Computer Vision & DL and Advanced Programming for AI

Resnet with SEED: a comparison of different ResNets in the class incremental setting

Bergamasco Davide, Alberto Righetti

Department of Computer Science
Master Degree of Artificial Intelligence
University of Verona
29/04/2024

Contents

Abstract

This project aims to compare the residual networks ResNet18, ResNet50, and ResNet101 as feature extractors on CIFAR100 and Food101 datasets, by using SEED, which is an ensemble method proposed by Rypesc in the Divide and Not Forget paper, to address the class incremental problem. In this problem the classes are presented little by little, collected into tasks: the model has only access to the data in the current task, therefore, it is prone to catastrophic forgetting of previously acquired knowledge. The general idea is to directly diversify experts by training them on different tasks and combining their knowledge during the inference. We evaluate the performance of the three residual networks in terms of classification accuracy and a forgetting measure.

1 Motivation

Image classification is a problem that was solved in a better way using neural networks, in particular, the ResNet architecture with the introduction of skipping connection allows to learn weights by understanding the transformation that is applied by weights to an input, which in some sense smooth and facilitate the training phase. In the case of Class-Incremental learning the data to classify is presented to the learner sequentially in a stream. The model has only access to the data in the current task. Therefore, it is prone to **catastrophic forgetting** of previously acquired knowledge. The challenge of catastrophic forgetting has spurred research in Class Incremental Learning (CIL), especially with the focus on exemplar-free solutions. While storing exemplars can be effective, it indeed comes with limitations, *particularly concerning privacy and memory constraints.* Exploring alternative approaches not reliant on exemplars opens up possibilities for more versatile and scalable solutions. In this project, we will explore and test a novel ensembling method for exemplar-free CIL called **SEED[1]**: *Selection of Experts for Ensemble Diversification.* It's a new method that leverages an ensemble of experts where a new task is selectively trained with only a single expert, which mitigates forgetting, encourages diversification between experts, and causes no computational overhead during the training.

2 State of the art

The most straightforward approach to combat catastrophic forgetting in continual incremental learning (CIL) involves retaining exemplars. Nevertheless, storing exemplars may not always be feasible. In such cases, the most challenging scenario arises in exemplar-free CIL, where various methods are available. Many of these methods prioritize stability and address forgetting by employing different forms of regularization on a feature extractor that is already performing well. Some strategies even focus solely on incrementally learning the classifier while maintaining the backbone network in a fixed state. However, this approach of keeping the backbone network frozen may restrict its adaptability and prove insufficient for more intricate scenarios, such as when tasks are unrelated.

3 Objectives

The main objective of the project is to provide a comparison of how the Resnet family (18, 50, 101) behaves in the Class-incremental setting, underling its capabilities of learning incrementally and also with difficult datasets. In particular, the focus of the research will be to measure the difference between the baseline (also known as joint) model, namely to train on all images and classes all at once, and the SEED model, the CIL technique we chose. Our code implementation can be found at: `https://github.com/davideberga/DivideAndNotForget`.

4 Dataset Selection

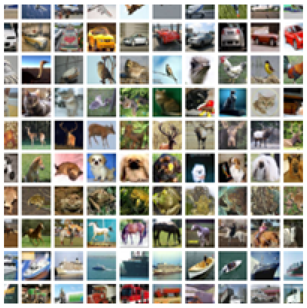


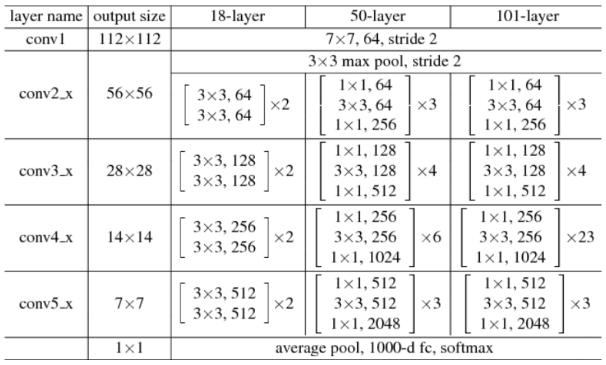
Figure 1: CIFAR100

Figure 2: Food101

The datasets chosen for the project are CIFAR100 and Food101. The choice has been driven mainly by the number of classes: with this amount of categories, it is possible to perform a more accurate survey on the class incremental setting. The CIFAR100 dataset is composed of 100 classes containing 600 images each of 32x32 size. There are 500 training images and 100 testing images per class. It contains various images like animals, flowers, house devices, people, and vehicles. The Food101 dataset contains 101 classes of different food dishes, for each one, there are 750 training images and 250 testing images, and each image is resized to have 256x256 size.
CIFAR100 is a good starting point because it is a commonly used benchmark dataset in the field of Continual Learning; on the other hand, Food101 brings us a domain where the classes are difficult to distinguish, specifically in this type of setting where the networks do not see them at the same moment.

5 Methodology

5.1 ResNet



layer name	output size	18-layer	50-layer	101-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2		
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax		

Figure 3: ResNet18, ResNet50 and ResNet101 schemas

The ResNet neural network is an architecture that presents different convolutional layers with skipping connections, batch normalization, and ReLU activation functions.
A convolutional layer is a layer used to change the dimensionality of input. A kernel, a padding, and a stride characterize it. The kernel is a mask that is applied to the input, with a step size specified by the stride and the padding tells us if, at the input, some extra values on the borders are added.

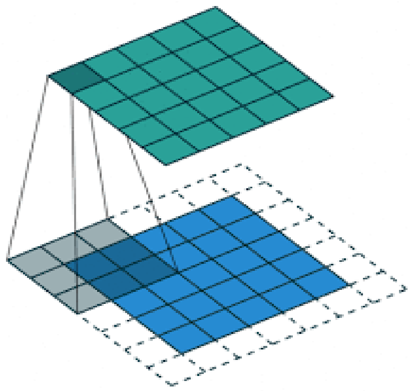


Figure 4: Convolution

In the ResNet architecture, even a max pooling and an average pooling are added. They simply unified more of the input pixel in one unique, either selecting the max (max pooling) or making an average of them (average pooling).

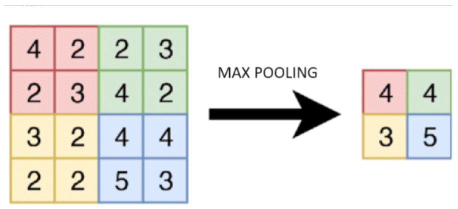


Figure 5: Max pooling

The idea of using skipping connections is to let layers learn residual functions with respect to the layer inputs. Batch normalization is an operation that improves the speed of the train and ensures the stabilization of the weights learned. It consists of computing the mean and variance of each batch and using them to normalize the data; after that, scale and bias terms are learned to allow the network to still use the non-linearity.

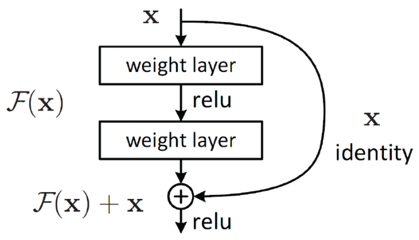

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 6: Residual skip connection

After the ResNet part, there is a fully connected block that is used to perform the classification. In our implementation, following the paper indication, this FC part is used only to compute the Cross-Entropy error on prediction, so it is trained, but not used in inference time. The just extracted features from the ResNet backbone are used to create Gaussian distributions given the input. The ReLU activation functions are used after the convolutional layers, which given an input, output the maximum value between the input and 0, so generally it removes the negative values. To allow the Gaussian distribution creation, the last ReLU activation function is omitted, because it can be composed of negative numbers too.

5.2 Class-Incremental Learning (CIL)

Class-Incremental Learning[2] aims to learn from an evolutive stream of i.i.d data with incoming new classes.
The key ingredients for CIL are;

1. A sequence of T training tasks $\{\mathcal{D}^1, ..., \mathcal{D}^{T-1}, \mathcal{D}^T\}$ without overlapping classes
2. $\mathcal{D}^t = \{(x_i^t, y_i^t)\}_{i=1}^{n_t}$ is the t-th incremental step, corresponding to the t-th task which has n_t training samples
3. $y_i^t \in Y_t$, where Y_t is the class space of task t
4. $Y_t \cap Y_{t'} = \emptyset \ \forall t \neq t'$, no overlapping classes in different tasks.
5. A model, during training of task t, can only access data from $\mathcal{D}^t$
6. After each task, the trained model is evaluated over all seen classes $\mathcal{Y}_b = Y_1 \cup ... \cup Y_t$

The primary objective of Continual Incremental Learning (CIL) is to consistently develop a classification model encompassing all classes. This model must not only assimilate knowledge from the present task, denoted as $\mathcal{D}^t$, but also retain insights from previous tasks. Following the learning phase of each task, the model undergoes evaluation across all encountered classes. An optimal model is expected to excel in newly introduced classes while retaining memory of past ones without experiencing catastrophic forgetting.

5.3 SEED

In this project, a novel ensembling method is used to overcome the classical problems of Class

Incremental Learning and improve the *plasticity* and *stability* of the model used. The *plasticity* is the ability to learn new features and adapt to this constant change continually; the *stability* is the ability to face the problem of catastrophic forgetting of features already learned. We chose SEED because of:

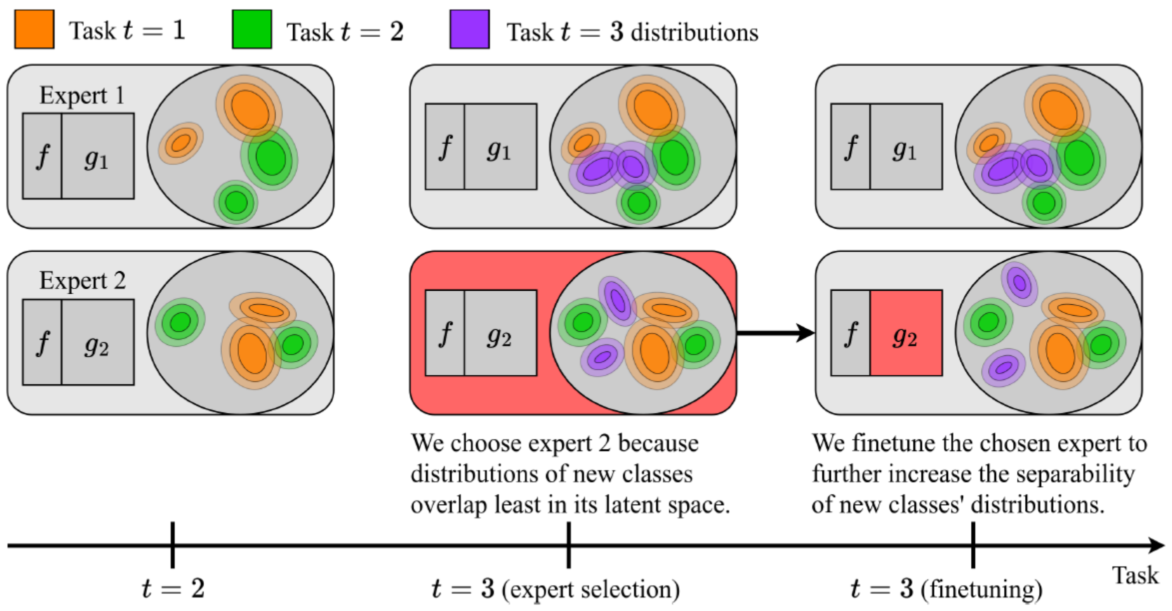
1. its intrinsic simplicity;
2. the training of a single expert per task mitigates forgetting but, at the same time, does not bring computational overhead concerning a naive finetuning method;
3. its impressive performance;
4. its portability: during our experiments, we chose to use the ResNet family as backbone for the experts, although SEED does not require it. It is possible to use every CNN architecture that can act as a feature extractor.

The core idea of the approach is to directly diversify experts by training them on different tasks and combining their knowledge during the inference.

5.3.1 Experts

SEED requires a fixed number of experts K that will constitute the ensemble. Each expert contains two components: a feature extractor $(g_k \circ f)$ that generates a unique latent space and a set of Gaussian distributions $(G_k^c = (\mu_k^c, \Sigma_k^c))$ (one per class and built in the latent space created by $(g_k \circ f)$. Each expert is trained on a different task causing disparities in the experts' embeddings and consequently different overlapping of class distributions. SEED takes advantage of this diversity, considering it both during training and inference. During the training phase, the class distributions are not taken into account: a linear layer is attached to the expert backbone, to be able to build a valid loss function and backpropagate the error. The authors tested the sharing of layers between experts, however, they discovered a significant performance drop increasing this number between them, so our experiments are based on experts without layers shared.

5.3.2 Training in deep



🟧 Task $t = 1$
🟩 Task $t = 2$
🟪 Task $t = 3$ distributions
Expert 1
f
g_1
Expert 2
g_2
We choose expert 2 because distributions of new classes overlap least in its latent space.
We finetune the chosen expert to further increase the separability of new classes' distributions.
$t = 2$
$t = 3$ (expert selection)
$t = 3$ (finetuning)
Task

Figure 7: Training SEED

Training assumes, as described in Section 5.2, T tasks, each corresponding to the non-overlapping set of classes $Y = Y_1 \cup Y_2 \cup ... \cup Y_t$ such that $Y_t \cap Y_{t'} = \emptyset \; \forall t \neq t'$. The objective is to train a model performing well both for classes of a new task and classes of previously learned tasks ($< t$). To do so the the training split of the dataset is divided into T splits, based on classes. The number of tasks and the number of classes in the first class are both parameters: the first task is a special case explained below and typically has more classes than the following tasks.

Expert finetuning and distribution creation As said earlier SEED is based on finetuning a single expert on a specific task. Once the expert $\bar{k}$ is selected for a task t, a linear head is attached to its backbone $g_k \circ f$ and the training starts on the task subset of data $\mathcal{D}^t$. As a loss function, we use cross-entropy combined with feature regularization weighted with α:

$$L = (1 - \alpha)L_{CE} + \alpha L_{KD}$$

where

$$L_{CE} = -\sum_{c=1}^{C} y_{i,c} \log(p_{i,c})$$

$$L_{KD} = \frac{1}{B}\sum_{i \in B} ||g_{\bar{k}} \circ f(x_i) - g_{\bar{k}}^{old} \circ f(x_i)||$$

, B is a batch and $g_{\bar{k}}^{old}$ is a frozen $g_{\bar{k}}$. Just after this phase, the head is removed. At this point for all experts the distributions of the new classes Y_t are computed and added to G_k. This is done in the following way for a new class c:

1. let $\mathcal{D}_c^t$: the training subset of task t and only sample of class c;

2. computes the forward pass on all the experts' backbone with $\mathcal{D}_c^t$ producing features;

3. fit K different Gaussian distribution on the K sets of features just extracted and add them to the corresponding expert.
4. repeat $\forall c \in Y_t$

At the end of this process, only one expert is actually fine-tuned on the new classes Y_t, however, every other expert has a distribution that approximates them.

Expert selection Expert selection starts with determining the distribution for each new class $c \in Y_t$ in each expert k. For this purpose, we pass all x from $\mathcal{D}_t$ with $y = c$ through deep network $g_k \circ f$. This results in a set of vectors in latent space for which we approximate a new multivariate Gaussian distribution $q_{c,k}$, thus each expert is associated with a set $Q_k = \{q_{1,k}, q_{2,k}, ..., q_{|C_t|,k}\}$ where $C_t = Y_1 \cup Y_2 \cup ... \cup Y_t$ is the set of classes seen so far, until task t. The expert selected $\overline{k}$ is the one for which those distributions overlap the least, or, equivalently, that maximize the Kullback-Leibler divergence d_{KL}:

$$\overline{k} = \underset{k}{\operatorname{argmax}} \sum_{q_{i,k}, q_{j,k} \in Q_k} d_{KL}(q_{i,k}, q_{j,k})$$

Task 1 As just described the first task has more classes w.r.t. the other tasks. The backbone $g_1 \circ f$ of the first expert is finetuned on this task and then the other experts will use its weights as initial ones.

First K tasks Since the expert selection is based on minimizing the overlapping between class distributions belonging to each of them and since the expert initialization does not guarantee a meaningful distribution creation: the expert selection is skipped for the first K tasks. The expert selected is simply the one with the same number as the number task ($k = t$). During the first K tasks the loss function is only the cross-entropy, $L = L_{CE}$

Last $T - K$ tasks In these tasks, the expert selection is done as described in the paragraph above 'Expert selection'. The loss function is the complete one: $L = (1 - \alpha)L_{CE} + \alpha L_{KD}$.

```python

def train_loop(self, t, trn_loader, val_loader):

    train_loss, valid_loss, train_acc, val_acc =
        0, 0, 0, 0

    if t < self.max_experts:
        print(f"Training backbone on task {t}:")
        train_loss, valid_loss, train_acc, val_acc =
            self.train_backbone(t, trn_loader, val_loader)
        self.experts_distributions.append([])

    if t >= self.max_experts:
        bb_to_finetune =
            self._choose_backbone_to_finetune(t,
                trn_loader, val_loader)
        print(f"Finetuning backbone {bb_to_finetune} on
            task {t}:")
        old_model, train_loss, valid_loss, train_acc,
            val_acc = self.finetune_backbone(t,
            bb_to_finetune, trn_loader, val_loader)

    print(f"Creating distributions for task {t}:")
    self.create_distributions(t, trn_loader, val_loader)
    return train_loss, valid_loss, train_acc, val_acc
```

Listing 1: Train loop code

5.3.3 Inference phase

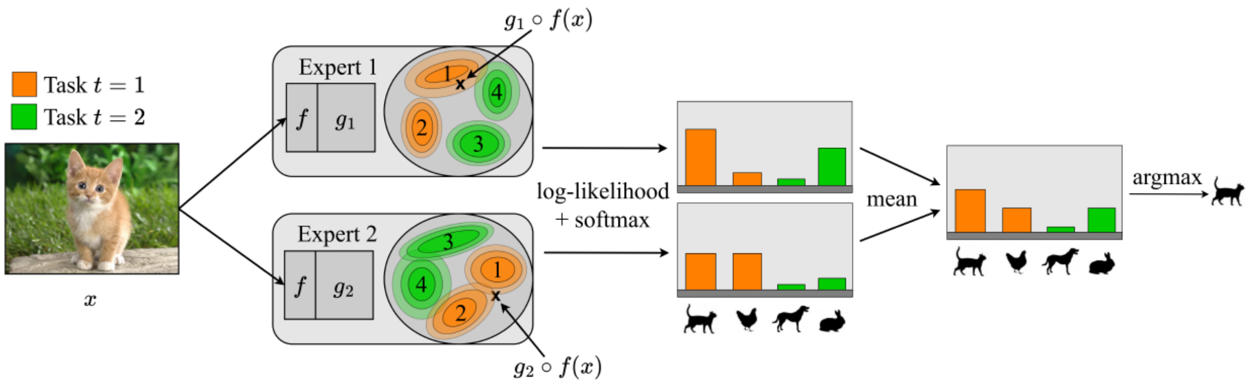


Figure 8: Inference SEED

During inference, we perform an ensemble of Bayes classifiers. Firstly, we generate representations of input x for each expert k as $r_k = g_k \circ f(x)$. Secondly, we calculate log-likelihoods of r_k for all distributions $G_{c,k}$ associated with this expert. Then, we softmax those log-likelihoods and compute their average overall experts. The class with the highest average softmax is considered as the prediction.

6 Experiments and Results

We performed 12 experiments using 3 different ResNet architectures: ResNet18, ResNet50, and ResNet101 for two types of datasets: CIFAR100 and FOOD101. We then compared two methods: baseline and seed.

- **Baseline setup**: training 200 epochs for food101 and 500 for cifar100, with 1 expert. This is to emulate a training without the class incremental setting.
- **Seed setup**: training 20 epochs per task for food101 and 50 for cifar100, with 5 experts.

Parameters:

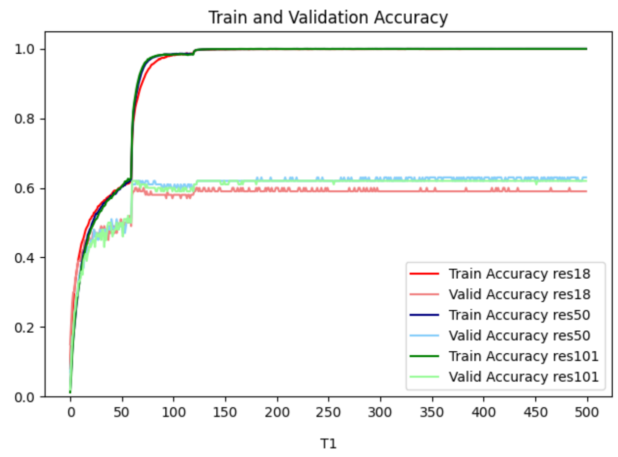
- **Number of tasks**: 10
- **Number of classes in the first task**: 20
- **Batch size cifar100**: 128
- **Batch size food101**: 64
- α: 0.99
- **momentum**: 0.9

Evaluation metrics and plots used:

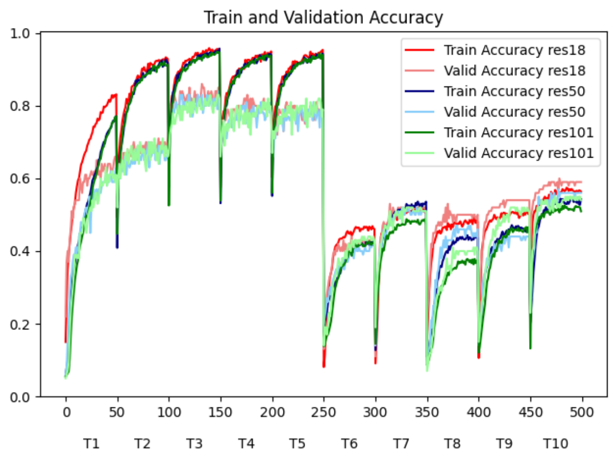
- **Train/Validation Loss/Accuracy**
- **Test accuracy aware and accuracy agnostic**: in the task agnostic setup all classes are considered during the softmax/average over all experts procedure, in the task aware one only classes for the selected task are taken into account.
- **Confusion matrices**: these also in both versions, agnostic and aware for the SEED method
- **Forgetting table**: table that summarizes the percentage of forgetting, computed as follows: $\max(acc_{M_{j,i}}) - acc_{M_{t,i}}$ where $M_{t,i}$ is the model trained after task t and tested on task i, and $max(acc_{j,i})$ is the best accuracy achieved on task i considering all models j $\forall j \in \{0, ..., t-1\}$

6.1 CIFAR100

6.1.1 Train and Validation Accuracy



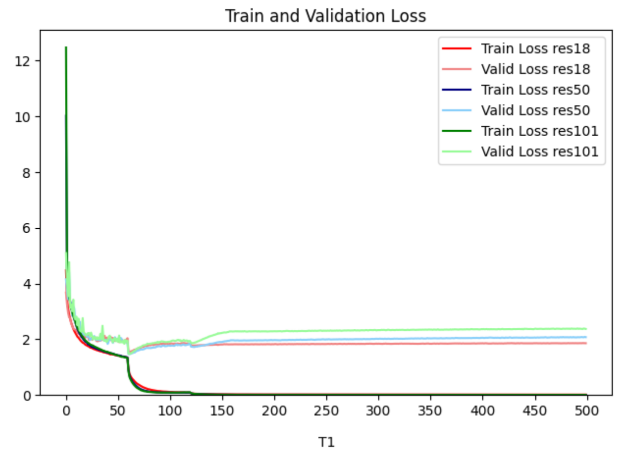
(a) Train and validation accuracy baseline



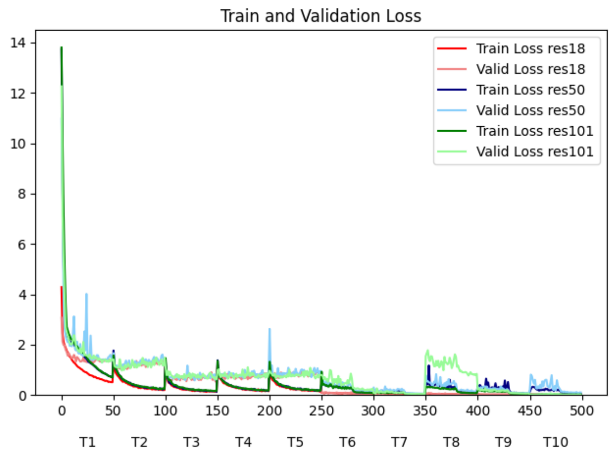
(b) Train and validation accuracy SEED

Figure 9: Train and validation accuracies

6.1.2 Train and Validation Loss



(a) Train and validation loss baseline



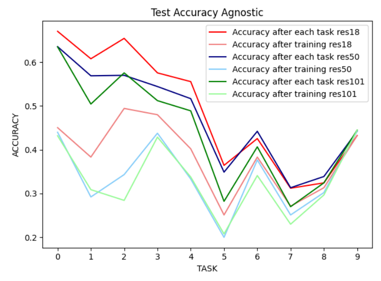
(b) Train and validation loss SEED

Figure 10: Train and validation losses

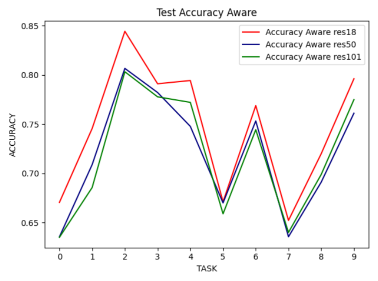
6.1.3 Test

	ResNet18	ResNet50	ResNet101
Test accuracy	0.6032	**0.6269**	0.6243

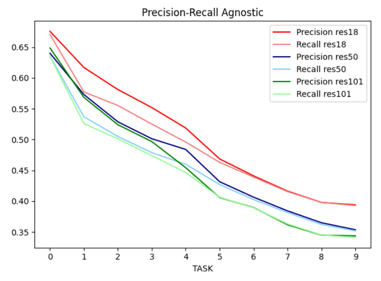
Table 1: Test accuracy baseline



(a) Test accuracy agnostic SEED



(b) Test accuracy aware SEED



(c) Precision-Recall SEED

Figure 11: Test accuracy agnostic, accuracy aware, precision-recall SEED

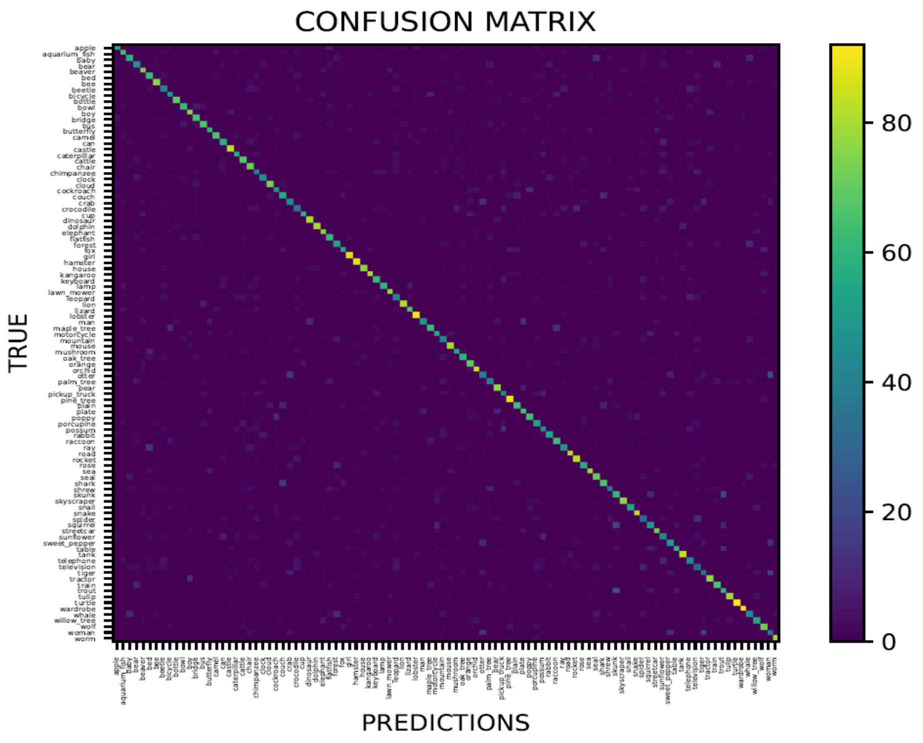
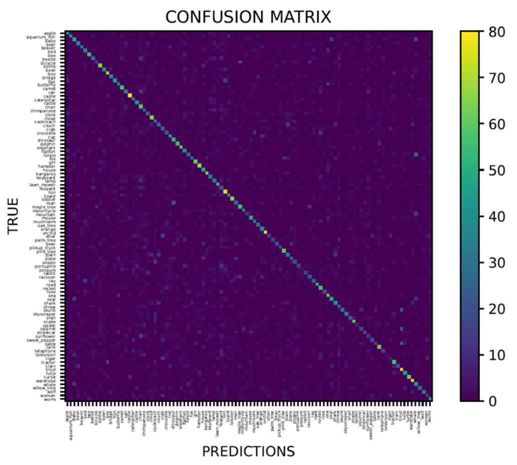
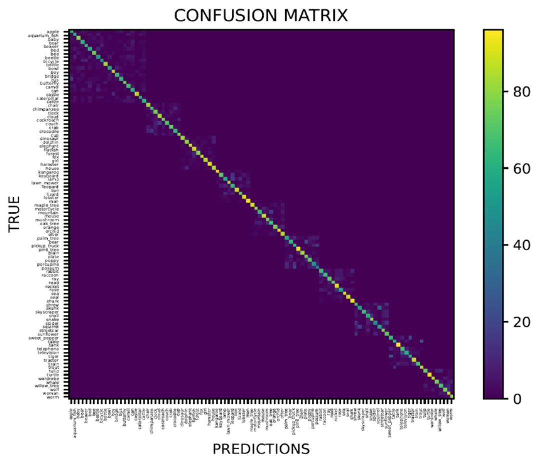


Figure 12: Confusion Matrix baseline with ResNet50



(a) Confusion Matrix agnostic



(b) Confusion Matrix aware

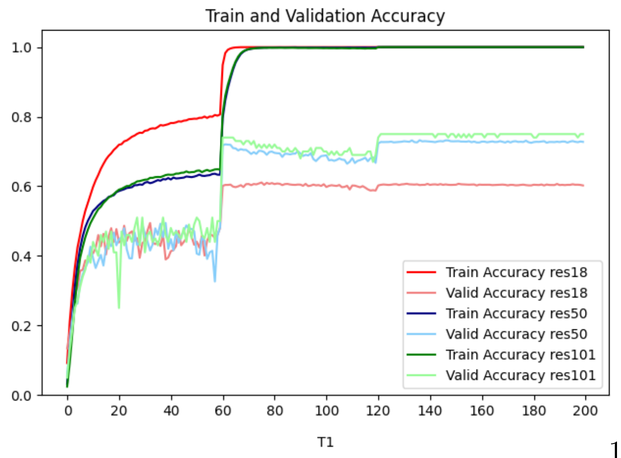
Figure 13: Confusion Matrix agnostic and aware SEED with ResNet18

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
M1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M2	10.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M3	14.2%	9.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M4	16.5%	13.3%	8.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M5	18.6%	19.8%	13.0%	5.6%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M6	18.8%	22.1%	14.7%	8.2%	3.8%	0.0%	0.0%	0.0%	0.0%	0.0%
M7	20.9%	23.6%	14.2%	8.6%	8.4%	5.1%	0.0%	0.0%	0.0%	0.0%
M8	21.3%	22.3%	14.4%	9.1%	10.2%	9.3%	2.6%	0.0%	0.0%	0.0%
M9	21.6%	22.4%	15.1%	9.1%	12.9%	10.2%	2.7%	3.7%	0.0%	0.0%
M10	22.0%	22.4%	16.0%	9.6%	15.3%	11.3%	4.2%	4.1%	1.1%	0.0%

Table 2: Forgetting SEED ResNet18

6.2 FOOD101

6.2.1 Train and Validation Accuracy



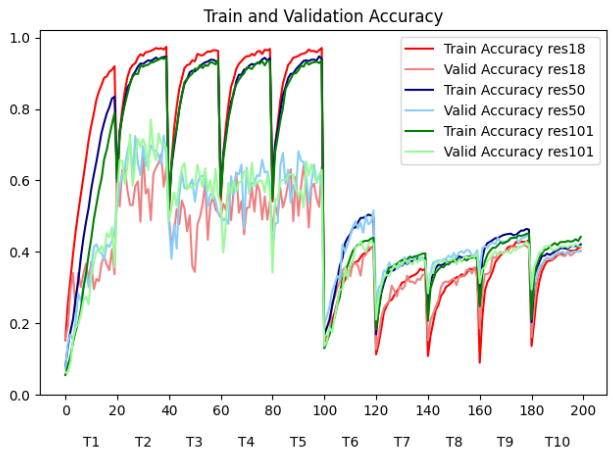
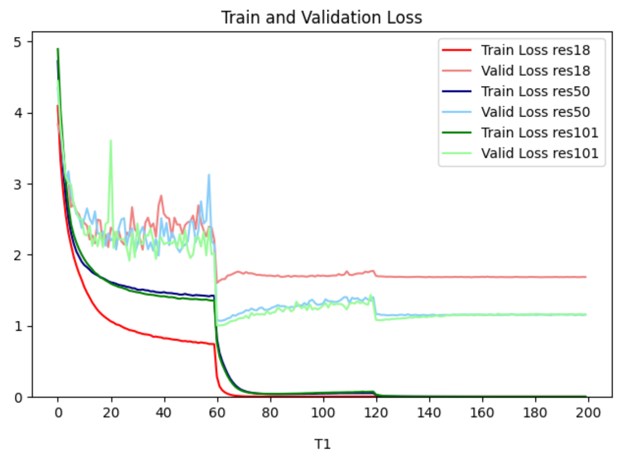


Figure 14: Train and validation accuracy baseline

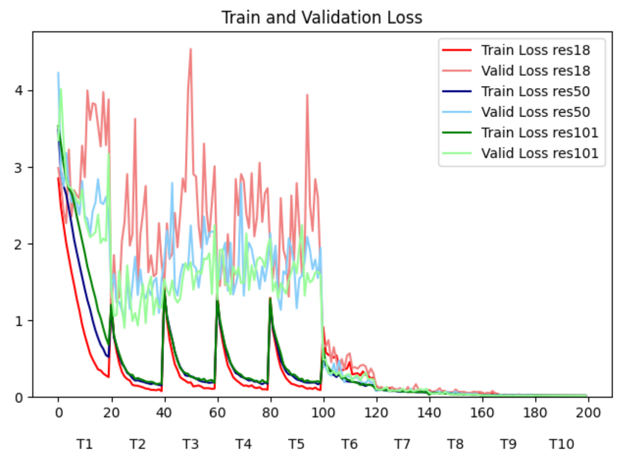
Figure 15: Train and validation accuracy SEED

Figure 16: Train and validation accuracies

6.2.2 Train and Validation Loss



(a) Train and validation loss baseline



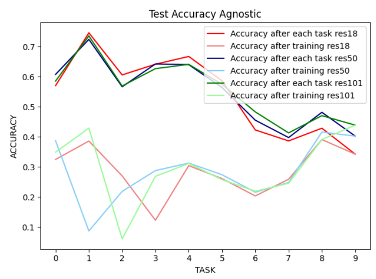
(b) Train and validation loss SEED

Figure 17: Train and validation losses

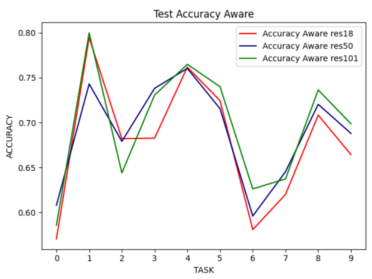
6.2.3 Test

	ResNet18	ResNet50	ResNet101
Test accuracy	0.655287	0.779762	**0.801703**

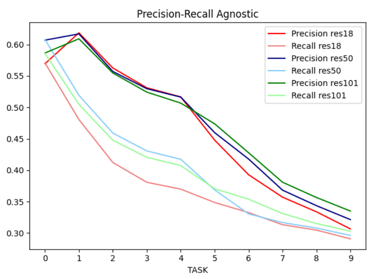
Table 3: Test accuracy baseline



(a) Test accuracy agnostic SEED



(b) Test accuracy aware SEED



(c) Precision-Recall SEED

Figure 18: Test accuracy agnostic, accuracy aware, precision-recall SEED

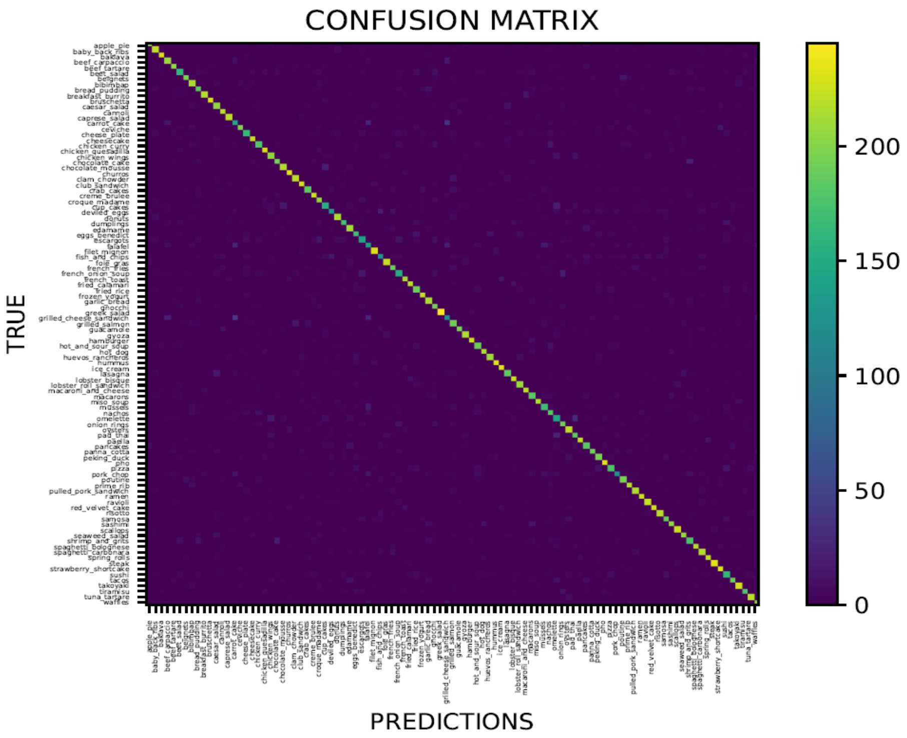


Figure 19: Confusion Matrix baseline with ResNet101

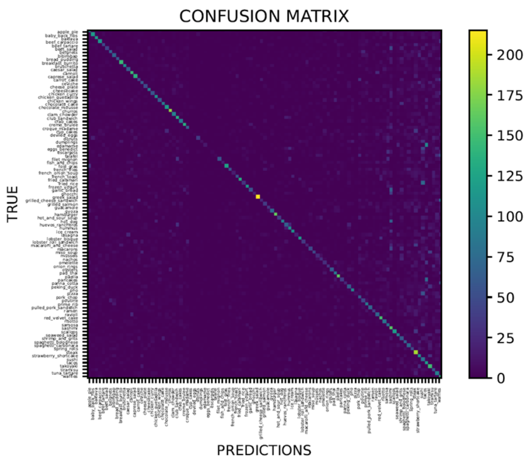


Figure 20: Confusion Matrix agnostic

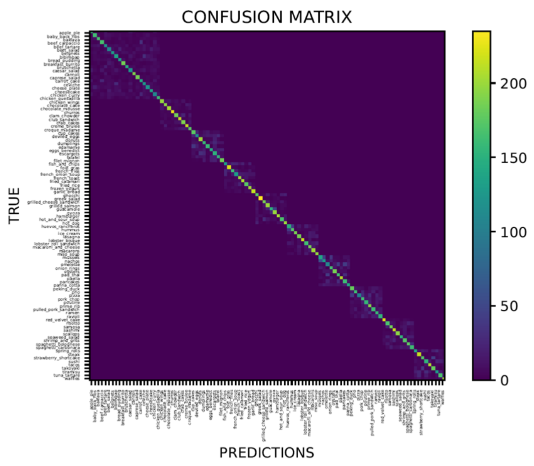


Figure 21: Confusion Matrix aware

Figure 22: Confusion Matrix agnostic and aware SEED with ResNet101

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
M1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M2	18.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M3	21.9%	23.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M4	23.8%	30.4%	20.6%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M5	26.2%	31.8%	26.0%	18.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
M6	24.2%	32.0%	43.6%	29.1%	19.1%	0.0%	0.0%	0.0%	0.0%	0.0%
M7	22.3%	31.6%	48.7%	30.7%	25.1%	16.8%	0.0%	0.0%	0.0%	0.0%
M8	23.2%	30.7%	49.2%	32.3%	27.2%	24.0%	14.6%	0.0%	0.0%	0.0%
M9	23.5%	31.2%	50.7%	35.3%	30.7%	29.1%	20.6%	10.0%	0.0%	0.0%
M10	23.7%	30.7%	50.8%	35.8%	33.0%	31.7%	26.4%	16.8%	8.0%	0.0%

Table 4: Forgetting SEED ResNet101

7 Conclusion

After the results extraction, we compared ResNet18, ResNet50 and ResNet101 using baseline and SEED methods on CIFAR100 and FOOD101.

Train/Validation Accuracy In the baseline training is evident that the train accuracy is always over the validation accuracy. Instead in SEED training while the number of tasks is less than the number of experts we can notice that the accuracy in train is greater than validation accuracy; but when the number of tasks overcomes the number of experts, the accuracy in training and validation seems to be similar grouped by ResNet architecture. This behavior can underlie an ability depending on architecture to classify images of different tasks. It's further interesting the little drop that we have between one task and another, this happens because the classes change and so for each task, the accuracy starts from a low value and reaches a higher one at the end of the task training.

Train/Validation Loss The loss in the baseline training is decreasing for all architectures in both datasets. Instead, the loss in SEED environment has a particular decrease after the moment the number of tasks is major to the number of experts, this is because from this point a different loss is used in training, which allows a regularization and so we can notice how the value for the loss is little than the first tasks. On the contrary in this case, between a task and another, there is a jump in loss, because the classes that are presented are changed.

Test Accuracy & Precision-Recall For the models produced in training on CIFAR100 the best baseline model is ResNet50, and the more accurate model with SEED is ResNet18. Whereas the models created in training FOOD101 state that the best architecture for baseline and SEED approach is ResNet101. The absolute higher performance found was on the dataset FOOD101. The precision and recall metrics computed in SEED confirm the decreasing performance that is present in CIL approaches, where classes are not reproposed in different tasks.

Confusion Matrices All the confusion matrices presented are related to the best models that were found in baseline and SEED. The baseline confusion matrices indicate for both datasets a good ability to predict that true class. The confusion matrices agnostic, focusing on the last learned classes, on CIFAR100 seem to predict all classes, instead, FOOD101 predicts majorly the just learned classes. The confusion matrices aware underlie the different task splits that are presented during the training, and the predictions are of course more precise because they are performed only on just learned classes.

Forgetting The forgetting tables show for the best ResNet architectures, how much each new model produced is forgetting the classes presented in the previous tasks. In the last row of the table is evident that more tasks are added, more the start tasks are forgotten. This behavior is normal because we used an exemplar-free approach.

References

[1] Grzegorz Rypeść et al. *Divide and not forget: Ensemble of selectively trained experts in Continual Learning*. 2023.

[2] Da-Wei Zhou et al. *Deep Class-Incremental Learning: A Survey*. 2023. arXiv: `2302.03648 [cs.CV]`.