baseline sample selection methods to compare our projects and also about uncertainty. With this information, we implemented the continual learning framework used in the experiments, as well as the baseline sample selection methods. Finally, iteratively, we designed and implemented several uncertainty-based sample selection methods for rehearsal based on their experimental results by evaluating and comparing them with the baseline methods.

In this section, we collect all the information and definitions gathered and learned from the literature. Starting with the definition of rehearsal in continual learning and the most important characteristics that define it, followed by the baseline methods that we have selected for this study. Finally, we also define in detail the implementation of uncertainty that has been used to create the sample selection methods for rehearsal in this study.

3.1 Rehearsal in Continual Learning

To understand what exactly rehearsal is, it is first necessary to define how training works within a continual learning paradigm. In a real case, a model within a continual learning framework would be trained iteratively as new classes or new data arrive or are added to the model. To simulate it, in an experimental environment, the target dataset is separated into several groups of classes which are trained iteratively, one after the other. These training phases with subsets of classes are called experiences. Clearly, it can be understood that if the classes are completely separated, the classes of the first experience will be completely forgotten for the last one (this is called catastrophic forgetting). To prevent this from happening, rehearsal is used. As such, rehearsal consists of collecting a small subset of the experience data at the end of an experience and buffering it for use in future experiences to "remind" the model of the classes of past experiences.

It is important to note that rehearsal can be implemented in many different ways and this can greatly affect the results of an experiment. First, it is important to define how the buffer will be filled with the data

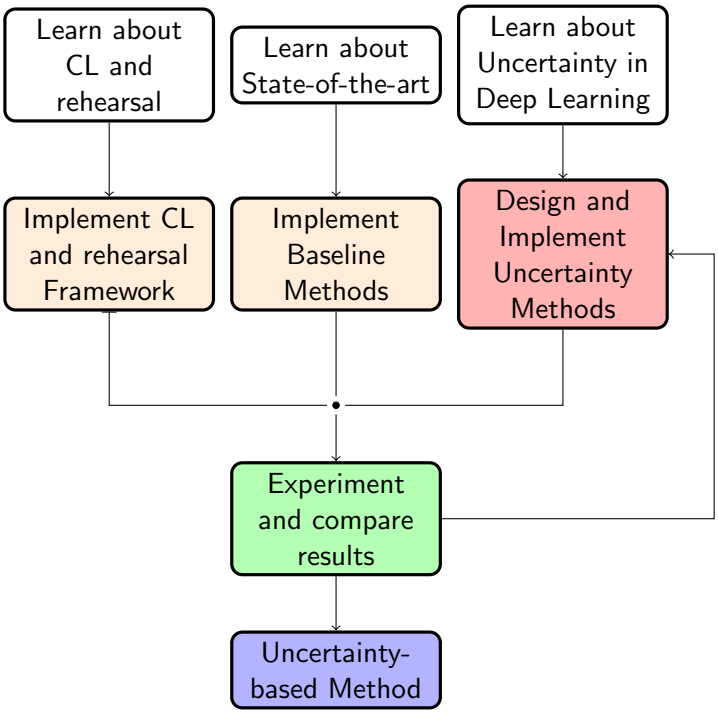

Figure 1: Project diagram

selected in the experiments. In general terms, there are two main types of buffers, fixed memory buffers and variable memory buffers. Fixed memory buffers are based on the idea of always having the same amount of data in the buffer, regardless of the training experience the model is in. For these buffers, the maximum number of data that will be stored throughout the training is defined at the start. Then, at the end of the first experience, the entire buffer is filled with data from the first classes of the training. In the rest of the subsequent experiences, the less important data from the previous experiences, that occupy buffer space are eliminated, following a defined criterion, and are replaced with the most important data from the new experience. This allows to define exactly how much data is going to be saved at the end of the training. The other type of buffer, on the other hand, stores data in a balanced way. Instead of defining the total buffer size, it is decided the percentage of data to be stored or the exact number of data to be stored per certain granularity (either by class, experience, etc). With this, the buffer size is constantly growing with each experience, but, contrary to the fixed memory buffers, the data selected as worth saving is not eliminated, and at the same time, ensures a minimum representation of all classes or experiences. For this study, we have used a variable buffer where we define the percentage of data that is stored per experience.

Another characteristic of the rehearsal, that must be clearly defined, is the amount of data to be stored in the buffer. This is important since the performance of the model will largely depend on the percentage of data that is saved and reused in subsequent experiments. The benchmarks against which a method should be compared, as well as the best methods used as a baseline, depend on this. Some methods give better results for very low percentages and others for high or medium percentages, so it is very important to clearly define in which data range an experiment is defined.

It is also important to define how the buffered samples will be used during training. In the case of this study, in each training iteration, the model is passed a batch of data consisting of two batches of equal size, one full of the data from the new experience and one full of old experiences data, so that in each experience the model has received the same amount of data from the current experience as it has received from all previous experiences combined (the buffer data). Other strategies can be implemented, such as combining without repetition the buffer data with the experience data to create the dataset loader and so the data from old experience data is not repeated in the same epoch.

Finally, the last feature of great importance that has to be defined with respect to rehearsal is the sample selection strategy. This algorithm or technique is in charge of assigning importance to each sample of the experience from which you want to select the data. With this score, the data can be sorted so that the most important ones are then selected and stored in the buffer for the following experiences. It is important to note that in a real case scenario, this feature would probably end up being the one that is most important to select carefully, since the others are usually defined by the constraints of the project (if there is no memory problem to store all the data, it does not make sense to apply a rehearsal strategy at all). Therefore, this study focuses on developing a new sample selection strategy for rehearsal and comparing it with state-of-the-art strategies.

3.2 Baseline methods

In this study, as mentioned in the previous section, we focus on the methods of sample selection for rehearsal. The other characteristics to be defined such as the type of buffer, its size, or how it is used in the training are constant in all the experiments performed, and more information about this information is found in the experiments section (Section 4).

As previously mentioned, the objective of this study is to develop a selection method based on uncertainty and to check how effective it is when compared to the best current methods. Therefore, it is necessary to prepare some baseline methods to compare their performance results with the results of the developed models.

In this case, three state-of-the-art selection methods have been used as the baseline. It is important to keep in mind that the best selection models are highly dependent on the other features mentioned above. For example, with different percentages of buffered data, different models have different results. Some models are very good for very very small percentages and bad for larger percentage ranges. Others the other way around. Therefore, in this study, we focus on the most important ones for the features used to carry out these experiments.

3.2.1 Random

Conceptually, the selection of samples for random rehearsal is very simple. With all samples having the same probability, the samples to be saved are randomly selected and added to the buffer. The implementation of this technique can vary greatly, but the final result is usually very similar. In many cases, a *Reservoir sampling algorithm* is used, which allows the selection of a sample of size k from a population of size n, provided that $n >= k$, in a very efficient way $O(n)$. In this study we use a different approach, the data is shuffled and for the rehearsal, the first samples are selected.

This study is carried out in a way where the selection strategy does not select the data as such, but it simply sorts the data in a list by preference and the rehearsal algorithm can take care of selecting the data it needs by taking the number of data it needs from the top of the list as it sees fit. With this implementation, given a list with the entire population of data from an experience, it can be shuffled so as to achieve an equivalent and equally efficient result to the one of the *Reservoir sampling algorithm*.

It is important to remember that, although it seems very simple, it is one of the best, if not the best, of the possible baselines we have found for the range of percentages of data to be stored in the buffer we have decided on for the experiments. On the other hand, this method gets much worse when using very small percentage ranges of data for the buffer ($\frac{k}{n} <= 0.01$, for example)[7]. Another great advantage of this method is its speed, being by far the fastest and most efficient of all the proposed methods.

3.2.2 CAL

Conceptually, *Contrastive Active Learning* tries to find samples that whose predictive likelihood diverges the most from their neighbors to fill the buffer.

More specifically, this implementation assigns a score to each of the samples and orders the samples with respect to this score, trying to maximize it (from highest score to lowest). This score s_i for a sample x_i is calculated, in a simplified form, such that:

$$s_i = \frac{1}{k} \sum_{j=1}^{k} KL(p(y|x_i)||p(y|x_j))$$

Where x_j with $j \in [1, k]$ being k the number of neighbors considered for the sample x_i, making x_j a neighbor of x_i. In this formula, KL is the Kullback-Leibler divergence between the two probability distributions. Finally, $p(y|x_i)$ is the predictive probability distribution of the model for the sample x_i.

It is important to note that this method was originally developed for the *Active Leraning*[11] paradigm, which is very different from the one this study focuses on, *Continual Leraning*. Therefore, we have based our implementation on the one used for the *CoreSet*[7] research line, which is much more similar.

3.2.3 Forget

The last of the methods considered baseline in this study is *Forgetting Events*. *Forgetting Events*[17] is an Error-based selection method. This selection method has a dictionary with an integer value, initialized to 0, for each of the training data of the experience being trained. Each of these values represents the number of forgets that have occurred, on the data the values refer, over all the epochs of the training experience. This value allows us to sort the data from the highest to the lowest number of forgotten events and thus to select for rehearsal the most forgetful data and at the same time to get rid of the unforgettable or easily remembered data. The dictionary is reset for each experience.

A forgetting event occurs during training when incorrectly classifying a data sample that had been correctly classified in the previous iteration, or rather epoch. Formally this is defined as $acc_i^t > acc_i^{t+1}$, where acc_i^t indicates whether the prediction of sample or data i at epoch t is correct or not (*True* or *False*).

In this study, like the CAL method, an adaptation of the implementation defined for the *CoreSet* research line[7] was used.

3.3 Uncertainty

As discussed above, the objective of this study is to develop an uncertainty-based sample selection method for rehearsal. In order to achieve this goal, it is first necessary to define uncertainty. Conceptually, uncertainty in deep learning can be interpreted as how confident the model is of the prediction it has made about a sample. In the literature, there are many interpretations and implementations of uncertainty for training deep learning models, such as the use of Kullback-Leibler diverge in CAL or other *CoreSet*[7] methods such as *Least Confidence, Entropy and Margin*[6].

However, in this study, we base the implementation of uncertainty on that presented in the work of Sensoy et.al[15]. The first thing to consider in understanding how this definition of uncertainty works, is the evidence e and how it is calculated:

$$e_i = \sigma(f_\theta(x_i)); \quad \alpha_i = e_i + 1. \tag{4}$$

Here, the evidence e_i of a sample x_i is given by the initial formula 4. The function $f_\theta(\cdot)$ returns the output logits, or prediction, of a sample using a neural network with the θ weights. It is important to note that this neural network does not have a softmax layer or another activation layer at the end, which makes the result of applying $f_\theta(x_i)$ on a sample x_i return the output logits of the sample x_i prediction. To finish calculating the evidence, a non-linearity must be applied to ensure that the evidence is non-negative, this function is the function $\sigma(\cdot)$. There are several functions that can be used to fulfill the role of the $\sigma(\cdot)$ function, for example, in the original paper where this definition of uncertainty was defined and developed, they used the *ReLU* function. However, in this study, we ended up using the exponential function $exp(\cdot)$, which is non-linear and ensures that the result is greater than 0, and is considered to be more stable than *ReLU*[1]. Finally, we have the definition of α in equation 4, which is how the evidence is used to calculate the uncertainty, as this ensures that the value of α is greater than, or equal, to 1. Finally, taking into account that K is the number of classes in the experiment, the uncertainty is calculated such that:

$$S_i = \sum_{j=1}^{K} \alpha_{ij}; \quad u_i = \frac{K}{S_i} \tag{5}$$

where α_{ij} is the value of α for sample x_i for class j. With this and taking into account that $\alpha_{ij} \in [1, \text{inf})$, $S_i \in [K, \text{inf})$ is ensured. Thus, $u_i \in (0, 1]$ where the value of 1 is only taken, i.e. maximum uncertainty, if the value of α, and therefore the evidence, is minimum ($\alpha_{ij} = 1, \forall j$). With this definition, an uncertainty value can be assigned to each of the training samples, at the moment of training the model with those samples, to obtain a value that can be sorted for each of them, so that it can be used to select samples for the rehearsal.

3.3.1 Evidential Deep Learning Loss

For the uncertainty to be properly calculated and used for sample selection in rehearsal, it is necessary that the logits resulting from the model have values that allow the correct interpretation of the evidence. For this, it is necessary to change the training loss from *Cross-Entropy* to *EDL*. Originally, the use given to the definition of uncertainty used in this study was to check how confident the model was about the predictions of a given sample on a classical Deep Learning framework. In order to give a correct prediction of this value, the designers of this method, or score, developed a loss function based on this implementation of uncertainty to train the models. This loss function is based on the Evidential Deep Learning[15] method, or *EDL* for short, and is the *Type II Maximum Likelihood*. For simplicity, we will refer to this loss as *EDL* in the remainder of this study.

Given the sample x_i and the ground truth of the same sample y_i, the *EDL* loss function is calculated such that:

$$y_{ij} = \begin{cases} 1, & \text{if } y_i = j \\ 0, & \text{otherwise} \end{cases}; \quad L_i = \sum_{j=1}^{k} y_{ij} \times (log(S_i) - log(\alpha_{ij})) \tag{6}$$

The value L_i is the principal term of the loss of the sample x_i. However, the *EDL* loss does not end here, it is necessary to add a *KL-divergence* term . This divergence term is calculated as follows:

$$\alpha_{KL_{ij}} = e_{ij} \times (1 - y_{ij}) + 1; \quad S_{KL_i} = \sum_{j=1}^{K} \alpha_{KL_{ij}} \tag{7}$$

$$KL_{1_i} = ln\Gamma(S_{KL_i}) - \sum_{j=1}^{K} ln\Gamma(\alpha_{KL_{ij}}) + \sum_{j=1}^{K} ln\Gamma(1) - ln\Gamma(K) \tag{8}$$

$$KL_{2_i} = \sum_{j=1}^{K} (\alpha_{KL_{ij}} - 1) \times (\frac{\Gamma'(\alpha_{KL_{ij}})}{\Gamma(\alpha_{KL_{ij}})} - \frac{\Gamma'(S_{KL_i})}{\Gamma(S_{KL_i})}) \tag{9}$$

$$KL_i = C_{ann} \times (KL_{1_i} + KL_{2_i}) \tag{10}$$

In these equations, $ln\Gamma(\cdot)$ is the natural logarithm of the absolute value of the gamma function $\Gamma(\cdot)$, such that $ln\Gamma(\cdot) = ln|\Gamma(\cdot)|$. At the same time, $\frac{\Gamma'(x)}{\Gamma(x)}$ is the logarithmic derivative of the gamma function.

On the other hand, C_{ann} is the annealing coefficient, which can be defined in many ways as another hyper-parameter. It can be defined as a constant value or as a value that mutates as training progresses. In the case of this study, we maintain the initial default value $C_{ann} = 1$ throughout the training. Finally, the *EDL* loss function is defined as:

$$L_{EDL_i} = 0.9 \times L_i + 0.1 \times KL_i \tag{11}$$

Algorithm 1 Uncertainty computation

procedure AFTER TRAINING ITERATION
Input:
    logits ← list of output logits from the batch data
    real_Index ← real data index of the batch samples
On memory:
    bs ← training batch size
    num_classes ← number of classes in the dataset
    uncert_dict ← dictionary with the uncertainity of each sample of the experience
Computation:
    evidence ← *exp(logits)*
    alpha ← $e + 1$
    uncert ← *num_classes/sum(alpha, dim=1)*
Store:
    uncert_dict[real_Index] ← *uncert*

3.4 Uncertainty-based sample selection strategies

As discussed in previous sections, the objective of this study is to develop a sample selection strategy for rehearsal in continual learning based on uncertainty, as defined in the last section. Uncertainty is a metric that can be assigned to each of the samples that have been used in the training of an experience separately. From the uncertainty of the samples, in this study, we propose several solutions to select samples, some simpler and some more complex.

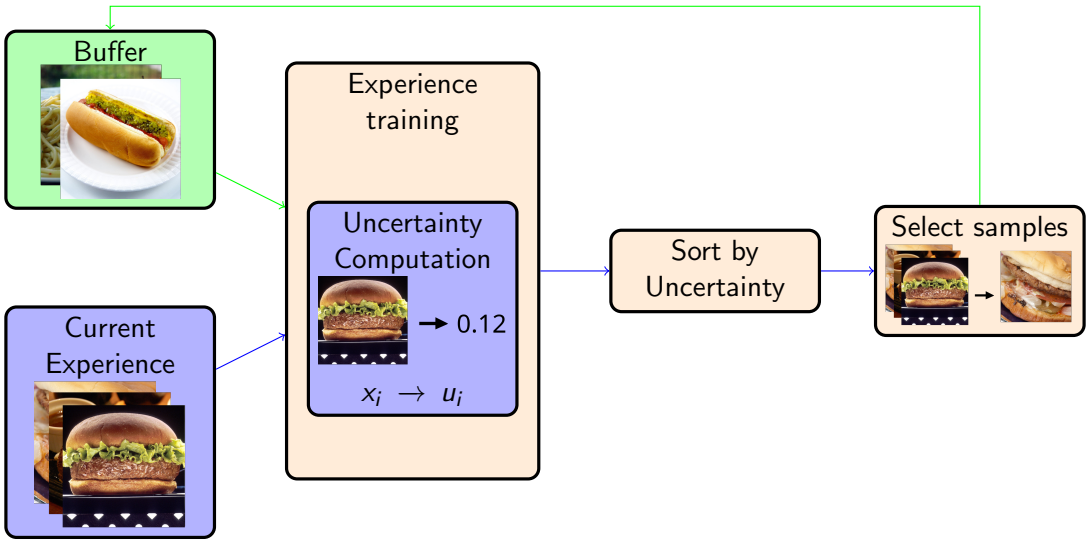

Figure 2: Simple Uncertainty computation and sample selection diagram

Starting with the simplest, we have several proposals based on sorting the samples according to their uncertainty. To begin with, the strategy that we named *Simple Uncertainty* (algorithm 2), is to choose the samples with the lowest uncertainty, so that the model will train in future experiences with the samples that the model can classify more reliably. A diagram of the training process of an experience using *Simple*

Uncertainty can be seen in the figure 2. The inverse of this strategy, choosing the samples with the highest uncertainty, which we call *Inverse Uncertainty* (algorithm 3), proposes to choose the samples of which the model is less certain and can take insight from the rarest cases. Subsequently, we have the two techniques that combine these two. First, by taking the most extreme samples in terms of uncertainty, which would be the *Half Uncertainty* (algorithm 4), and the case of choosing the less extreme samples, i.e. the samples more central or closer to the median in terms of uncertainty. We call this last strategy *Middle Uncertainty* (algorithm 5).

Algorithm 2 Simple Uncertainty

procedure SIMPLE UNCERTAINTY SAMPLE SORTING
Input:
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    sorted_examples_index ← *argsort*(*uncert_list*)
    sorted_examples_real_index ← *real_Index*[*sorted_examples_index*]
    return *sorted_examples_real_index*

Algorithm 3 Inverse Uncertainty

procedure INVERSE UNCERTAINTY SAMPLE SORTING
Input:
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    sorted_examples_index ← *argsort*(*uncert_list*)
    inverse_sorted_examples_index ← *sorted_examples_index*[:: −1]
    sorted_examples_real_index ← *real_Index*[*inverse_sorted_examples_index*]
    return *sorted_examples_real_index*

Another approach this study followed is to ensure that the chosen samples are as evenly distributed as possible with respect to the uncertainty, we propose to use a clustering of the samples with the K-Means algorithm. The same number of samples can be chosen, if possible, from each cluster to ensure that the selected data samples are as evenly distributed as possible. The samples can then be chosen in many ways, for example, randomly, which we named *Kmeans Random Uncertainty* (algorithm 6), or by choosing the most central sample iteratively, either the one closest to the mean (*Kmeans Mean Uncertainty*) (algorithm 8) or the median (*Kmeans Median Uncertainty*) (algorithm 7).

Finally, eliminating the samples with the highest uncertainty before applying another strategy to ensure that the samples chosen are not too far outside what the model has been able to learn, is another approach this study follows. Here this idea is combined with two other methods. First with the best baseline, random, by choosing the samples randomly over the non-eliminated samples, *Filtered Random* (algorithm 9). Another strategy proposed is to apply this idea of eliminating the samples and then applying the technique of *Kmeans Random Uncertainty*, which we call *Kmeans Filtered Uncertainty* (algorithm 10)

Algorithm 4 Half Uncertainty

procedure HALF UNCERTAINTY SAMPLE SORTING
Input:
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    base_sorted_examples_index ← *argsort*(*uncert_list*)
    sorted_examples_index ← []
    max_ind ← *int*(*length*(*uncert_list*)/2)
    for $i \in [0, max\_ind)$ **do**
        sorted_examples_index.append(*base_sorted_examples_index*[i])
        sorted_examples_index.append(*base_sorted_examples_index*[$-i$])
    sorted_examples_real_index ← *real_Index*[*inverse_sorted_examples_index*]
    return *sorted_examples_real_index*

Algorithm 5 Middle Uncertainty

procedure MIDDLE UNCERTAINTY SAMPLE SORTING
Input:
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    base_sorted_examples_index ← *argsort*(*uncert_list*)
    max_ind ← *int*(*length*(*uncert_list*)/2)
    sorted_examples_index ← [*base_sorted_examples_index*[*max_ind*]]
    for $i \in [1, max\_ind]$ **do**
        sorted_examples_index.append(*base_sorted_examples_index*[$max\_ind - i$])
        if $i + max\_ind < length(uncert\_list)$ **then**
            sorted_examples_index.append(*base_sorted_examples_index*[$max\_ind + i$])
    sorted_examples_real_index ← *real_Index*[*inverse_sorted_examples_index*]
    return *sorted_examples_real_index*

Algorithm 6 Kmeans Random Uncertainty

```
1: procedure KMEANS RANDOM UNCERTAINTY SAMPLE SORTING
2: Input:
3:     nC ← number of clusters for the KMeans algorithm
4:     real_Index ← real data index of the experience samples
5:     num_classes ← number of classes in the dataset
6:     uncert_list ← list with the uncertainity of each sample of the experience
7: Kmeans:
8:     kmeans ← KMeans(nC)
9:     kmeans < −kmeans.fit(uncert_list)
10:    cluster_data ← {}
11:    sorted_examples_index ← [ ]
12:    remaining_clusters ← [0, 1, ... , nC − 1]
13:    for i ∈ remaining_nc do
14:        cluster_data[i] ← index of the samples in cluster i in kmeans
15:        cluster_data[i] ← shuffle(cluster_data[i])
16: Sorting:
17:    while length(sorted_examples_index) < length(uncert_list) do
18:        vals_per_cluster ← int((length(uncert_list)−length(sorted_examples_index))/length(remaining_clusters))
19:        if vals_per_cluster < 1 then
20:            vals_per_cluster ← 1
21:        for i ∈ remaining_clusters do
22:            if length(cluster_data)[i]) > vals_per_cluster then
23:                sorted_examples_index.append(cluster_data[i][:vals_per_cluster])
24:                Remove cluster_data[i][:vals_per_cluster]
25:            else
26:                sorted_examples_index.append(cluster_data[i])
27:                Remove cluster_data[i]
28:    sorted_examples_real_index ← real_Index[sorted_examples_index]
29:    return sorted_examples_real_index
```

Algorithm 7 Kmeans Median Uncertainty

procedure KMEANS MEDIAN UNCERTAINTY SAMPLE SORTING
Input:
    $nC \leftarrow$ number of clusters for the KMeans algorithm
    real_Index $\leftarrow$ real data index of the experience samples
    num_classes $\leftarrow$ number of classes in the dataset
    uncert_list $\leftarrow$ list with the uncertainity of each sample of the experience
Kmeans:
    kmeans $\leftarrow$ *KMeans*(nC)
    kmeans $< -$*kmeans.fit*(*uncert_list*)
    cluster_data $\leftarrow \{\}$
    sorted_examples_index $\leftarrow$ [ ]
    remaining_clusters $\leftarrow [0, 1, \dots, nC - 1]$
    for $i \in$ *remaining_nc* **do**
        cluster_data[i] $\leftarrow$ *index of the samples in cluster i in kmeans*
Sorting:
    while *length(sorted_examples_index)* $<$ *length(uncert_list)* **do**
        vals_per_cluster $\leftarrow$ *int*((*length(uncert_list)*$-$*length(sorted_examples_index)*)/*length(remaining_clusters)*)
        if *vals_per_cluster* < 1 **then**
            vals_per_cluster $\leftarrow 1$
        for $i \in$ *remaining_clusters* **do**
            if *length(cluster_data)[i])* $>$ *vals_per_cluster* **then**
                mid_val $\leftarrow$ *int*(*length(cluster_data[i]) / 2*)
                min_val $\leftarrow$ *max(mid_val - vals_per_cluster, 0)*
                max_val $\leftarrow$ *vals_per_cluster* $\times 2 -$ (*mid_val* $-$ *min_val*) $+$ *mid_val*
                sorted_examples_index.append(cluster_data[i][min_val:max_val])
                Remove *cluster_data[i][min_val:max_val]*
            else
                sorted_examples_index.append(cluster_data[i])
                Remove *cluster_data[i]*
    sorted_examples_real_index $\leftarrow$ *real_Index*[*sorted_examples_index*]
    return *sorted_examples_real_index*

Algorithm 8 Kmeans Mean Uncertainty

procedure KMEANS MEAN UNCERTAINTY SAMPLE SORTING
Input:
    $nC \leftarrow$ number of clusters for the KMeans algorithm
    real_Index $\leftarrow$ real data index of the experience samples
    num_classes $\leftarrow$ number of classes in the dataset
    uncert_list $\leftarrow$ list with the uncertainity of each sample of the experience
Kmeans:
    kmeans $\leftarrow$ *KMeans*(nC)
    kmeans $< -$ *kmeans.fit*(*uncert_list*)
    cluster_data $\leftarrow \{\}$
    sorted_examples_index $\leftarrow$ [ ]
    remaining_clusters $\leftarrow [0, 1, \dots, nC - 1]$
    for $i \in$ *remaining_nc* **do**
        cluster_data[i] $\leftarrow$ *index of the samples in cluster i in kmeans*
Sorting:
    $i \leftarrow 0$
    while *length(sorted_examples_index)* $<$ *length(uncert_list)* **do**
        clstr $\leftarrow i\%nC$
        if *length(cluster_data[clstr])* > 0 **then**
            cluster_uncert $\leftarrow$ *uncert_list*[*cluster_data*[*clstr*]]
            $m \leftarrow$ *sum(cluster_uncert)/length(cluster_uncert*
            mean_diff $\leftarrow$ [ ]
            for $x \in$ *cluster_uncert* **do**
                mean_dif .append($|x - m|$)
            selected_val $\leftarrow$ *argmin*(*mean_dif*)
            sorted_examples_index.append(selected_val)
            Remove *cluster_data[clstr][argmin(mean_dif)]*
    sorted_examples_real_index $\leftarrow$ *real_Index*[*sorted_examples_index*]
    return *sorted_examples_real_index*

Although there are many proposed strategies, the experiments in this study focus on finding the best one by comparing their results and the results with the baselines to determine if these methods are better.

Algorithm 9 Filtered Random

procedure FILTERED RANDOM SAMPLE SORTING
Input:
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    perc_rm ← percentage of removed data
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    sorted_examples_index ← *argsort*(*uncert_list*)
    split_index ← *length*(*uncert_list*) × *perc_rm*
    sorted_examples_index ← *shuffle(sorted_examples_index[:split_index])*
    sorted_examples_real_index ← *real_Index*[*sorted_examples_index*]
    return *sorted_examples_real_index*

Algorithm 10 Filtered Kmeans Random

procedure FILTERED KMEANS RANDOM SAMPLE SORTING
Input:
    nC ← number of clusters for the KMeans algorithm
    real_Index ← real data index of the experience samples
    num_classes ← number of classes in the dataset
    perc_rm ← percentage of removed data
    uncert_list ← list with the uncertainity of each sample of the experience
Sorting:
    sorted_examples_index ← *argsort*(*uncert_list*)
    split_index ← *length*(*uncert_list*) × *perc_rm*
    sorted_examples_index ← *sorted_examples_index[:split_index]*
    selected_uncert_list ← *uncert_list*[*sorted_examples_index*]
    return *KMeans_Random_algorithm(nC, real_Index, num_classes, selected_uncert_list)* (algorithm 6)

4. Experiments Design

This section explains how the experiments performed in this study were designed, as well as all the important decisions made in designing them.

4.1 Datasets

Three different datasets were used in this study, all with a different motif. One was used for testing the implementations, checking everything worked as expected and to do some benchmarking in a small and simple dataset. Another one was used as an extension of the first dataset, to see how the implemented methods performance results differ when confronted with a more complex dataset. The final dataset is used to compare the results in a case very different from the other two and much more complex, while completing the ultimate goal of classifying food data.

First, we used *CIFAR10*, which is a relatively simple dataset. This dataset has 10 classes with 6,000 images each, 1,000 images in the evaluation set and 5,000 in the training set, making a total of 50,000 training images and 10,000 evaluation images. Having only 10 classes makes it a relatively simple dataset to train in a continual learning paradigm and being the color images (*RGB*, 3 channels) quite small (32×32) allows fast training and testing. Moreover, this dataset is widely used as a standard benchmark in the literature to which we want to compare, as well as in many other paradigms of deep learning in image classification. For these two reasons, we have used *CIFAR10* in this study for initial testing, during the implementation of the models and strategies, as well as to discard ineffective techniques in early phases, and for one of the final benchmarks of the developed sample selection strategies and baselines.

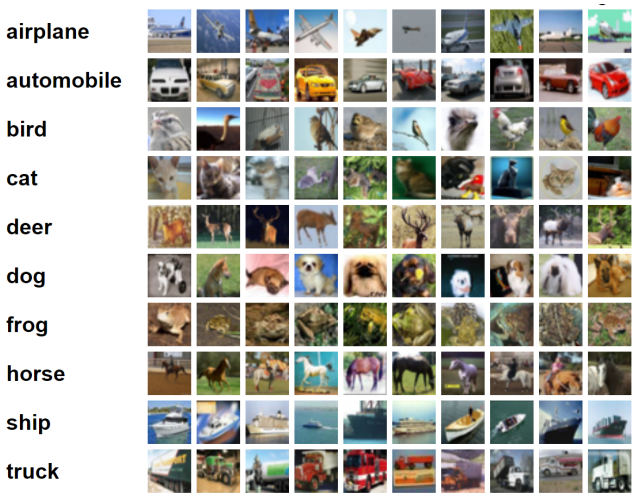

Figure 3: Classes and examples of *CIFAR10*

Continuing along the same line we have *CIFAR100*. This dataset is a kind of extension of *CIFAR10*, where the total number of images is the same but the number of different classes has been increased to 100. This way, each class has 600 color images, 500 training images and 100 evaluation images, with the same size as in the other dataset (32 x 32). This dataset has almost the same advantages as *CIFAR10*, the images are relatively small which makes the training quite fast and at the same time it is widely used

as a standard benchmark. The main difference, which makes it interesting to use this dataset in addition to *CIFAR10*, is that having 10 times more classes, and 10 times fewer images per class, complicates the training. For these reasons, *CIFAR100* is used as an extra standard benchmark to compare the results of the proposed strategies with the baseline.

Finally, the last dataset used, and the one that defines the completes this study, is the *FOOD101* dataset. This dataset contains 101 different kinds of food images. For each class there are 750 training images and 250 evaluation images, making a total of 101000 images. The size of these images is not always the same, unlike the other two datasets. Although it is not as standard a benchmark as the other two datasets used, it is still widely used. Therefore, in this study we use this dataset to compare the best sample selection strategies for rehearsal. It should be noted that this dataset is more complex and larger than the other two, which makes it difficult to train with these images, so we have only used it as a final benchmark and not for the initial and intermediate tests during the development of the methods.

Figure 4: Examples of *FOOD101* dataset.

4.2 Framework: Avalanche

Avalanche[4] is an End-to-End Continual Learning Library based on PyTorch, born within ContinualAI, with the goal of providing a shared and collaborative open-source codebase for fast prototyping, training and reproducible evaluation of continual learning algorithms.

We have decided to use the avalanche framework in this study to facilitate the implementation of the selection methods thanks to the great modularity of this library. Another important reason why we have used it, is that, by having many methods and models implemented, as well as the skeleton for deep continual learning based on rehearsal, we could focus the development time on the main research of the study, instead of wasting much of the limited time on implementing the foundations of continuous learning framework.

4.3 Models

In this study, we have mainly used two distinct models. One for the initial phases and one to evaluate the final benchmarks.

4.3.1 ICARL-Net

For the most initial phases of the project, i.e. during the implementation and initial testing of the sample selection strategies, both the ones developed in this study and the baseline strategies. For this, the model is called *ICarl-Net*.

This model is a modification of a *ResNet-18* that was originally created to implement the *ICarl*'s[13] training. There are mainly two reasons why we use this model in initial phases. First, this model was already implemented within the Avalanche framework and also was already prepared to work with the *CIFAR10* dataset without requiring any extra internal modifications to the model or the dataset. On the other hand, this model is a simplified version of a *ResNet-18* model, this implies that it is much lighter, *ICarl-Net* weighs in memory a total of *1.7MB* and *ResNet-18* weighs a total of *44.6MB*. Because of this, the initial training, to test the correct functioning of the implementations, in a rather restricted cloud environment, in the initial phases was greatly accelerated by using this module. This model is, in turn, very similar to a *ResNet18* model, which we used as the final benchmark model. All this favored the use of this model initially.

4.3.2 ResNet-18

ResNet or residual neural networks are deep learning models widely used for computer vision. They are based on Residual Blocks and have been evolving over time since their creation in 2015[9], making even today their newer versions standard models for image classification with competitive results.

Residual blocks are the defining characteristic of *ResNet*. Instead of following the classical mode where the input is transformed by a series of convolutional layers and then by an activation layer, in residual blocks the input is added to the output of the block. To form *ResNet* networks, many of these residual blocks are stacked.

These networks usually give quite good results regardless of the training dataset, which has led to be a standard benchmarking model within an image classification problem. Specifically, *ResNet-18* (and *ResNet-32*) is widely used in the literature to compare benchmarkings[12][1]. For these reasons, we have decided to use it within our study as the model to train and evaluate our sample selection strategies for rehearsal.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections

Figure 5: *ResNet-18* base architecture. In this study the last fully connected layer size changes depending on the dataset trained.

4.4 Evaluation Metrics

To evaluate and compare the sample selection models for rehearsal, in this study, we have mainly used accuracy, which computation follows the equation 12, where TP = True positive, FP = False positive, TN = True negative and FN = False negative. In particular, the mean evaluation accuracy on the test set. Although the average accuracy sometimes presents some problems for its interpretation in unbalanced datasets, as all the datasets we used in this study are perfectly balanced at the class level, this is not a problem.

$$Acc = \frac{TP + TN}{TP + TN + P + FN} \tag{12}$$

On the other hand, the average accuracy for the classes of each experience separately in experiences subsequent to the first ones where that classes samples were trained, is also considered to some extent.

This study also took into account the *Forgetting* metric. This metric is computed as the average over the difference between the accuracy result obtained after the first training on an experience and the accuracy result obtained on the same experience data at the end of successive experiences, as in the equation **??**, where $nE = \#$ of experiences, Acc_{final_i} is the accuracy of the experience i after the whole training finished (after last experience training) and Acc_{exp_i} is the accuracy of the experience i after the training of the said experience i has finished. This means, that unlike accuracy, the less *Forgetting* value the better the result.

The differences in execution times and RAM usage as a metric of the effective performance of each strategy, are also a metric considered when comparing results.

$$Forgetting = \frac{1}{nE} \sum_{i=1}^{nE} Acc_{exp_i} - Acc_{final_i} \tag{13}$$

4.5 Hyper-parameters and Decision

In order to carry out all the experiments it was first necessary to agree and define several points of the training, both in terms of the design of the experiments and the hyper-parameters we have used.

First, in all experiments for all datasets and models equivalently, we defined the type of buffer and prepared the continual learning framework. As previously mentioned, we used a buffer of variable memory size, where, in a balanced way (per experience), we kept a small percentage of the samples for rehearsal. Specifically, we decided to keep 10% of the samples of each experience in the buffer. We focus on this range of percentage of samples because, for this range in the literature the random selection baseline gives very good results compared to more complex strategies.

On the other hand, the main hyper-parameters for training and rehearsal depended largely on the dataset, but remained stable throughout all experiments done with each dataset. The Table 1 lists these main parameters.

Dataset	**Epochs/EXP**	**Classes/EXP**	**Number EXPs**	**Number Classes 1st EXP**
CIFAR10	10	2	5	2
CIFAR100	50	20	5	20
FOOD101	120	10	10	11

Table 1: Hyper-parameters used in the training of experiments per Dataset. In this context, "*EXP*" means experience.

5. Results

Before starting to look at the results, it is important to remember that the metrics we will mainly focus on to evaluate and analyze an experiment are:

- The final accuracy of the model at the end of all experiments when evaluated with all dataset classes. Equation 12
- The final accuracy of the model at the end of all the experiments when evaluated with all the classes of the first experiment. Equation 12
- The final forgetting of the model. Equation 13

Once the *Simple Uncertainty* method worked with *EDL* correctly, we experimented to compare its results with the three baseline sample selection methods and to compare, these baseline methods, with each other. The results are shown in Table 2.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	**0.4221**	**0.5115**	**0.1383**
CAL	0.2917	0.1975	0.3943
Forget	0.306	0.1035	0.4257
Simple Uncertainty	0.3635	0.3715	0.1410

Table 2: Initial experiment results. The best selection strategy in the initial setup is *Random*, followed by the proposed method *Simple Uncertainty*. The other two baseline methods are under-performing.

On the one hand, the results for *Random* sample selection are clearly superior to the other strategies, followed relatively closely by the proposed *Simple Uncertainty* strategy. On the other hand, the results of the two other baseline strategies (*CAL* and *Forgetting Events*) are abysmally inferior. It is necessary to add that these two samples have an extra problem which is the considerable increase of computational cost, both, in RAM memory usage and in training time. In the case of *Forgetting Events*, this is greatly accentuated. At the time we started with the real tests with the benchmarking model (*ResNet-18*), even by greatly reducing the size of the training batch size, we could not finish execution without first running out of GPU RAM space. For all these reasons, combined with the computational and time constraints of this study, we abandoned the *CAL* and *Forgetting Events* methods as the baseline for the rest of the experiments.

After making sure that the *Simple Uncertainty* sample selection method worked correctly, we did the same with the implementations of the other uncertainty-based sample selection methods and then we experimented to see what results they gave in comparison to the basic one. In this case, we tested all the basic selection methods, which would be the *Inverse Uncertainty*, the *Half* and the *Middle*, and also the two initial Kmeans based methods, both *Kmeans Mean* and *Kmeans Random*. The other methods were designed after these initial experiments. It is important to note that, although we tested the *Kmeans Random* method for multiple numbers of clusters, since the implementation of *Kmeans Mean* is much

Strategy	Final Acc 2nd Experience	First Experience Classes Final Acc	Forgetting
Simple	0.6930	0.6845	**0.022**
Inverse	0.6478	0.598	0.2325
Half	0.6378	0.6635	0.167
Middle	0.6450	0.6355	0.195
Kmeans Mean ($k = 10$)	0.680	0.676	0.1545
Kmeans Random ($k = 5$)	0.6765	**0.775**	0.0655
Kmeans Random ($k = 10$)	0.6915	0.7185	0.122
Kmeans Random ($k = 15$)	0.6950	0.7320	0.1085
Kmeans Random ($k = 50$)	**0.703**	0.7455	0.095

Table 3: Comparison of sample selection methods based on uncertainty for only two experiences in *CIFAR10* (two classes each experience).

slower, we only did one to see its viability as a strategy. Also, from here experiment onward, all the tests were done with the *ResNet-18* architecture. The results are shown in Table 3.

Two main conclusions can be drawn from these results. On the one hand, the results of the basic methods of *Inverse*, *Middle* and *Half* are well below the rest, including the *Simple Uncertainty* selection method. This suggests that, in general, data with lower uncertainty give better results than data with high uncertainty, since not including them at all in the most basic methods (all but *Simple Uncertainty*, i.e., *Inverse*, *Middle* and *Half*) gives better results. On the other hand, the *Kmeans*-based methods, both *Random* and *Mean*, give more promising results. Therefore, in the next experiments the *Simple Uncertainty*, *Kmeans Mean* and *Kmeans Random* methods are going to be evaluated, but the others will be discarded.

To better compare the results of the *Kmeans Mean* strategy with the other selected strategies, including the *Random* Baseline strategy, we experimented with them again. The results can be found in Table 4.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	0.4752	0.407	0.3245
Simple Unc.	0.4609	0.399	0.2182
Kmeans Mean ($k = 5$)	0.5604	0.348	0.2225
Kmeans Mean ($k = 10$)	0.5610	0.3895	**0.2099**
Kmeans Mean ($k = 15$)	0.5526	0.4005	0.2180
Kmeans Random ($k = 5$)	0.5529	0.3565	0.2578
Kmeans Random ($k = 10$)	0.5469	0.2940	0.2634
Kmeans Random ($k = 15$)	**0.5770**	**0.4075**	0.2119
Kmeans Random ($k = 50$)	0.5600	0.3885	0.2363

Table 4: Comparison of selected sample selection methods based on uncertainty in *CIFAR10* with the random Baseline.

As in the previous experiment, *Kmeans Random* gives better results than all the others. This strategy is closely followed by the *Kmeans Mean* strategy. The baseline *Random* strategy falls far behind these two

strategies, by a total of 7% in total accuracy difference with the best strategy, and is closely followed by the *Simple Uncertainty* selection strategy. However, apart from the fact that we have not yet discussed 3 uncertainty-based selection strategies, this experiment was only done with a single seed, so even though it is very promising, it is not clear that the *Kmeans* strategies are superior to the *Random* baseline.

At this point, we started with the three remaining sample selection methods. The ideas behind implementing these methods are based on the results or problems of the other methods already mentioned. To begin with, the *Filtered Random* sample selection method is born from the *Simple Uncertainty* sample selection results, where we concluded that samples with higher uncertainty gave worse results than samples with lower uncertainty. With this idea, we implemented a method based on the *Random* baseline, but only applying the sample selection after eliminating from the set of possible samples to be selected the samples with higher uncertainty.

As we initially did not know what percentage of eliminated samples would give the best resultwe performed several experiments, the results of which are shown in Table 5.

Losss	**Removed images**	**Final Acc**	**First Experience Classes Final Acc**	**Forgetting**
EDL	10%	0.5494	0.3570	0.2288
EDL	20%	**0.5704**	0.3975	**0.1527**
EDL	25%	0.5588	0.3545	0.2466
EDL	40%	0.5449	0.3855	0.2545
EDL	50%	0.5413	**0.4305**	0.2085

Table 5: Comparison of results with the Filtered Random sample selection strategy with and different percentages of samples removed.

Even though only one experiment has been done with each percentage of deleted images, some conclusions can be drawn. With both loss functions, the best percentage of sample removal is 20%, so in the rest of the experiments, this percentage is used.

We then implemented the last two sample selection methods for rehearsal. First, *Filtered Kmeans Random* as a mixture of *KMeans Random* with *Filtered Random*. This idea was born from trying to get a better result from *Filtered Random*, which seems to give the best results, by combining it with a *Kmeans*, so that ensures that the selected samples are evenly distributed over all the data, except of course for those initially eliminated, to see if the results of comparing *Random Baseline* with *Kmeans Random*'s, where *Kmean Random* gave vastly better results than *Random* base, extrapolated to this case. On the other hand, to create a method similar to *Kmeans Mean* but much faster, we created *Kmeans Median*, where the median is used instead of the mean to select the samples. The results of these two methods in comparison to the *Random Baseline* and the *Filtered Random* methods are shown in Table 6.

The results are not too significant, at least not enough to eliminate a strategy before the final experiments. On the one hand, the *Filtered Random* strategy has given the best average accuracy results of all, with a margin of 2% of the second, and also the best *Forgetting* score. On the other hand, *Filtered Random Kmean* follows it closely and both, *Kmeans Median* and the *Random* baseline, have similar results. The final experiment with multiple seeds will include all these methods.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	0.6090	0.3715	0.2079
Filtered Random	**0.6827**	0.5775	**0.06112**
Filtered Random Kmean ($k = 5$)	0.6628	0.6120	0.07188
Filtered Random Kmean ($k = 10$)	0.6323	0.5760	0.1196
Filtered Random Kmean ($k = 15$)	0.6508	0.5100	0.0975
Filtered Random Kmean ($k = 20$)	0.6243	0.5080	0.1314
Filtered Random Kmean ($k = 50$)	0.6516	**0.6175**	0.1000
Uncertainty Kmean Median ($k = 5$)	0.5762	0.4505	0.2219
Uncertainty Kmean Median ($k = 10$)	0.5359	0.5560	0.2884
Uncertainty Kmean Median ($k = 15$)	0.6315	0.5445	0.1439
Uncertainty Kmean Median ($k = 20$)	0.6007	0.5070	0.1943
Uncertainty Kmean Median ($k = 50$)	0.5995	0.3985	0.1869

Table 6: Comparison of the merged modes (*Filtered* methods: *Filtered Random* and *Filtered Kmeans Random*) with the *Uncertainty Kmeans Median* strategy for multiple clusters (k).

In order to do the final benchmarking experiments, it is first necessary to agree on the learning rate to be used for all training with all sample selection methods. To do this, we trained the models several times, with multiple combinations of learning rate and sample selection strategy, to decide which learning rate is used for the final experiments on the *CIFAR10* datasets (on which these learning rate experiments are done), as well as on *CIFAR100* (whose images have the same specifications as the other dataset). The results of these experiments are summarized in the table 7.

lr	Mean Final Acc	Max Final Acc	Min Final Acc
0.01	0.3327	**0.6828**	0.1468
0.005	**0.6171**	0.6778	**0.5542**
0.001	0.5228	0.5697	0.4609
0.0005	0.4580	0.4939	0.3916
0.0001	0.2700	0.2839	0.2559

Table 7: Comparison of different learning rates in training. Various methods based on uncertainty and the random baseline were executed for each learning rate.

Clearly, the learning rate that gives the best results is $lr = 0.005$. Smaller learning rates fail to give competitive results. On the other hand, the learning rate of 0.01, gives very mixed results. Some cases give mediocre results and some give very good results. Specifically, it seems that it depends on the loss function used, for *EDL* the results are catastrophic, but for *CE* the results are very good. Given the nature of this case, we have preferred to use the general best case $lr = 0.005$ in all the cases.

The final dataset, *FOOD101*, which was the final objective of this study, has images of a relatively different nature than the other two datasets. Therefore, we decided to repeat the learning rate experiments, to find out which value gave better results for the benchmarking. The results can be found in the table 8. In this case, we have not tested with the smaller learning rates because in order to properly train the

models with this dataset, it was necessary to train with many epochs, and the number of experiments had to be reduced.

lr	Mean Final Acc	Max Final Acc	Min Final Acc
0.01	0.4022	**0.4636**	0.3413
0.005	0.3918	0.4377	0.3498
0.001	**0.4065**	0.4538	**0.3543**

Table 8: Comparison of different learning rates in training for the FOOD101 dataset. Various methods based on uncertainty and the random baseline were executed for each learning rate.

All the learning rates tested gave similar results. In the end, we decided to use $lr = 0.001$ for the experiments as it was a little more stable and with a little more average accuracy than the others.

5.1 Final Results Comparison

Finally, we have selected the best strategies, whenever it was feasible to train them due to the computational cost, to train the *ResNet-18* model. Using the parameters stipulated in Table 1, we trained on the *CIFAR10* dataset with five different seeds for each strategy. As this dataset is relatively simple, we have not only tested the fastest and best strategies, but also some slower ones and, in some cases, multiple tests for the same strategy, such as different cluster numbers for *Filtered Kmean Random* or different loss functions for the *Filtered Random* strategy. The results are summarized in Table 9.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	0.6387 ± 0.017	0.6124 ± 0.2751	0.1212
Filtered Random	$\mathbf{0.6510 \pm 0.024}$	0.5772 ± 0.1528	0.1178
Simple Uncertainty	0.5363 ± 0.067	0.5159 ± 0.0750	0.2270
Kmean Random (k=15)	0.6252 ± 0.066	0.5285 ± 0.1415	0.1578
Kmean Mean (k=15)	0.6299 ± 0.059	0.5350 ± 0.1545	0.1536
Kmean Median (k=15)	0.638 ± 0.024	0.5617 ± 0.1663	0.1306
Filtered Kmean (k=15)	0.6458 ± 0.036	0.5910 ± 0.1345	**0.1085**
Filtered Kmean (k=50)	0.6438 ± 0.055	$\mathbf{0.6398 \pm 0.0606}$	0.1183

Table 9: Final comparison of the selected best methods on the *CIFAR10* dataset.

The best sample selection strategy based on the average final test accuracy at the end of training, is *Filtered Random*. However, it is important to note a few things, first that both, *Random* baseline and *Filtered Random* strategies, have a small variance in results. In contrast, the *Kmean*-based sample selection methods, with the exception of *Kmean Median*, have a much larger variance. In particular, all *Kmean*-based selection strategies, together with the *Filtered Random* strategy of course, have given considerably larger maximum results than the *Random* baseline method. In reference to the other metrics, the best strategy would be one of the *Filtered Kmean*, followed by *Filtered Random*. In this dataset, multiple strategies significantly outperform the *Random* baseline.

However, it should be noted that *CIFAR10* is a somewhat simple dataset, and the results may not necessarily be the same in a much more extreme case. Therefore, we tested the same experiments on a similar, but more complex dataset, which is *CIFAR100*. As this dataset is much more complex, more training epochs are needed, and in turn, more run time per experiment. Therefore, we have not tested the slowest strategy, *Kmeans Mean*.

As discussed above, we used the hyper-parameters written in the 1 table for the *CIFAR100* dataset and the learning rate used for *CIFAR10* $lr = 0.005$, since the images in both datasets are of the same nature. In the table 10 the results are shown.

In this case, the strategy with the best average results in terms of accuracy was the *Random* baseline strategy. It is important to note that the variance of this method is practically non-existent, well below the others. Thus, the methods of *Filtered Kmeans*, and more notably *Simple Uncertainty*, that even have lower average final accuracy, have better maximum results in this metric. On the other hand, for the other metrics, the *Simple Uncertainty* strategy gives consistently better results.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	$\mathbf{0.5801 \pm 0.0038}$	0.4360 ± 0.0235	0.3536
Filtered Random	0.5670 ± 0.0127	$\mathbf{0.5241 \pm 0.0374}$	**0.2739**
Simple Uncertainty	0.5689 ± 0.0276	0.4940 ± 0.0455	0.2943
Kmean Random (k=15)	0.5250 ± 0.0283	0.4499 ± 0.0266	0.3312
Kmean Median (k=15)	0.4958 ± 0.0332	0.4209 ± 0.0506	0.3630
Filtered Kmean (k=15)	0.5513 ± 0.0110	0.4994 ± 0.0426	0.2980
Filtered Kmean (k=50)	0.5684 ± 0.0184	0.5155 ± 0.0585	0.2792

Table 10: Final comparison of the selected best methods on the *CIFAR100* dataset.

Finally, using the hyper-parameters found in Table 1, we trained the most promising methods on the *FOOD101* dataset. It should be noted that, due to the very slow training with this dataset, only three methods have been trained and with only three seeds, compared to the five seeds used for the other two datasets. The results can be found in Table 11.

The training methods used for this dataset are the ones that gave the most promising results in the experiments done to select the best learning rate for the *FOOD101* dataset. These are:

- *Random* Baseline, even if its results were not entirely good, is the baseline and we should compare the results with this strategy.
- *Simple Uncertainty*, as with *CIFAR100*, seems to be giving the best results more or less consistently.
- *Filtered Random*, in the few tests of learning rates, I got very similar results to *Simple Uncertainty*.

Strategy	Final Acc	First Experience Classes Final Acc	Forgetting
Random	0.4139 ± 0.0702	0.2792 ± 0.0430	0.4492
Filtered Random	0.4737 ± 0.0640	0.4015 ± 0.0374	**0.3247**
Filtered Kmeans Random	0.4423 ± 0.0807	$0.3638. \pm 0.0882$	0.3435
Simple Uncertainty	$\mathbf{0.4856 \pm 0.0586}$	$\mathbf{0.4316 \pm 0.0553}$	0.3333

Table 11: Final comparison of the selected best methods on the *FOOD101* dataset

The results in this dataset are very favorable for the uncertainty-based strategies. *Simple Uncertainty* outperforms the *Random* baseline sample selection strategy mean final accuracy metric, by a total of 7%, with the difference in final accuracy being even more extreme for the first experience samples. On the other hand, *Filtered Random* follows the results just below Simple Uncertainty. Finally, *Filtered Kmeans Random*, is behind the other two uncertainty-based methods, but still outperforms the baseline.

5.2 Comparison with the State-of-the-art

The final results of some of the proposed uncertainty-based sample selection strategies for rehearsal are really good when compared to *Random* Baseline. However, for each dataset the conclusions are different.

To start with the initial *CIFAR10* dataset, we look at the results of all the strategies that have been tested. The results can be viewed in tabular form in Table 9 or in graphical form for easy comparison in Figure 6. The mean final accuracy for all sample selection strategies, with the exception of *Simple Uncertainty*, on the evaluation set, after training is finished, is similar to the one of the baseline. However, the *Filtered Random* sample selection strategy, gives a higher average accuracy, by about 1.5% improvement. On the other hand, multiple strategies give a final accuracy, with at least one of the five seeds used, higher than the maximum value achieved by the *Random* Baseline strategy. The improvements, in terms of maximum accuracy achieved per strategy, are between 1.5% and 4%. The strategies that meet this condition are:

- *Filtered Random*
- *Filtered Kmeans Random*
- *Kmeans Random*
- *Kmeans Mean*

It should be noted that although the highest final accuracy result of the *Kmeans Median* strategy is higher than that of the *Random* Baseline, the difference is about 0.6% and since the final average accuracy is below the baseline's, we do not consider this strategy to be considerably better than *Random* Baseline.

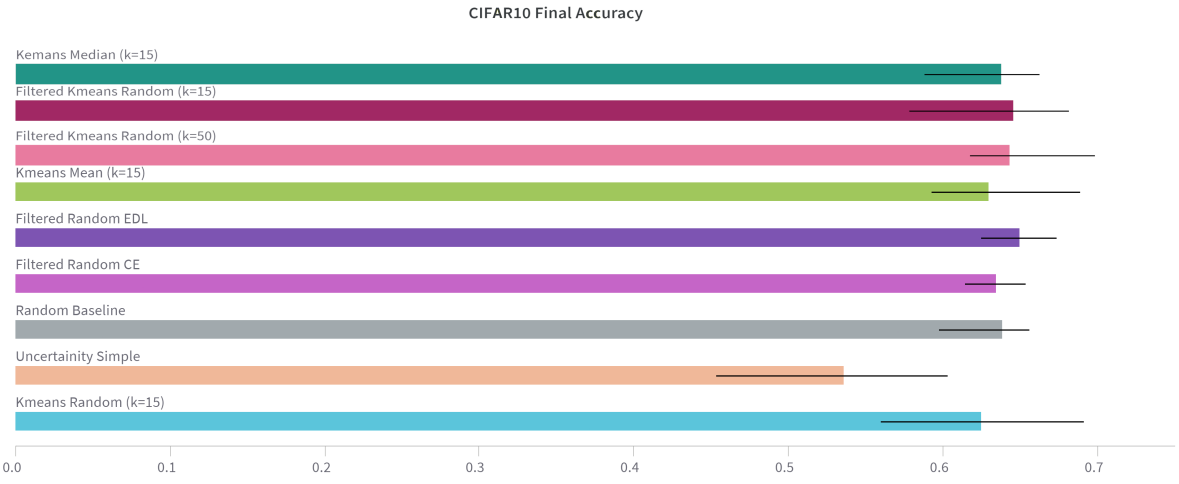

Figure 6: Barplot results of the *CIFAR10* experiments

The results for *CIFAR100* are quite different from those obtained for the *CIFAR10* dataset, and as with *CIFAR10*, the results can be seen in both Table 10 and Figure 7. The sample selection strategy with the highest final average accuracy is the *Random* baseline, and by a considerable 1.2% difference from the second best method, which would be *Filtered Kmeans Random*. However, in terms of maximum accuracy achieved by the proposed methods compared to the baseline, the *Simple Uncertainty* strategy exceeds the maximum results of the Random Baseline strategy, by a total of 1.3%. This is interesting for two reasons when taking into account the similarities and differences of the *CIFAR10* and *CIFAR100* datasets, where the image specifications are the same, but the total number of classes as well as the number of images per class changes widely. First, the strategy of *Simple Uncertainty* in *CIFAR10* gives very poor results, while in this dataset it gives very good and competitive results. On the other hand, the more complex strategies have ended up giving worse results in general.

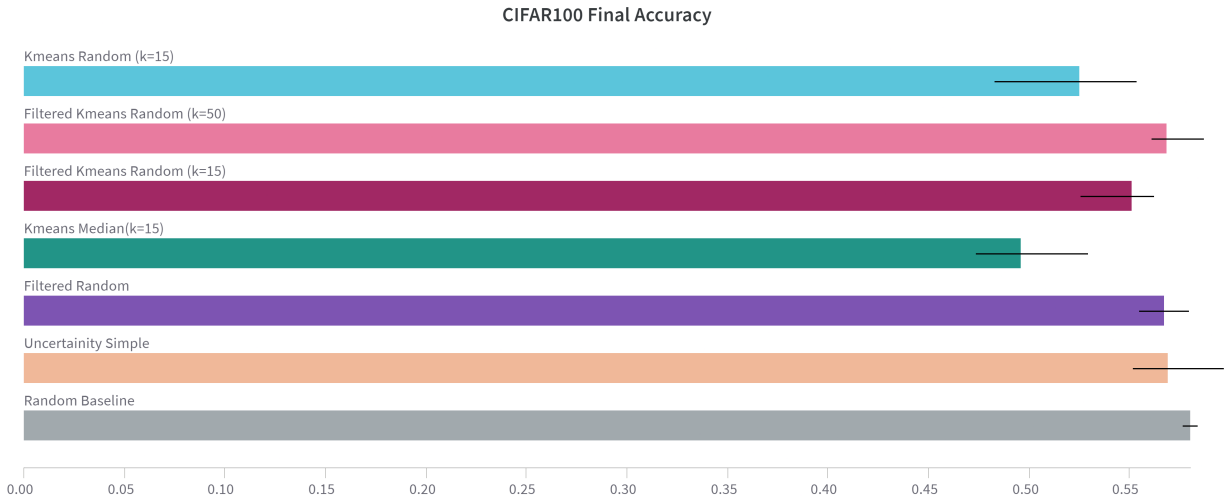

Figure 7: Barplot results of the *CIFAR100* experiments

Finally, let us review the results of the *FOOD101* dataset. These results can be seen in Table 11 and Figure 8. In this case, due to the constraints of this project, far fewer models were evaluated, so we do not have the results of the others. The two uncertainty-based strategies evaluated are:

- *Filtered Random*
- *Simple Uncertainty*

They have outperformed, by a good margin, the *Random* Baseline results. On the one hand, the final average accuracy of these strategies exceeds by 6% the *Random* baseline's, in the case of *Filtered Random* and 7% in the case of *Simple Uncertainty*, which is a lot. Similarly, at the level of the maximum accuracy of each strategy among all seeds where it has experimented, the *Filtered Random* selection strategy outperforms the baseline strategy by 5% and the *Simple Uncertainty* strategy by 6%.

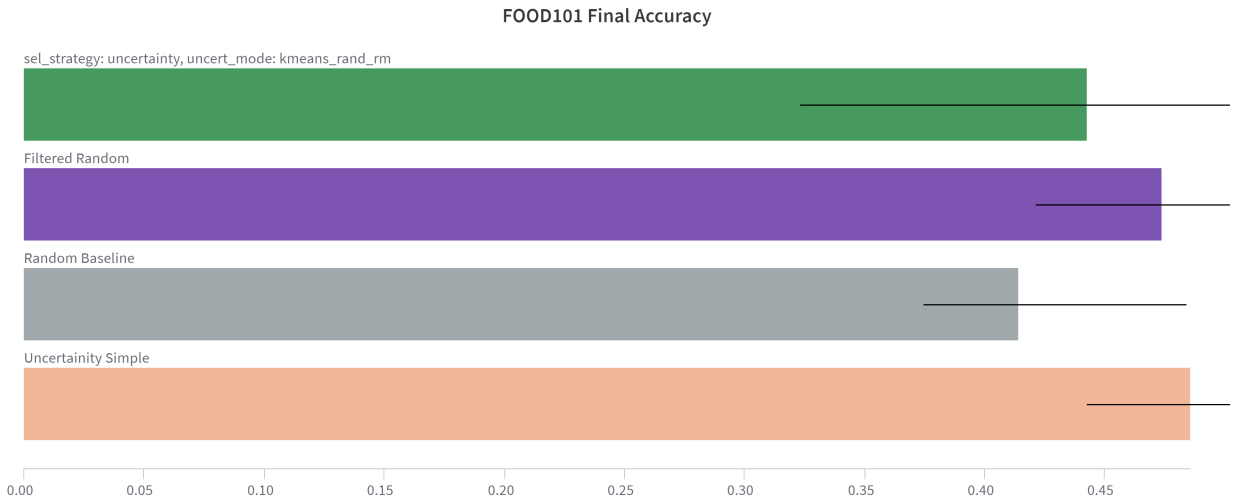

Figure 8: Barplot results of the *FOOD101* experiments

Finally, the comparison of these methods in terms of computational cost should be discussed. Firstly, in terms of execution time there are two types to take into account. The first is the training time difference within the training phases and the second is the time difference in sample selection.

The training time for each Mini-Batch can be seen in the figure 9. From the figure it can be seen that there is no noticeable difference between different uncertainty-based strategies and *Random* Baseline. This implies that the process of calculating the uncertainty of the samples being trained during training does not add appreciable computational cost. It is important to note that the big jump over the 4000 steps in this figure is due to the fact that, starting from the second experience, samples from the rehearsal buffer are added to the training Mini-Batch, thus doubling its size. and thus, increasing the time needed to train.

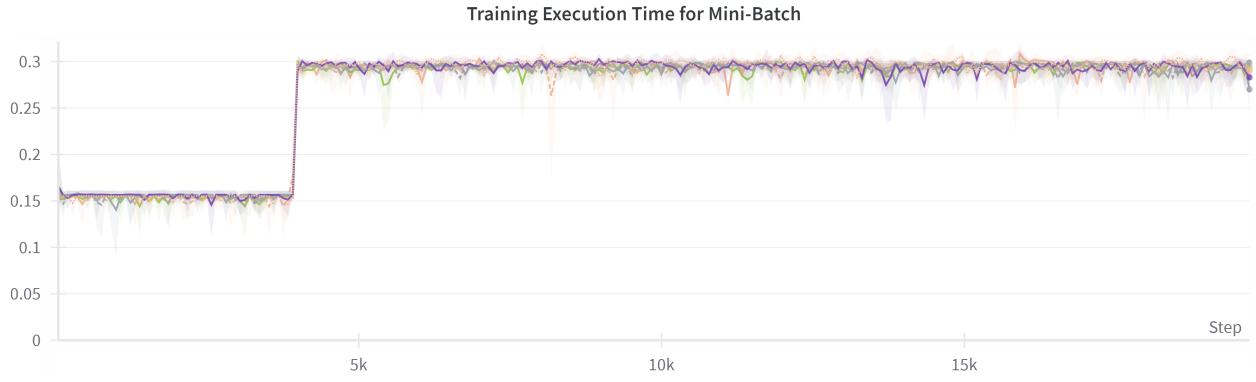

Figure 9: Time cost for each Mini-batch in the training phase in the *CIFAR100* dataset.

On the other hand, the sample selection phases do not show a very large difference, especially the simpler techniques such as *Simple Uncertainty*. However, the rest of the more complex strategies, all based on *Kmeans* in particular, do show a little more slowness when compared. This is due to the time

it takes to run the *Kmeans* model on the data. This difference, when compared to the total training cost of the models within the *Continual Learning* paradigm, is negligible. This excludes the *Kmeans Mean* and *Forgetting Events* strategies, which, as already discussed in the section 5, the selection time is too long on large datasets, and thus, these strategies were abandoned in the early stages. These selection times could take up to several hours depending on the case.

Another metric that may be interesting to compare is the RAM usage for each strategy. Figure 9 illustrates the average maximum RAM usage for each training Mini-Batch. Here it can be seen that there is some difference between the *Random* Baseline strategy (gray in the figure) and the other uncertainty-based strategies. This is because all the uncertainty strategies save, after each Mini-Batch, the uncertainty calculated from the logits generated by the model during training to avoid having to do an extra evaluation period to calculate the uncertainty value for all samples within the sample selection process, which would add a significant and unnecessary computation time to the process.

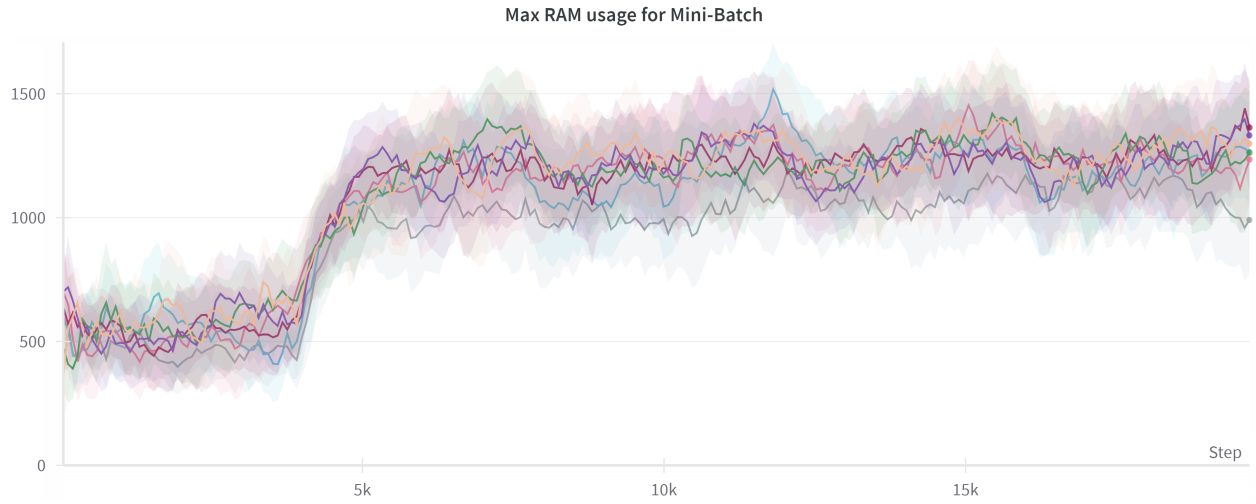

Figure 10: Mean max RAM usage for each Mini-batch in the training phase in the *CIFAR100* dataset.

6. Discussions

As a reminder, the main objective of this study was to create a sample selection method for rehearsal within a continual learning paradigm and to improve the current state-of-the-art benchmark rankings with this novel strategy. In general terms, it can be said that we have succeeded in surpassing this objective.

6.1 Conclusions

To begin with, all the research objectives have been met. We could not have reached the end of this study without understanding how Continual Learning or uncertainty works, because we could not have implemented an environment for experimentation and sample selection methods. Therefore, we believe that we have succeeded in meeting the initial objectives correctly, and this is demonstrated in sections 3 to 6.

In terms of what we have achieved by developing the sample selection methods for rehearsal, we have managed to overcome the baseline, to a greater or lesser extent, in all the datasets on which we have experimented. On the one hand, in the, rather auxiliary, datasets *CIFAR10* and *CIFAR100*, which have been used mainly for initial experiments or to test if the results are generalizable, we have achieved a strategy that exceeds in average final accuracy the random baseline and a method that in some cases exceeds the maximum value achieved by the baseline, respectively. Although the results are positive, it is worth mentioning that we would have liked to exceed in average final accuracy the baseline in both datasets, and that the difference would have been greater and more consistent. However, as I said, these are very positive results.

On the other hand, there is the dataset on which this study was focused, FOOD101, since the main objective of the sample selection method for rehearsal was to use it for an incremental classification of food images, i.e. in Continual Learning. In this case, we are very convinced of the good results we have obtained, outperforming the baseline by a significant margin, both on average and in the maximum cases.

We expected to obtain better results is in the consistency of the results of the proposed methods, but at the end this was not the case. We would have liked to find a method that was better overall in all the datasets, however, we found a very clear difference in which methods perform the best in each of the selected datasets. To be able to define in which cases each of the methods performs better than the others would require a large number of experiments, and this is beyond the scope and capabilities of this study.

6.2 Limitations

The main limitations of the results are due to the limitations of this study. Mainly due to the tight time limit that this project has by nature.

First of all, due to this lack of time, we have had to discard methods, both proposed and baseline, at very early stages in the first dataset used. These methods, even if in the most initial phases they gave lower results than the others, it is possible that they would outperform the other methods by evaluating them with multiple seeds with the final model or on other more complex datasets. In the same way, having three different datasets where to evaluate and compare the strategies, and that two of them were quite complex and therefore needed more training time, has made that many experiments of certain strategies that could perform well and compared positively with the rest of strategies have been discarded or simply were not evaluated. This can be seen very clearly, for example, the *Kmeans Mean* strategy, which has given very good performance for *CIFAR10*. But, because it took so long to do the sample selection, it blocked the rest of the experiments and, in order to finish the important experiments within the time limit of this project, it was necessary not to launch these experiments with *Kmeans Mean* strategy. Another clear example of this case is that for the final dataset, and the most important for this project, only three different strategies have been launched, the two most promising of previous experiments and a baseline strategy for comparison.

6.3 Future work

Linking with the limitations of this study, we would like to finish doing the experiments that have been missing in this study. This includes evaluating all baseline methods on all datasets, as well as evaluating methods that we have discarded, or avoided, for the larger datasets. We also would want to carry out more hyper-parameter experiments on multiple datasets for some strategies, such as the number of clusters for the *Kmeans*-based strategies or the percentage of removed images in the so-called *Filtered* methods.

Another thing we would like to do is a more exhaustive study of the sample selection times for all the strategies. Because even if the differences are not very significant in most of the strategies, it would be interesting and it would bring another point of view, to infer in more detail in the comparison of strategies and to draw more conclusions or possible improvements.

On the other hand, we would have liked to have more time to evaluate and experiment with other sample selection methods. There are other methods already defined in the literature that could give very interesting results, such as *Graph Cut*, which gives very good results within the CoreSet selection research line[7], or as an alternative to uncertainty, we could have implemented *Example Influence*[16], which is contextually defined as the opposite of uncertainty and its calculation is very different.

Finally, we plan to submit this study to the *3rd Workshop on Uncertainty Quantification for Computer Vision* in an attempt to publish it.

References

[1] Eduardo Aguilar, Bogdan Raducanu, Petia Radeva, and Joost Van de Weijer. Continual evidential deep learning for out-of-distribution detection, 2023.

[2] Zalán Borsos, Mojmír Mutný, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming, 2020.

[3] Daniel Brignac, Niels Lobo, and Abhijit Mahalanobis. Improving replay sample selection and storage for less forgetting in continual learning. In *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 3532–3541, 2023.

[4] Antonio Carta, Lorenzo Pellegrini, Andrea Cossu, Hamed Hemati, and Vincenzo Lomonaco. Avalanche: A pytorch library for deep continual learning. *Journal of Machine Learning Research*, 24(363):1–6, 2023.

[5] Gong Cheng Jifeng Ning Chen Zhuang, Shaoli Huang. Multi-criteria selection of rehearsal samples for continual learning, 2022.

[6] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning, 2020.

[7] Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning, 2022.

[8] Jie Hao, Kaiyi Ji, and Mingrui Liu. Bilevel coreset selection in continual learning: A new formulation and algorithm. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 51026–51049. Curran Associates, Inc., 2023.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Rishabh Iyer, Ninad Khargonkar, Jeff Bilmes, and Himanshu Asnani. Submodular combinatorial information measures with applications in machine learning, 2021.

[11] Katerina Margatina, Giorgos Vernikos, Loïc Barrault, and Nikolaos Aletras. Active learning by acquiring contrastive examples, 2021.

[12] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D. Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification, 2022.

[13] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016.

[14] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018.

[15] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty, 2018.

[16] Qing Sun, Fan Lyu, Fanhua Shang, Wei Feng, and Liang Wan. Exploring example influence in continual learning, 2022.

[17] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning, 2019.

[18] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. *CoRR*, abs/2106.01085, 2021.