A Procedural World Generation Framework for Systematic Evaluation of Continual Learning: Supplementary Material

Timm Hess, Martin Mundt, Iuliia Pliushch, Visvanathan Ramesh
Goethe University, Frankfurt am Main, Germany
hess@ccc.cs.uni-frankfurt.de
{mmundt, pliushch, vramesh}@em.uni-frankfurt.de

Content Overview

The supplementary material contains further details for the content presented in the main body. The first two sections provide additional information on the simulator, spanning its 3D assets and configuration interface for the user, as well as its limitations. Thereafter, datasets generated for the classification tasks are discussed in more detail, and we elaborate on our conducted experimental training details, hyper-parameters and used neural network architectures. In the ultimate section of the supplementary material, we provide a description and intuition behind the photometric color invariance operators, as used in the main body to demonstrate the severity of the shortcomings of the deep continual learning algorithms in the progressive lighting experiments. In summary, the overall structure of the supplementary material is as follows:

A. Additional simulator details

B. Limitations of the simulator

C. Dataset generation for classification

D. Deep neural network training hyper-parameters

E. Operators for quasi-invariance to illumination

Please note that further detailed readme files, containing extra instructions on installation for our software contributions, as pointed out in the main body, can be found in the respectively linked repositories.

A Additional Simulator Details

In section 2 of the main body we have introduced our key contribution of a computer graphics simulation framework for the flexible composition of data streams that facilitate assessment of continual learning. There, we have detailed the underlying modular generative model and its random variables. Recall, that a majority of objects and actors are placed stochastically and their appearance is modelled largely through categorical variables. In this supplementary section, we further illustrate the currently available 3-D assets that are subject to this categorical sampling, their placement in the scene, and the overall possibilities of configuration for the simulator to re-emphasize its flexibility in creating specifically customized scenarios in the context of continual learning.

Sampled assets: In the main body, we have already described E_i, a convenience variable which defines whether a certain object or actor category i, for example a tree or a human, is sampled into the generated scene. Depending on the result or the user's choice, a spawned object or actor is then subject to categorical sampling, followed by stochastic placement on a street tile. This categorical

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

Figure 1: Overview of 3-D assets used in the current implementation of our simulator, spanning 6 street segment layouts, 10 buildings, 19 human actors, 7 trees, 1 street lamp, and 2 cars with corresponding 3 and 4 colorings.

sampling represents a simple process of randomly selecting a 3-D asset from the available repository. Recall, that for each street tile this categorical sampling is repeated in order to place multiple objects or actors of the same type within the bounds of each randomly selected street segment. As such, controlling the probabilities on the categorical variables and extending or limiting the used repertoire of 3-D assets directly impacts the diversity of possibly generated scenes.

Figure 1 illustrates the assets that have currently been included in our simulator for the presented experiments. The figure shows the six available street segment types, representing right and left curves, a continuing straight road and three choices for crossings. The discretized set of buildings and trees can be observed to have been picked to balance variations in width and height. Vehicles vary in color and car model. Pedestrians were selected with different body type, skin color, age and gender in mind. The presently used amount of distinct 3-D assets is made up of 6 street segment layouts, 10 buildings, 7 trees, one street lamp, 2 cars with respective 3 and 4 colors, and 19 human actors. These assets where mostly generated by hand, using editors such as *Blender*[1], *TreeIt*[2], *MakeHuman*[3] and modular asset packs provided by the Unreal Engine store [4].

The heterogeneity of the scenes is thus presently controlled solely through the set of available assets and their categorical sampling probabilities. In order to allow for more nuanced control, it is however imaginable to further decompose these categorical variables into separate complementary variables that manage e.g. the geometry of a car and its color independently. For our current investigative CL experiment purposes, the purely categorical sampling has been observed to be sufficient and further extensions are left to future work.

Sampled positioning: Sampling the position of an object upon placing it into the scene is implemented by a finite support uniform distribution, with its bounds resembling intuitive assumptions on real-world statistics, such as trees not being planted in the middle of a road. In figure 2 we show a schematic illustration of the bounds for each object category's placement for the example of a straight street segment. Object-spawning areas are depicted as colored bounding boxes, which indicate the limits for the uniform sampling of each object's placement in the tile's space with respect to its category. These areas are individually defined for each tile layout, however, all bounds' definitions follow the same structure of design, as is illustrated in the figure. Providing each object category its own, non-overlapping, space on the tile is a deliberate choice to reduce the complexity to prevent objects of different categories from overlapping or otherwise colliding at runtime. Collision between objects of the same category, e.g. humans being placed into one-another, are initially avoided by the spawn routine itself checking for overlaps when placing an object into the scene. For dynamic actors collision avoidance is handled by their behavioral implementation.

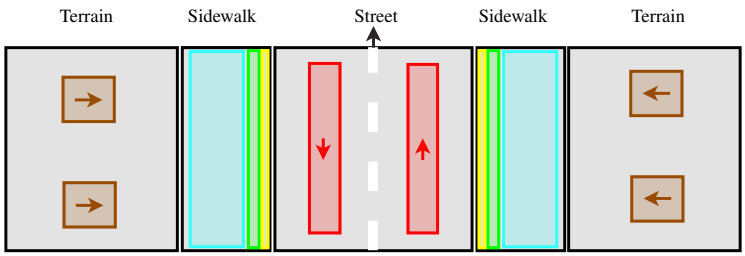

Figure 2: Schematic illustration of the object placement for the example of a straight street segment. The bounded areas for each object category are color coded as follows: vehicles (red), streetlamps (yellow), trees (green), pedestrians (cyan), buildings (brown). Arrow-markers inside these areas indicate an additional direction specification in which spawning objects should face. This way, vehicles do not drive against their lane direction, and building-fronts face the street. The black arrow-marker at the top of the tile, indicates the spawn anchor for the next sampled tile's attachment.

Configuration: Simulator configuration by the user is a straightforward process on the basis of the generative model for video sub-sequences, as defined in section 2.1 of the main body. Being able to manipulate the parameters of this model, the user is provided with the freedom to define arbitrary chains of sub-sequences in the generated video stream. We re-emphasize that this translates to a vast number of scenarios to be generated with nuanced control over abrupt and/or continuous distribution changes. In the executable version of the simulator, provided with this work, all configuration of the sequences to be constructed can be specified prior to actually running the data generation process. The configuration itself is interfaced by JSON-config files. Example files containing the configurations to the data generation for all experimentation conducted in the main body are part of the repository containing our simulator.

For the following further explanation of the configuration parameters we proceed with the Unreal Engine editor's view of a sequence's configuration. A corresponding visualization is depicted in figure 3. We note that the presented visualization of a sequence configuration is automatically generated by the Engine's-Editor from our data-structure holding the configured parameters.

Overall, the configuration of a sequence can be understood as being divided into two major parts. The first part defines parameters that are applied to the entire stream. These stay unchanged for all sub-sequences and include the camera's settings θ_C, such as the frames-per-second and resolution of the rendered output and the render-modes to be considered. Additional included parameters are

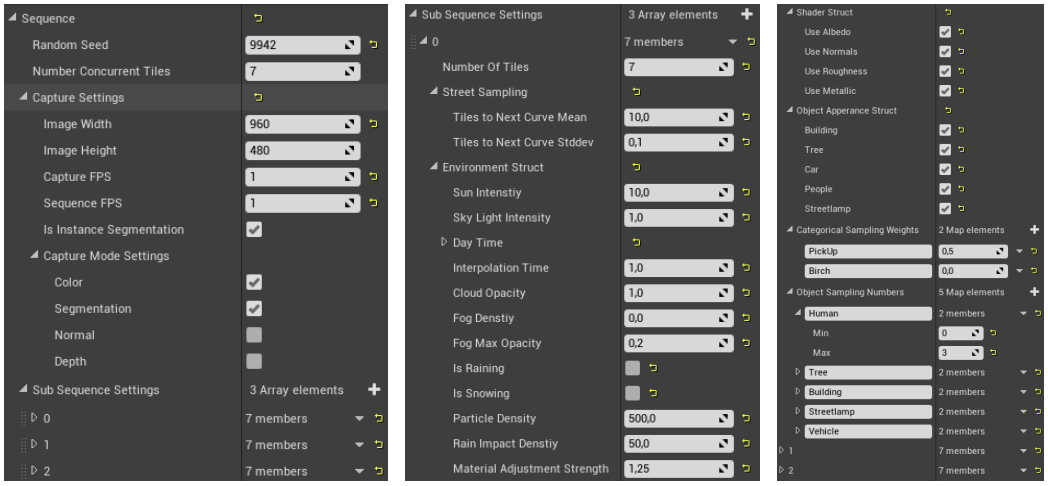

Figure 3: Overview of a sequence's parameter configuration as visualized by the Unreal Engine editor. Left: parameter settings that are applied to the entire stream, including rendering options and random-seed. Center and right: parameters settings of an example individual sub-sequence, outlining the range of configuration possibilities.

a general random-seed that is the basis for all sampling in the sequence, and the number of tiles to make up the main street track at any time during the simulation.
The second part covers all remaining detailed parameters to provide the specification for each individual sub-sequences. The length of a sub-sequence is specified by the 'Number of Tiles' to be sampled. The 'Street Sampling' covers θ_S, controlling the curvature in the street track. The 'Environment Settings' are composed of θ_L and θ_W, manipulating lighting and weather related variables. The 'Material Settings' control the shading options, equivalent to θ_R. The 'Object Presence' governs parameters θ_E for the Bernoulli variables.
Finally, the 'Categorical Sampling Weights' used for the conditional selection of specific objects can be adjusted, which correspond to θ_H, θ_V, θ_{Tr}, θLp, θ_B. Similarly, 'Object Sampling Numbers' control the number of said objects $N_H, N_V, N_{TR}, N_{Lp}, N_B$ to be sampled for each tile. The categorical sampling weights are specified by mapping identifiers, i.e. string-typed names, to probability values. These correspond to names in the directory structure, which is used to store the respective object-assets inside the simulator. This way, sampling probabilities can be assigned to individual object instances, e.g. one specific birch tree, or alternatively, to entire sub-categories, e.g. all birches. The exact directory structure is provided to the user as part of the repository of this work.
A similar specification process is used for the object numbers. However, here only top level identifiers, that correspond to respective object categories, e.g. tree, car, or people, are considered.

To lift potential confusion regarding the interplay of 'Object Presence' settings (parameters of θ_E), 'Object Sampling Numbers', and 'Categorical Weights', we briefly recapitulate their relations: as already touched upon in the main body, the 'Object Presence' settings are convenience parameters, to easily allow or deny sampling of certain object categories. This design has been chosen in light of often applied incremental class continual learning scenarios. The same effect could be achieved by setting 'Object Sampling Numbers' for the respective category to a value of 0. However, once the 'Object Presence' for a given object is turned off, further 'Object Sampling Numbers' are ignored. Regarding the categorical weights in this context, we want to highlight that these are understood as being conditioned on the event of an object being sampled. Setting an object-category identifier to zero, or likewise all identifiers of this particular category to zero individually, will fall back to consistently sampling the first asset entry of that category.

B Simulator Limitations

Building on the previously described extended information on assets and sequence configuration, we discuss the limitations of the provided simulator in its current state. As we pointed out, the heterogeneity of the scenes to be generated is eventually dependent on the diversity of the 3-D assets available for sampling. This is because the restriction to the used asset repository naturally affects the number of achievable, truly distinct, tile configurations. In principle, the data generation may 'endlessly' sample a continuous world, which is in contrast to static maps that can be internalized over time. However, there is inevitably going to be a re-occurrence of some previously observed patterns after the repertoire has been exhausted in all possible configurations (which we re-emphasize is still a large number considering randomness in locations, environmental factors etc.) Towards the ultimate goal of transferring models trained in virtual environments to applications in the real-world, which is the most common use-case of synthetic data generation, the total number of distinct possible patterns to be observed should be maximized and likewise, the distance to matching real-world statistics minimized. This issue is presently inherent to any graphics simulation for computer vision and has seen multiple solution attempts, as elaborated in the main body's related work section. At this point, adaptation from our simulator to real remains a desideratum. Even though this limitation is recognized, we re-emphasize that our main body's experimentation suggests that there nevertheless is large merit in the generated datasets as benchmarks for continual learning.

One imaginable way to easily overcome the asset limitation is through the inclusion of further publicly provided assets. Online repositories for the latter are fortunately growing at a rapid pace. However, being publicly accessibly and both non-commercially and commercially distributable are two distinct factors. Most commonly used licensing unfortunately prohibits re-distribution by any third-party, particularly if assets require a prior purchase. This includes some of the above mentioned assets, where we have referenced third party software respectively. For our open-source contribution we have thus opted for a two-fold distribution of our work: the source code of our simulator, where various assets have been removed, but the core code mechanisms are accessible and can be build upon.

We believe that reintegration of assets and inclusion of new ones can be managed with reasonable effort. At the same time, for the continual learning practitioner and investigator, we also distribute our simulator in a self-contained and encrypted executable, as encouraged by Unreal Engine's market place policy. Although the assets cannot exchanged in this version, it provides the user with the necessary means to set the generative process' parameters through configuration files, as described in the previous section.

C Dataset Generation for Classification

An empirical analysis of multiple continual learning scenarios has been shown in the experimental section of the main body. There, we have argued that the simplest conceivable assessment is a classification setup. The respective empirical results have indicated that this already provides a significant challenge in consideration of the even more complex semantic segmentation, surface normal or depth prediction tasks. In this supplementary material section we first provide additional detail on general parameter choices for the generation of our video streams, illustrate how they can almost trivially be used for classification purposes, and then proceed to further specifying the precisely constructed datasets. Before continuing, we would at this point like to remind the reader that the constructed classification scenarios present but a tiny fraction of the imaginable applications of our proposed simulation framework, as thoroughly explained in the main body's discussion.

Parameter Choices: For our current purposes, the default parameters of all categorical variables in the generative model are set such that each of the M choices is equally likely:

$$P(Obj = Obj_m \,|\, \theta_{Obj}) = 1/M \quad \forall\, m = 1, \ldots, M\,. \tag{1}$$

The parameters governing the number of individual objects and actors is set in analogous fashion, with empirically set maximum number of 4 buildings, 8 humans, 2 additional cars, 4 street lamps and 6 trees per street segment. The number of tiles N_S is treated as deterministic and set to 7 at a time, as a practical trade-off to surpass the view distance of the employed camera. In order to prioritize multiple straight street segments before occurrence of a curve or crossing, the number of the next consecutive straight street segments is predetermined by drawing from a normal distribution $n_S \sim \mathcal{N}(4, 0.45)$ each time a straight street is spawned. The specific straight street layout for the individual segments is then determined by:

$$P(S = Straight_i \,|\theta_S) = (1/I), \forall i = 1, ..., I\,. \tag{2}$$

After n_S straights have been placed, a curve or crossing is spawned with equal sampling weights for all available layouts, resulting in their placement probabilities:

$$\begin{aligned} P(S = Curve_k \mid \theta_S) &= (1/K) \cdot 1/2 \;\; \forall\, k = 1, \ldots, K \\ P(S = Crossing_l \mid \theta_S) &= (1/L) \cdot 1/2 \;\; \forall\, l = 1, \ldots, L\;, \end{aligned} \tag{3}$$

where I, K, L indicate the respective number of available distinct street tiles. For an initial practical assessment of continual learning, only one straight tile of constant width and length, two choices for left and right turning curves and three variations for crossings have been included. We have visualized the collection of currently available object and actors in previous section's figure 1. These parameter settings are kept constant over time. In contrast, considered continuous learning scenarios can be characterized through a change of parameters over time. Whereas this could be nuanced modifications in above probabilities on specific object models or locations over time, we choose to investigate sequences with respect to variations in weather, lighting and the Bernoulli variables E for existence of entire object categories, mirroring class incremental scenarios. As an example, an incremental weather video sequence can be generated by initially sampling the categorical weather with equal probability, akin to equation 1, and subsequently setting the already sampled condition's probability to zero and raising the remaining choices' likelihoods to sum to unity. Alternatively, the probability for a desired categorical outcome can just be set to one. In the same spirit the existence of particular objects defined through Bernoulli variables $E_B, E_{Tr}, E_{Lp}, E_H, E_V$ can just be assigned probability vectors. For instance, short notation $\pi_{E,t=0} = (1, 0, 1, 0, 0)$ indicates the presence of buildings and lamps, and the absence of humans, additional vehicles and trees in a video sub-sequence. Exact considered set-ups are described in the experimental section.

Classification Dataset Construction: Given that our simulator already provides exact pixel-wise ground truths of the scene, construction of the classification tasks for our class incremental, incremental weather and incremental lighting scenario is but a matter of straightforward image patch extraction from video frames. In other words, given an object's bounding box only the content enclosed by its extent is presented to the neural network to learn a simple prediction with respect to the contained class. Intuitively, this can be thought of as being granted access to exact object positions in the video stream by an all-knowing oracle. An example video frame with depicted bounding boxes and their extracted contents is illustrated in figure 4. At a closer look, it can be observed that the bounding boxes are of a general rectangular format, with e.g. a street lamp always being much taller than its corresponding width. In contrast, the extracted image patches for the classification task always featuring a 1:1 aspect ratio. This represents a choice and further simplification in dataset construction, as a direct consequence of assuming a deep convolutional neural network that is designed for static quadratic spatial input dimensions, see supplementary part D for neural network details.

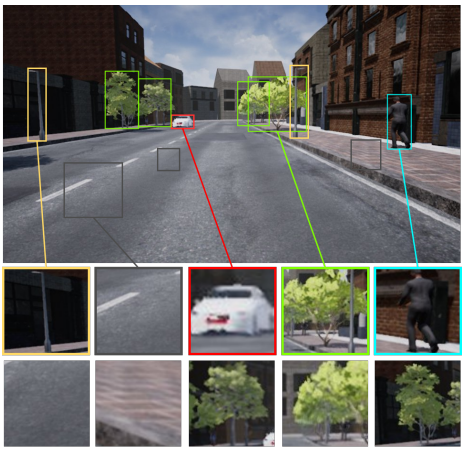

Figure 4: Example video frame illustrating the extraction of squared image-patches for a pure classification task. Given respective bounding boxes for each object instance, a quadratic crop is taken based on the larger of the two bounding box dimensions, such that the object is centered in the resulting patch. Example extracted image patches have been connected to their original bounding boxes to provide better intution for this process.

We can summarize the explicit extraction procedure from a full video frame to separate square image patches in a few key steps:

1. **Object bounding boxes:** Given the exact semantic segmentation pixel-wise ground-truths, a tight bounding box is directly acquired based on the furthest labelled pixels with respect to the height and width dimensions.
2. **Background bounding boxes:** For each video frame, four "background" bounding boxes are randomly sampled in order to include regions that are not explicitly included as a separate category in the classification task. To imitate the average extent of the bounding boxes for the other classes, background width and height are randomly sampled from a uniform distribution in a range of 64 to 200 pixels. To avoid conflicts between categories of interest and the background, a sampled background bounding box is rejected if it collides with an already existing object bounding box of the particular frame.
3. **Image patch crops:** Create a set of image patches for each video frame by taking crops of the bounding boxes. In our specific case of a later trained neural network that requires spatially symmetrical input, we take a square crop based on the larger of the bounding box's dimensions, such that the contained object is centered. In other words, a small amount of background is included for the shorter side of the bounding box. We do not at this point randomize the cropping to contain stochastic translations of the object, as is commonly done in training of datasets such as ImageNet [5] for data augmentation purposes.
4. **Consistency step:** For our main body's experiments we have chosen to include a further simplification step to facilitate the classification and assure consistency between extracted patches. Specifically, we discard any bounding boxes and resulting crops with a minimum width or height that does not surpass 32 pixels. That is we do not include objects that are far in the distance in our current assessment. We also guarantee that the classification task is strictly single target by excluding bounding boxes that have more than 20% overlap with adjacent objects. That is, we do not include image patches that feature more than one class at its center or contain any occluded objects.

We further note that for our presently constructed classification dataset, we skip above steps 1-4 whenever subsequent video frames are identical. The latter scenario can arise e.g. when the camera car is stopped at a street intersection. This represents a simple choice to balance the constructed

classification dataset as much as possible and avoid heavy redundancy. For all videos the camera model is assumed to be unchanging and corresponds to Unreal Engine 4's default camera, with an effective shutter speed of $1/s$, an ISO value of 100 and an Aperture equal to unity.

For our specific continual learning experiments the data generation for all scenario's corresponding five training and testing datasets has been conducted using an overall amount of $N_S = 150$ street tiles per video sub-sequence. Because our used investigated deep convolutional neural network does not explicitly encode temporal dependency between video frames and the extracted image patches no longer contain most of the respective scene context, we have further lowered the capturing rate to 5 frames per second, equalling approximately 15 minutes of discretized video per sub-sequence. Depending on the precisely generated video sequence, the above process configuration yields an approximate amount of ~ 20000 classification image patches per video sub-sequence. We show an example illustration of the obtained class balance in figure 5 for one such sub-sequence. We can see that the classes background, tree and humans are balanced almost equally, with a factor less than four in terms of decreased amount of overall car and street lamp instances. However, we did not observe any difficulties with respect to training due to this imbalance. We re-emphasize the main body's statement that all generated dataset sequences are simply a result of particular choices for the specific continual learning experiments to illustrate their volatility across considered scenarios, but are made publicly available for full reproducibility of exact experiments. The respective detailed set-up for how the constructed classification datasets are used for convolutional neural network training are provided in the upcoming supplementary material section. We specifically encourage future researchers to lift our currently included dataset construction simplifications and evaluate our simulator and respective continual learning techniques in light of more challenging tasks.

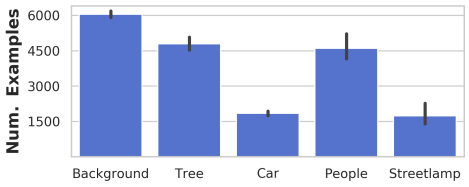

Figure 5: Number of image patches per object category extracted from a video sub-sequence according to the procedure presented in supplementary part C.

D Deep Neural Network Training Hyper-parameters

The preceding supplementary section has provided further details on the main body's experimentation with respect to the constructed and used classification datasets. In this section, we elaborate on the respective hyper-parameters that have been used in conjunction with these datasets. Recall that we have shown empirical results for continual learning classification tasks to contrast five deep continual learning techniques in three scenarios. Consequently the performance of Synaptic Intelligence (SI) [6], Learning without Forgetting (LwF) [7], Elastic Weight Consolidation (EWC) [8], Gradient Episodic Memory (GEM) [9], and Open set Classifying Denoising Variational Auto-Encoders (OCDVAE) [10] has been shown for object classification in incremental scenarios. To reinforce the main body's description of these scenarios in an intuitive form, we show a snapshot of each training video sub-sequence in figure 6. Here, we can see an illustration of the appearance and disappearance of entire categories in the class incremental scenario (left column), a progressive decrease in lighting intensity (center column), and changes in weather (right column).

To focus on the limitations of our selection of deep continual learning procedures in the paper's main body, even in very simple settings, we have used the previously described constructed image patch datasets and have treated each sub-sequence as a separate dataset. That is, we have trained neural networks for multiple epochs on repeatedly shuffled mini-batches until convergence on one sub-sequence, before proceeding to the next sub-sequence that is then trained analogously. Intuitively, this breaks any assumption of temporal dependency, as our used model does not explicitly encode temporal information and predicts on a patch-by-patch basis. It further lifts any requirement of online learning or prediction. Each video sub-sequence has thus been decomposed into a training dataset that is treated in isolation, as is frequently done in simple computer vision classification benchmarks. The respective generated test sequences have been treated in similar isolation. For each sub-sequence a separate image patch dataset has been constructed. The key difference to the training datasets is that for evaluation the test datasets are always evaluated over all images of all sub-sequences ranging up to the current task, where the training is conducted solely on one task at a time.

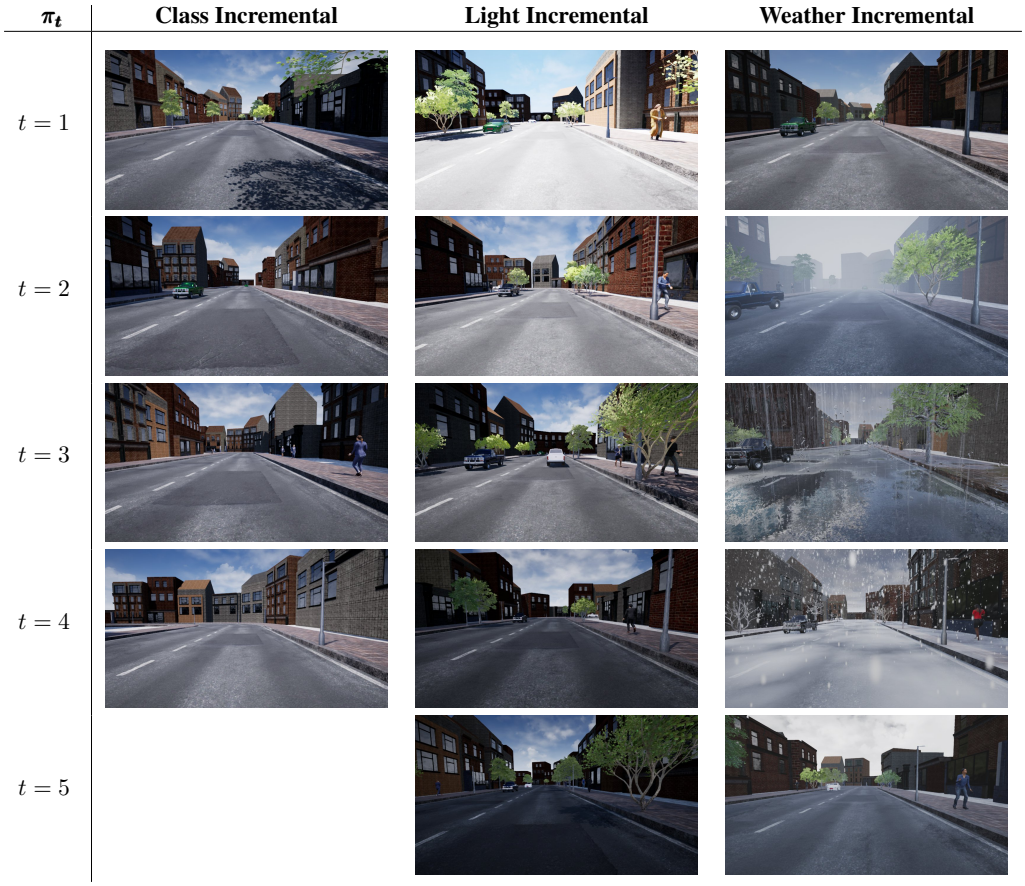

Figure 6: Snapshot of training sub-sequences for incremental class, lighting and weather, learning scenarios. Recall that the task is always object classification with given location.

To assure that the neural network baseline's accuracy is in principle sufficient to learn our tasks and the generated data is sufficient in amount, we have thus first trained a so called continual learning upper-bound. That is, we have trained on the sequence of training data, whereas all sequential tasks are accumulated similar to the test set to investigate whether we can reach a 100% accuracy if the neural network has access to all data. This experiment has been validated for each of the three scenarios and has a particular purpose. It allowed an initial investigation with respect to different state-of-the-art neural networks and their basic training hyper-parameters, in which it has quickly been determined that a four layer convolutional architecture based on the popular DCGAN architecture [11] suffices for our classification tasks.

The DCGAN based "DCNN" architecture (as we do not train a GAN) serves as a common baseline for all experiments, where the encoder is always trained for all methods, and the mirrored decoder is required exclusively in the variational autoencoder based generative replay experiments. Note that as indicated by the original authors of OCDVAE, the classification nevertheless is based on the latent space resulting from the encoder. The main difference is that the generative classifier also takes into account the joint distribution between labels and data, instead of solely predicting the conditional probability of a class given a data point. The precise configuration of encoder and decoder architecture components is shown in tables 1. For each convolutional layer, the tables specify the size of the convolutional kernel, the used amount of learnable features, employed zero padding and convolutional strides. Each convolutional layer is to be understood as a block that is followed by batch-normalization [12] with $\epsilon = 10^{-5}$ and rectified linear unit activations. Classifiers are linear in nature and simply project from the resulting feature space of the encoder to a vector of length corresponding to the amount of classes. As the considered classification task is of single target nature, a Softmax function is used to obtain prediction confidences. In case of the variational auto-encoder, this classifier is preceded by the variational re-parametrization.

Table 1: 4-layer DCNN encoder and decoder architecture. Convolutional (conv) and tranposed convolutional layers (conv_t) are parameterized by size and number of filters, input padding p, and stride s. Fully-connected layers (fc) are parameterized by their number of input and output units, where L_{Dim} is the dimension of the latent space, and $Flat_{\text{Enc}}$ the number of units for the enoder's flattened last layer. Each full-connected and convolutional layer is followed by batch-normalization with value of 10^{-5}, and a rectified linear unit activation function. The decoder's ultimate layer ends on a Sigmoid activation function.

Layer	Encoder
Encoder 1	(conv) $4 \times 4,\ 128,\ p = 1,\ s = 2$
Encoder 2	(conv) $4 \times 4,\ 256,\ p = 1,\ s = 2$
Encoder 3	(conv) $4 \times 4,\ 512,\ p = 1,\ s = 2$
Encoder 4	(conv) $4{\times}4,\ 1024,\ p = 0,\ s = 2$

Layer	Decoder
Decoder 1	(fc) $L_{\text{Dim}} \rightarrow Flat_{\text{Enc}}$ + Reshape
Decoder 2	(conv_t) $4 \times 4 - 512\ p = 0,\ s = 2$
Decoder 3	(conv_t) $4 \times 4 - 256\ p = 1,\ s = 2$
Decoder 4	(conv_t) $4 \times 4 - 128\ p = 1,\ s = 2$
Decoder 5	(conv) $4 \times 4 - 3\ p = 1,\ s = 1$

An adequate length of the training procedure on shuffled mini-batches of the extracted patch dataset for each video sub-sequence has empirically been determined to correspond to 60 epochs for encoder only models and 120 epochs for VAE models, after which convergence of losses has been observed. Optimization is conducted using an Adam optimizer, with an identified learning-rate of 0.001, and the momenta parameter set to $\beta = (0.9, 0.999)$, as proposed by Kingma et al. [13]. All model weights are initialized according to He et al. [14]. The used mini-batch size is 64, and all quadratic image patches have been resized to meet a static input resolution of 64×64, with no further data augmentation applied. Realization of the experimentation has been conducted in approximately 800 GPU hours on an NVIDIA A100-SXM4-40GB GPU cluster using the repository provided in the main body, which is a fork of the public OCDVAE codebase [10] in combination with the Avalanche [15] continual learning library.

Apart from these general shared training hyper-parameters, the five deep continual learning methods each come with their own additional hyper-parameters. For the reader's convenience we also provide a short summary of each method, to provide a better intuition behind the hyper-parameters' significance.

- **LwF:** Learning without Forgetting [7] is a functional regularization mechanism that relies on knowledge distillation [16]. Based on the latter, a trained classifier for an initial task is used as a reference for desired predictions of the task's classes, so called soft-labels, when proceeding to train the classifier on additional classes of subsequent tasks. Specifically, before updating the encoder's and classifier's weights to incorporate new classes, their output vector is recorded. Even though the new class labels are not yet included, the assumption is that preserving this output to an extent also prevents forgetting of the older classes for data instances where the prediction is actually correct. As such, a hyper-parameter λ_{LwF} controls the strength with which the additional loss term that imposes this constraint acts on the continual learning.
- **EWC and SI:** Elastic Weight Consolidation [8] and Synaptic Intelligence [6] both employ parameter regularization by associating each learned parameter with an importance measure. In EWC this measure is based on the diagonal of the Fisher information matrix for each weight layer. For SI it is derived from the extent to which specific parameters have contributed to the total loss decrease. Assuming a generally over-parametrized deep neural network, an additional quadratic loss is then imposed in order to prevent changes to the subset of important parameters. The strength of the loss that simply minimizes the difference between the previous important parameter state and the next is controlled by a hyper-parameter λ_{EWC} and λ_{SI} respectively.
- **GEM:** Gradient Episodic Memory [9] is a hybrid approach of regularization that includes an additional memory buffer to store examples for each learned task. Stored samples are used to impose constraints on gradient updates for future task learning. During optimization, the gradient obtained from the observed new examples is projected using gradients recovered from each task memory, such that the resulting parameter update of the model shall not increase the loss on any of the memorized examples of a previous task. A respective hyper-parameter λ_{GEM} (gamma in the original paper) biases the gradients' projection to favor backwards transfer, as stated by the authors.

- **Exemplar Replay:** Exemplar replay utilizes an external memory and a pattern storing heuristic to interleave the model's training process for a continual learning mechanism [17, 18, 19, 20]. In our implementation we use a naive heuristic where for each trained task a number ϵ of randomly selected patterns from the experienced data is stored to the external memory. During training for any new task, the resulting mini-batches are interleaved with patterns from the external memory, such that they are balanced to be comprised from patterns of each experience.
- **OCDVAE:** Open set Classifying Denoising Variational Auto-Encoder [10] relies on training a deep generative model to replay data in order to prevent older tasks from being forgotten. It uses the VAE's approximation to the data distribution and employs an open set recognition mechanism to sample data distribution inliers for formerly seen tasks and decodes these sampled values into images to rehearse. A respective hyper-parameter serves as a constraint on the inlier likelihood that is expected to yield clear and representative exemplars.

Following the original authors' suggestions we assume a 60-dimensional latent space for OCDVAE, use a unit Gaussian prior and consider sampled data points with an outlier likelihood of less than 5% to be key representatives of the dataset. With respect to SI and LwF, we have observed the choice of λ_{LwF} and λ_{SI} to barely influence the experimental outcome. In our main body's experiments we have used values of $\lambda_{SI} = \lambda_{LwF} = 0.5$. These parameters have been verified by a preliminary grid-search which revealed no fundamental deviations in performance for λ-values of 0.1, 0.5, 1.0, 10.0, and 100.0. Regarding the sizes for external memory of GEM and the Exemplar Replay approach, we allowed storing of $\epsilon = 200$ examples per task, which amounts to approximately 1% to 4% of the total dataset's size of the respective scenarios.

E Operators for Quasi-invariance to Illumination

In the main body we have emphasized the lack of robustness of some continual learning approaches, particularly in scenarios such as the incremental lighting setting, where the overall illumination intensity is varied. For the latter, we have shown that rather simple transformations, as a pre-processing step to operate on a space that is quasi-invariant to homogeneous illumination changes, can solve the imposed continual learning task, without the necessity of any additional mechanism to prevent catastrophic forgetting. Naturally, this is due to the image not changing in the quasi-invariant space when the illumination intensity is varied, much in contrast to the original raw RGB color image input that a deep neural network is typically expected to process.

To provide the visual intuition behind the lighting invariant transformations, we provide illustrations of snapshots showcasing a simulated human with varied degrees of lighting strength and the respective image after transformation in figures 7 and 8. Here, figure 7 exhibits the quasi-invariance of the space resulting from application of local binary patterns (LBP) [21, 22]. Figure 8 provides an analogous depection for the raw image transformation through calculation of color ratios [23], which yield a similarly invariant space under the assumption of non-changing illumination color. The chosen illumination intensity in these figures corresponds to the precise Lux values assumed in the main body's experiments. To explain how these visualized spaces are formed, we briefly summarize the core concepts of the two respectively investigated transformations.

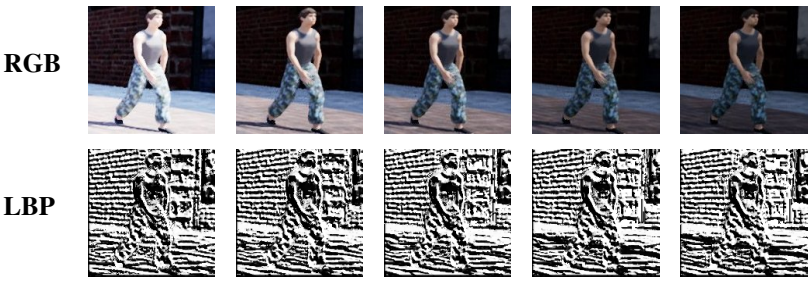

Figure 7: Illustration of the LBP transform applied to the incremental lighting scenario for the example of a pedestrian.

Figure 8: Illustration of the color ratio transform applied to the incremental lighting scenario for the example of a pedestrian.

The quasi illumination invariance of the LBP operator originates from an encoding of pixels according to their relative relations to their neighbors in a limited spatial vicinity. That is, within a chosen radius the operation checks whether values have greater or smaller magnitude. In the case of homogeneous lighting variations, these relationships remain unaltered. For our specific experiments of the main body, the LBP radius has been set to 3, i.e. including 24 neighboring pixels. We note that we have found no necessity to tune this value and have refrained from investigation of different radii, as the initially chosen radius immediately resulted in almost perfect accuracy in the main body's experiment. This is not very surprising as the transformed images of figure 7 are almost identical and learning the task on the any sub-sequence is expected to provide the solution to all other sub-sequences.

The quasi illumination invariance of Gever's and Smeulder's color constancy transformation results from the simple insight that ratios of individual intensity channels remain constant under varying lighting intensity. Recalling the transformation for an individual image channel as introduced in the main body: $c_1 = \arctan(R/\texttt{max}\{G, B\})$, and corresponding definitions for the other two channels, we can qualitatively grasp this concept for the images presented in figure 8. Note how the sequence of raw RGB images and their transformation also nicely illustrates the quasi-invariance, in contrast to an ideally desired full invariance. In the case of too strong illumination intensity coupled with a specific set of non-adaptive camera parameters, we can observe how the brightest image suffers from an onset of over-exposure. As a result, the transformed image features some discrepancy to the other transformed images that remain unchanged for the other considered intensities. Instinctively, this explains why the resulting final accuracy of this approach in the main body is worse than that of LBP, although we reinforce that it still provides significant improvement over processing of raw images.

The short visual examples of this section and their respective quantitative results presented in the main body reinforce our message that continual learning can benefit from further explicit modelling beyond relying on representations derived exclusively from raw data. In particular, operation in quasi-invariant spaces can provide a massive benefit and sometimes a straightforward solution to the catastrophic forgetting challenge. We have chosen the two above operators to illustrate the importance of taking into account the application context and encourage future researchers to conduct additional investigations towards a symbiosis of quasi-invariant operators and deep neural networks with the help of our proposed continual simulation framework.

References

[1] Blender Foundation. Blender.

[2] EVOLVED Software. TreeIt.

[3] MakeHuman Community. MakeHuman.

[4] Epic Games. Unreal Engine 4, 2015.

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[6] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. *International Conference on Machine Learning (ICML)*, 70:3987–3995, 2017.

[7] Zhizhong Li and Derek Hoiem. Learning without forgetting. *European Conference on Computer Vision (ECCV)*, 2016.

[8] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526, 2017.

[9] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. *Neural Information Processing Systems (NeurIPS)*, 2017.

[10] Martin Mundt, Sagnik Majumder, Iuliia Pliushch, Yong Won Hong, and Visvanathan Ramesh. Unified Probabilistic Deep Continual Learning through Generative Replay and Open Set Recognition. *arXiv preprint arXiv:1905.12019*, 2020.

[11] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *International Conference on Learning Representations (ICLR)*, 2016.

[12] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International Conference on Machine Learning (ICML)*, 2015.

[13] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *International Conference on Computer Vision (ICCV)*, 2015.

[15] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an End-to-End Library for Continual Learning. *CVPR-W, Continual Learning in Vision Workshop at Computer Vision and Pattern Recognition (CVPR)*, 2021.

[16] Geoffrey E. Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *NeurIPS Deep Learning Workshop*, 2014.

[17] Sylvestre A. Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational Continual Learning. *International Conference on Learning Representations (ICLR)*, 2018.

[19] Olivier Bachem, Mario Lucie, and Andreas Krause. Coresets for nonparametric estimation-The case of DP-means. *32nd International Conference on Machine Learning, ICML 2015*, 1:209–217, 2015.

[20] Ameya Prabhu, Philip H.S. Torr, and Puneet K. Dokania. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12347 LNCS:524–540, 2020.

[21] Dong-Chen He and Li Wang. Texture Unit, Texture Spectrum, And Texture Analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 28(4), 1990.

[22] Li Wang and Dong-Chen He. Texture Classification Using Texture Spectrum. *Pattern Recognition*, 23(8):905–910, 1990.

[23] Theo Gevers and Arnold W. M. Smeulders. Color based object recognition. *Pattern Recognition*, 32(3):453–464, 1999.