Class Incremental Learning in Deep Neural Networks

by

JunYong Tong

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

Statement of Contributions

The content from one future publication is included in this thesis with contributions from others. The contributions for the work are listed below. I performed the majority of the design, background review, experimentation and writing.

The group of work presented in Chapter 3.3 and 3.4, with the accompany results in Chapter 4 are drafted in a manuscript submitted to IEEE CVPR2021 conference proceeding (rejected). The paper's authors, excluding myself, include Dr. Javad Shafie and Dr. Paul Fieguth. All the authors supported with problem conceptualization and editing. Dr. Javad Shafie and Dr. Paul Fieguth provided useful advice on the structure of the report and experiments design. The revised manuscript was submitted and accepted to IEEE CVPR2021 Continual Learning Workshop under ”Findings”.

Abstract

With the advancement of computation capability, in particular the use of graphical processing units, deep learning systems have shown tremendous potential in achieving super-human performance in many computer vision tasks. However, deep learning models are not able to learn continuously in scenarios where the data distribution is non-stationary or imbalanced, because the models suffer from catastrophic forgetting. In this thesis, we propose an Incremental Generative Replay Embedding (IGRE) framework which employs a conditional generator for generative replay at the image embedding level, thus combining the superior performance of replay and reducing the memory complexities for replay at the same time. Alternating backpropagation with Langevin's dynamics was used for efficient and effective training of the conditional generator. We evaluate the proposed IGRE framework on common benchmarks using CIFAR10/100, CUB and ImageNet datasets. Results show that the proposed IGRE framework outperforms state-of-the-art methods on CIFAR-10, CIFAR-100, and the CUB datasets with 6-9% improvement in accuracy and achieves comparable performance in large-scale ImageNet experiments, while at the same time reducing the memory requirements significantly when compared to conventional replay techniques.

Acknowledgements

I would like to thank Prof. Paul Fieguth for his continuous support during my masters program as my supervisor. Thank you for all the guidance and support in my journey.

I would like to thank Prof. Krzysztof Czarnecki and Dr. Javad Shafiee for serving as readers of my Masters thesis.

I would like to thank Amir and Javad for the stimulating conversations about this research topic. Finally, I want to thank my parents and my girlfriend for their continuous support and encouragement throughout my years of study, and giving me space the past month to complete this thesis.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This chapter aims to introduce the importance of continual learning in deep neural network models as these models find themselves in more and more engineering applications ranging from computer vision [72, 88] to biomedical science [77]. There are three sections in this chapter. The first section aims to motivate the problem of continual learning through the phenomena that deep neural networks are able to outperform human in many tasks, yet unable to learn continuously. The second section aims to narrow the scope and objective of continual learning for this thesis. The third section outlines the contributions and the thesis structure for the reader.

1.1 Motivation

In this section we motivate the problem of continual learning in deep neural networks.

In recent years, machine learning with deep neural networks gave rise to significant improvement to solving many research and industrial problems. Many tasks in computer vision have achieved state of the art performance surpassing human level performance. For example, deep convolutional neural network models can surpassed human level classification performance [72]. In natural language domain, deep neural network technologies are being used in search engine [88] or text analysis [99]. Deep learning techniques are also used in physical sciences to solve previously unsolve problems, for example AlphaFold technique in [77] is used to solve protein folding problem in biology. Deep neural networks are able to achieve performance far exceeding human level in many tasks and at a faster speed [72, 99].

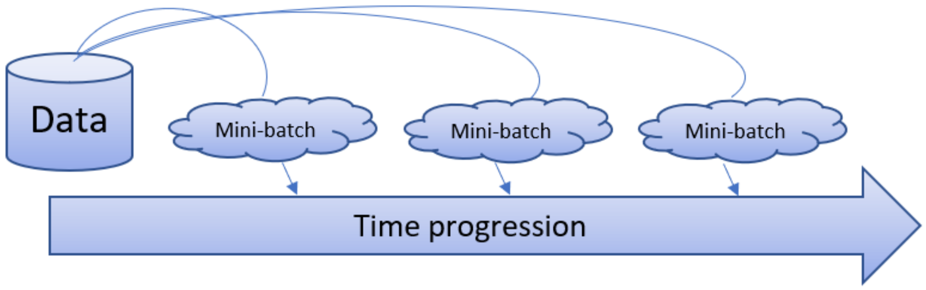


Figure 1.1: This figure illustrates the batched learning process where the learning algorithm samples mini-batches from a dataset. Ideally, each mini-batch is representative of the dataset distribution which the learning algorithm is set up to learn.

Two of the most important components to the success of deep neural networks (DNN) are the abundance of labelled data and the use of automatic differentiation [20, 63] for stochastic mini-batch gradient descent [71]. Training a deep neural network relies on the assumption that the data that we use are *independent and identically distributed* (i.i.d) within the dataset [18, 71]. The stochastic gradient descent of [71] is designed based on the i.i.d assumption and required to loop through the dataset for many epochs. Ideally, each mini-batch sample will be representative of the dataset distribution. The algorithm will adjust the weights of the DNN based on the mini-batches sampled from the dataset as shown in Figure 1.1. Without the i.i.d assumption, there is no reason to believe the model trained with stochastic gradient descent would generalize to the test data.

Despite DNNs' superhuman performance in many tasks, a hall-mark of human intelligence is that human is able to learn continuously from experience. The term *continual learning* [62, 70] is used in the machine learning literature to describe the scenario where the model is required to learn from data stream continuously. Learning continuously from data poses a particular challenge for machine learning models as the i.i.d assumption is violated because we cannot sample representative mini-batches. As such the models fail to learn the data distribution. Enabling deep neural networks to be continual learners is made more challenging as the models can easily overfit to the latest available data.

The *catastrophic forgetting* or *catastrophic interference* [15, 16, 55] problem describes the phenomenon whereby a DNN loses information about previously learned tasks during the learning of a new task [7, 73, 80]. The issue occurs because important network weights, trained on some previous tasks, are modified in the process of learning a new task. This problem plagues DNNs and is the main bottleneck toward continual learning of DNN models in different domains and for different applications. As a result, a continual learning framework to address these issues is highly desirable.

1.2 Continual Learner Objectives and Evaluation

The objectives of a continual learner/learning agent are to be able to learn from current data and continue learning later without forgetting previously learned information. In particular, this thesis focuses on the class incremental learning scenario where a model is require to distinguish between all classes from all tasks. More precisely, the ideal class incremental learner should possess the following three properties:

1. It should perform well on previously learned tasks.
2. It learns new tasks, building on the knowledge of previous tasks.
3. It has the ability to improve or maintain performance on the previously learned tasks.

Objectives 1 and 2 are directly related to the stability-plasticity dilemma [5, 56] of a learning agent. In the context of continual learning, the agent needs to trade-off its ability to learn new information (plasticity) without forgetting previously learned information (stability) [33, 97]. Objective 3 describes the ability of the learning agent to have the ability to consolidate previous knowledge with new information.

Although many existing continual learning algorithms are designed based on the three objectives, the algorithms are often evaluated on different scenarios. Broadly, there are three main evaluation scenarios in the literature [26, 80, 97], namely *domain incremental* learning, *task incremental* learning and *class incremental* learning. Each scenario provides the learning agent different compute resources during the course of learning and evaluation. The variability in evaluation scenarios makes choosing and classifying existing algorithms particularly difficult, and inspired large scale survey research papers like [12, 26].

The three scenarios are of different difficulty. The availability of task labels affects the difficulty of a continual learning scenario significantly. Task incremental and domain incremental are given task labels during evaluation but class incremental is not. The results in [12, 26] suggest that having task labels during evaluation allows a model to choose the correct prediction head and hence outperforming scenario without task labels. More importantly, it is unrealistic to have task labels in real-life applications like self-driving cars.

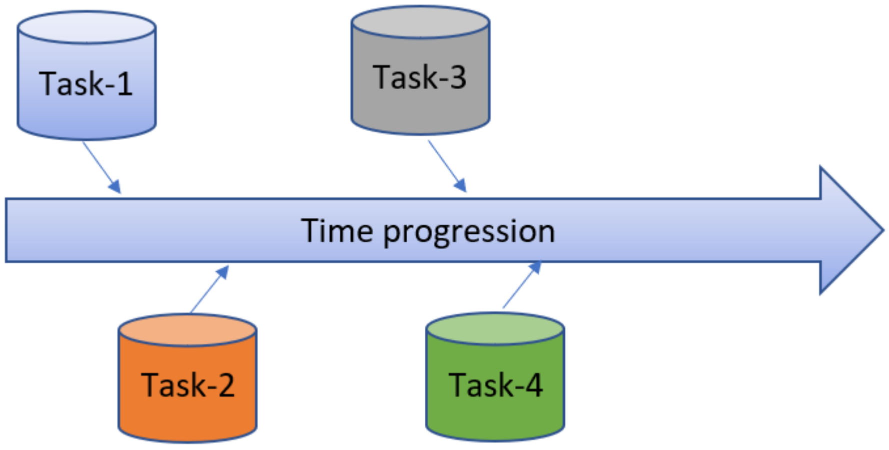


Figure 1.2: This figure illustrates the class incremental learning process where data are presented to the model incrementally. Different colours represent different task data. The task data are only accessible at one time point. (Best viewed in colour)

This thesis focuses on the particular continual learning scenario of *Class Incremental Learning* (CIL) for image classification. Each task in CIL introduces new disjoint classes into the learning and past classes will no longer be accessible. This process can be seen in Figure 1.2. This means the model will not be able to see past examples again when proceeding to a new task. In CIL, we aim to learn a single classifier, also called a *single-headed* classifier, to classify new classes without forgetting previously-seen classes [67]. In this setting, we can evaluate whether the algorithm is able to differentiate classes from different tasks which is a more realistic setting compared to when task labels are given during evaluation.

1.3 Contributions and Thesis Structure

As will be discussed in Chapter 2, a continual learning strategy can be divided into three categories: I) regularization, II) dynamic architecture and III) replay. This thesis focuses on generative replay — a type of replay strategy where a generative model is utilized to generate memory for replay purposes. In particular, this thesis presents an incremental generative replay embedding (IGRE) framework, a simple generative replay technique for incremental supervised learning.

In summary, the contributions of this thesis are:

1. A simple and efficient generative replay framework to mitigate catastrophic forgetting in class incremental learning scenario with modest computation and memory requirements.
2. A method to train a generative model for supervised incremental learning without task labels during the training and evaluation phases.
3. An effective generative model for long learning sequences compared to equivalent state of the art methods.

The thesis is structured as follows:

- Chapter 2 will provide necessary background information to understand the CIL problem and terminologies used in the other chapters.
- Chapter 3 will provide intuition and visualize the changes of feature embedding under the CIL problem. The intuition will be developed into the IGRE framework.
- Chapter 4 will present the experimental results in various CIL scenarios on three popular datasets and the comparison results with related state of the art methods. We also provide analysis on the limitations of the proposed framework.
- Chapter 5 will summarize and conclude the thesis. Three future research directions are suggested in this chapter.

Chapter 2

Background and Literature Review

This chapter formally introduces the necessary notation and background knowledge required for the rest of the thesis. We assume that readers have some basic knowledge of deep neural networks (DNN), in particular the basic training procedure using stochastic gradient descent. We point the reader to the deep learning journal paper [41], online deep learning book [18] or Coursera's online resource for the basic knowledge, and recap only the concepts necessary to make this thesis self contained.

In Section 2.1, DNNs will be presented as discriminative models for image classification [41, 42]. In Section 2.2, we present generative models as an alternative to discriminative models for data distribution learning. Finally, in Section 2.3, we present our definition of class incremental learning and review some literature related to continual learning.

2.1 Discriminative Models

In this section we present our model of a deep neural network [18, 40, 41] as a discriminative model for classification problems. In particular, we present deep convolutional neural network designs and training paradigms as discriminative models.

2.1.1 Feature Extraction

An *image classification* problem refers to the task of assigning a class label to an image based on extracted information/features from the image. A DNN usually consists of stacks of different neural layers that learn to detect essential features in the data and classify the data according to the extracted features [18, 41]. Given a DNN, any stack of layers before the final output layer can be considered as a *feature extractor*, as a series of transformations [18, 53]. Ideally, the feature extractor transforms the input to a feature representation that can be classified easily. This representation is called a *feature embedding*.

The feature extractor is designed based on the task or problem at hand. A feature extractor for image-based tasks usually consists of convolutional layers, normalization layers and non-linear activation layers [18, 23, 40]. Mathematically, given an input $\mathbf{x}$, we denote the feature extractor as f, then the extracted feature map $\mathbf{f}$ is given by,

$$\mathbf{f} = f(\mathbf{x}). \tag{2.1}$$

The feature extractor f is a function mapping $\mathbb{R}^D \to \mathbb{R}^d$, a mapping from the image space to a lower dimensional space, that is $d \ll D$ and so $\mathbf{f} \in \mathbb{R}^d$, such that classification can be done at much lower dimension which is an easier problem to solve. The composition of L layers for f is then expressed as,

$$f = f_L \circ f_{L-1} \circ \cdots \circ f_1, \tag{2.2}$$

where f_i, $i \in \{1, \ldots, L\}$, denotes a layer.

The convolutional layers consist of filters which are convolved with the layer input to extract certain patterns, and stacking the layers allows the DNN to detect complex features [23, 82]. The deep convolutional neural networks are shown to have inductive bias at learning spatial information [4, 10] and efficient at learning functions for image-based tasks [10]. The filters are able to capture local dependencies and able to learn features that are invariant to certain transformations, like rotation and shift [53]. The non-linear activation, normalization and pooling layers [18, 23, 82] are included in the network so that complex transformations can be learned from the data and improve the training dynamics. In addition to its inductive bias, convolution filters have significantly fewer parameters compared to fully connected layers, thus making convolution filters a desirable architectural design choice.

The DNN is trained to learn the right combination of filters to discriminate and classify the classes within a given data set. Although the convolutional network may achieve great classification performance within the data set, it is susceptible to adversarial examples [39] and it is not robust to examples in different coordinate frames [74]. As such, it requires a large database to be able to train a deep convolutional neural network.

2.1.2 Classification Module

Recall that an image classification problem refers to the task of assigning a class label to an image. As such we need a function to map the extracted features, **f** from Equation (2.1), to a class label. In this section, we describe the classification modules which convert the detected features into a class label.

The classification modules are tasked with classifying the data based on the extracted feature embedding. Thus, the classification modules should be flexible at learning a wide variety of functions. Fortunately, the combination of fully connected (FC) layers and non-linear activation has been shown to be a universal function approximator in previous seminal works [11, 18, 48]. This means that the FC layers possess the capability to approximate the ideal classification function for a given dataset. Therefore, FC layers and non-linear activation functions are often good architectural design choices for classification.

In general, the FC layers apply a linear transformation to their input, then non-linear activation functions will introduce non-linearity to the transformation. Without loss of generality[1], let $\mathbf{x}$ be an input to a FC layer, then the transformation that $\mathbf{x}$ undergoes can be described as,

$$\mathbf{h} = \sigma(\mathbf{W} * \mathbf{x} + \mathbf{b}), \tag{2.3}$$

where $\mathbf{W}$ is a matrix of learnable weights, $\mathbf{b}$ is a learnable bias vector, and $\sigma(\cdot)$ is an element-wise non-linear activation function. Examples of such activation functions include *sigmoid, tanh* and Rectifier Linear Unit (*ReLU*) [18, 43]. The output $\mathbf{h}$ is usually called a hidden output or, if the output is the last output of the network, then it is also called *logits*.

For K-class classification, the logits $\mathbf{h} \in \mathbb{R}^K$ are usually normalized with the *softmax* function into a probability vector,

$$softmax(\mathbf{h})_i = \frac{\exp(h_i)}{\sum_j^K \exp(h_j)}, \mathbf{h} = (h_1, \ldots, h_K). \tag{2.4}$$

Then the index with highest value is chosen to be predicted class label, that is, the prediction p_i for the i-th sample will be given as,

$$p_i = \underset{i \in \{1,\ldots,K\}}{\arg\max}\ softmax(\mathbf{h})_i. \tag{2.5}$$

[1]If we stack the FC layers on top of the feature extractor, then we have $\mathbf{f}$ as the input.

Since the index with the highest probability is chosen as the predicted class, the softmax normalized logits are often interpreted as a probability.

Feature extractors and classification modules stacked together are often called an *end-to-end* model. Together with a differentiable loss function, the end-to-end model can be trained through stochastic gradient descent (or its variant) [30, 71], with the gradients of the hidden layers calculated with automatic differentiation algorithm [18, 20]. For example, a popular loss function for multi-class classification is cross-entropy loss [18, 41],

$$\mathcal{L}_{cls}(\mathbf{h}, y) = -\log\left(\frac{\exp(h_y)}{\sum_j^K \exp(h_j)}\right), \tag{2.6}$$

where $\mathbf{h}$ is the logits of a sample and y is the class label. Cross-entropy loss takes the log of softmax thus having numerical stability as the additional benefit. Since the softmax function is differentiable and numerically stable it is an ideal function for normalizing the logits, and ensuring the gradients can be back propagated to the feature extractor of the network.

Since this end-to-end model is able to discriminate between classes, it is also called a *discriminative* model in literature [18, 28]. In the next section, we will introduce a different class of models, generative models, where their main goal is to model a given data distribution.

2.2 Generative Models

In addition to the discriminative models, introduced in Section 2.1, we now introduce a different class of model called generative models [28, 31]. Generative models focus on learning the data generation process, also called *data distribution* [28]. A generative model learns to represent the data in a space with fewer dimensions than the original data space. The space with fewer dimensions is called *latent space*. The goal of generative models is not to discriminate between classes, but to learn a mapping between a smooth latent space and the data space.

In the generative model literature there are four main generative model frameworks:

1. Generative moment matching networks [17, 46, 68],

2. Variational Autoencoders (VAE) [31, 32],

3. Generative Adversarial Networks (GAN) [19, 44],
4. Normalizing flow networks [35, 69].

Despite the nice theoretical properties of normalizing flow networks, this framework has yet to achieve comparable performance against the other generative methods [36], hence we do not consider them in this thesis. We will describe the generative moment matching networks, VAE, and GAN frameworks in order for the reader to have a more comprehensive understanding of the proposed method as compared with the state of the art methods in Chapter 4.

2.2.1 Generative Moment Matching Networks

The first family of generative models is the generative moment matching network [46]. The models in this family are similar to statistical models learned through maximum likelihood estimation. This framework aims to train a generative model that matches a smooth latent space to the data observations, hence learning the distribution induced by the data.

In this thesis we only consider the d-dimensional multivariate Normal distribution with diagonal unit variance, denoted $\mathcal{N}(0, I_d)$, as the smooth latent space. Note that it is possible to use any arbitrary distribution for the latent space, we choose multivariate Normal for its computational efficiency. We can then formulate our generative model explicitly as

$$\mathbf{z} \sim \mathcal{N}(0, I_d) \tag{2.7}$$

$$\mathbf{x} = G_\theta(\mathbf{z}) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2 I_n). \tag{2.8}$$

Here the latent variable $\mathbf{z}$ is sampled from a multivariate unit variance Normal distribution. G_θ denotes a generator neural network, with θ as the parameters in the generator. G_θ maps a latent vector from $\mathcal{N}(0, I_d)$ to match $\mathbf{x}$ with some noise error ϵ, with tunable σ. Thus, the complete log joint-likelihood of Equation (2.8) is given by

$$\log P_\theta(\mathbf{x}, \mathbf{z}) = -\frac{\|\mathbf{z}\|_2^2}{2} - \frac{\|\mathbf{x} - G_\theta(\mathbf{z})\|_2^2}{2\sigma^2} + c, \tag{2.9}$$

where c is a constant independent from $\mathbf{z}$, and σ is the pre-specified standard deviation of the model. Thus the observed data log-likelihood is obtained by integrating out the latent variable

$$L(\theta) = \sum_{i=1}^{m} \log \int_z P_\theta(\mathbf{x}_i, \mathbf{z})\, d\mathbf{z} \tag{2.10}$$

$$= \sum_{i=1}^{m} \log P_\theta(\mathbf{x}_i), \tag{2.11}$$

where m is the number of data samples. The goal then is to maximize (2.11), for which we can adopt gradient ascent. The gradient of observed data log-likelihood with respect to parameter θ is

$$\frac{\partial}{\partial\theta} \log P_\theta(\mathbf{x}) = \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})}\left[\frac{\partial}{\partial\theta} \log P_\theta(\mathbf{x}, \mathbf{z})\right]. \tag{2.12}$$

However the expectation in (2.12) is usually intractable, since the set of training data is discrete and sparse, and as such the model cannot see adequate variations of samples to model the space effectively. In order to partially alleviate the problem, we model $\mathbf{x}$ at reduced dimensionality instead of at full dimensionality. The details will be described in Section 3.2.

2.2.2 Variational Autoencoder

The second framework for learning a data distribution is through the Variational Auto-Encoder (VAE) introduced in [31]. In this framework, we aim to train an encoder to map/compress an input to a latent vector and a decoder to reconstruct the input from a latent vector.

The main idea behind VAE is that we use a variational distribution that is easy to sample and compute to approximate the data distribution through Kullback-Leibler (D_{KL}) divergence [38]. The Gaussian distribution is usually chosen to be the variational distribution for its sampling efficiency and its easily computed likelihood. The encoder, denoted as q_θ, maps the input to the variational distribution, $\mathcal{N}(\mu, \sigma)$, by mapping the input to a latent representation/vector. The decoder, denoted as g_θ, is trained to reconstruct an input from its latent representation. The variational distribution in VAE is regularized with a unit Normal, $\mathcal{N}(0, 1)$, as a prior distribution to prevent distribution collapse [9].

Formally, given an input $\mathbf{x}$, VAE aims to find the optimal latent representation in $\mathcal{N}(\mu, \sigma)$ such that the reconstruction error for $g_\theta(\mathbf{z})$ is minimized. In particular, the loss of VAE can be written in the form

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{recon} + \mathcal{L}_{prior} && (2.13)\\ &= \|\mathbf{x} - g_\theta(\mathbf{z})\|_2 + D_{KL}(q_\theta(\mathbf{x}), \mathcal{N}(0,1)). && (2.14)\end{aligned}$$

The $D_{KL}(q_\theta(\mathbf{x}), \mathcal{N}(0,1))$ term regularizes the VAE, preventing the decoder from memorizing the input's latent representation. Similar to generator moment matching network, this framework allows user to sample from a closed-form distribution to generate samples that approximates the underlying data distribution.

Noticed that the VAE loss in Equation (2.14), is quite similar to the generative moment match network loss in Equation (2.9). The difference is that the VAE framework is dependent on a good quality inference network $q_\theta(\mathbf{x})$ to map $\mathbf{x}$ to a latent vector $\mathbf{z}$ which would then be used as input for the decoder g_θ, whereas the generator moment matching networks from Section 2.2.1 learn the mapping from $\mathbf{z}$ to $\mathbf{x}$ directly, which is more straight forward and computationally efficient.

2.2.3 Generative Adversarial Networks

The third generative modelling framework which we will describe is the Generative Adversarial Network (GAN) of [19]. In this framework, a discriminator network, D, and a generator network, G, are trained together such that the generator learns the data distribution. The generator learns to generate realistic inputs whereas the discriminator learns to differentiate between the real and generated inputs. In other words, the generator learns to fool the discriminator and the discriminator learns to not be fooled.

In this framework, the generator learns to map a latent space to the data manifold. The generator takes a latent vector, $\mathbf{z} \in \mathbb{R}^d$, and generates $\hat{\mathbf{x}} \in \mathbb{R}^D$ that is in the same space as $\mathbf{x} \in \mathbb{R}^D$. The discriminator will differentiate between $\mathbf{x}$ and $\hat{\mathbf{x}}$. The discriminator acts as a regularizer and guides the generator to produce realistic $\hat{\mathbf{x}}$. Both the generator and discriminator have the same loss function given by,

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right]. \tag{2.15}$$

The discriminator learns to maximize the loss, while the generator learns to minimize the loss. Note that the first expectation is taken over $\mathbf{x} \sim p(\mathbf{x})$ for $D(\mathbf{x})$, this can be interpreted as the average probability that the discriminator can correctly identifies a real image. The second expectation is taken over $\mathbf{z} \sim p(\mathbf{z})$ for $D(G(\mathbf{z}))$, in other words the discriminator maximizes this quantity by correctly identifying generated images. As such, the discriminator guides the generator to learn a mapping from smooth latent space to data space and to generate realistic inputs.

This class of generative model is able to produce visually realistic images, but the model suffers from training dynamics instability. Many methods [3, 21, 27] have been proposed to stabilize the training dynamics. In particular, conditional GAN [27] uses conditional information for training the model. In the next subsection, we will introduce conditional generators, which allow the user to generate specific conditions from the data distribution, making conditional generators particularly useful for class incremental learning.

2.2.4 Conditional Generative Models

All generative models described previously can be extended into conditional generative models. Conditional generative models are trained with conditional variables such that the models are able to generate specific conditions of interest. For example, using class labels as conditional variables enables the models to generate images from specific classes [27, 83]. We will introduce some of the known benefits of using conditional variables in generative models in this subsection.

The conditional VAE in [83] models the data distribution through latent variables and class labels as conditional variables. The conditional model is obtained through slight modification to Equation (2.14), by simply including the conditional variable c into the encoder and decoder:

$$\begin{aligned} &\|\mathbf{x} - g_\theta(\mathbf{z})\|_2 + D_{KL}(q_\theta(\mathbf{x}), \mathcal{N}(0,1)) \\ \rightarrow &\|\mathbf{x} - g_\theta(\mathbf{z}, c)\|_2 + D_{KL}(q_\theta(\mathbf{x}, c), \mathcal{N}(0,1)). \end{aligned} \tag{2.16}$$

This way, the conditional information enables the VAE to learn the styles and features of different classes, and generate images of specific classes.

Conditional GANs (cGAN) [27, 57] are the extension of GANs with conditional variables. Similar to conditional VAE, we simply include the conditional variables to the discriminator and generator. The loss function for training GANs from Equation (2.15) becomes

$$\begin{aligned} &\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log(D(\mathbf{x}))\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right] \\ \rightarrow &\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log(D(\mathbf{x}, c))\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z}, c), c))\right]. \end{aligned} \tag{2.17}$$

In addition to being able to generate specific conditions, cGANs are observed to have more stable training dynamics [21, 27].

Lastly, conditional Generative moment-matching networks [68] are the conditional variant of generative moment-matching networks, where the generators are trained with conditional variables. Similar to cVAE and cGAN, we simply include the conditional variables as input to the generator, hence Equation (2.9) becomes

$$\begin{aligned} & -\frac{\|\mathbf{z}\|_2^2}{2} - \frac{\left\|\mathbf{x} - G_\theta(\mathbf{z})\right\|_2^2}{2\sigma^2} \\ \to & -\frac{\|\mathbf{z}\|_2^2}{2} - \frac{\left\|\mathbf{x} - G_\theta(\mathbf{z}, c)\right\|_2^2}{2\sigma^2}. \end{aligned} \tag{2.18}$$

The feature-to-feature generator in [100] uses class attribute information as conditional variables for the generator network to learn class specific style. The conditional generator in [100] was able to generate images of comparable quality to cGAN, while having fewer parameters in the network. In this thesis, the features-to-features method is extended to continual learning scenario.

2.3 Class Incremental Learning

In this section, we formally introduce the class incremental learning (CIL) problem [26, 67, 86].

The *class incremental learning* scenario refers to the requirement of learning classification tasks on data incrementally [26, 67] with a single classifier. Each task in the CIL scenario introduces new disjoint classes into the learning and past classes will no longer be accessible, as illustrated in Figure 1.2. In other words, once the data for a class are presented, the presented labelled class data will not be accessible again for learning in the future, unless the data are selected for memory replay.

In order to simulate the CIL scenario with a dataset, the dataset will be split into τ groups with each group containing disjoint sets of labels from the dataset. The groups will be presented individually to the model sequentially as a *task* to be learned. The stability and plasticity trade-off of the model, recall the stability-plasticity dilemma [5, 56] from Section 1.2, will be evaluated with all previously seen class labels.

The DNN model is presented with a group of data at task t, where $t \in \{1, \ldots, \tau\}$. We let

$$\mathcal{X}^t = \left\{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^D,\, y_i \in \mathcal{Y}_t\right\}_{i=1,\ldots,m_t} \tag{2.19}$$

denotes the set of labelled images for task t, $\mathbf{x}_i$ denotes image with a class label y_i, $\mathcal{Y}_t$ denotes the set of class labels for task t, and m_t denotes the number of labelled pairs in $\mathcal{X}^t$. In the CIL scenario, the model is presented with $\mathcal{X}^t$ at task t for learning. At task s, where $t < s$, the model will not be able to access the entire $\mathcal{X}^t$ for learning, in particular we have $\mathcal{Y}_t \cap \mathcal{Y}_s = \emptyset$. The model is required to learn the information for $\mathcal{X}^s$ at task s without forgetting information for $\mathcal{X}^t$.

Without continual learning strategies, the model will catastrophically forget [15, 16, 33] information of task t after learning task s. That is because the data distribution in task t is not identical to the data distribution in task s. In this case, as we train the model for task s, the weights in the model will shift towards the set of weights that is optimal for task s without consideration for task t. As the weights shift for task s, the model's performance for the previous tasks will deteriorate.

2.4 Regularization

The simplest non-trivial approach to continual learning is via weight regularization [2, 33, 97]. Weight regularization can be separated into two streams, namely weight importance computation and knowledge distillation.

Weight Importance

Weight importance regularization methods, such as Elastic Weights Consolidation (EWC) [33], Memory Aware Synapses (MAS) [2] and Synaptic Intelligence [97], include a penalty term in addition to the classification loss function of the discriminative model to regularize individual weights from being updated significantly during the training of a new task. The weight importance regularization methods are simple and conceptually attractive, however these methods were shown to be inefficient at class incremental learning [8, 26], as regularization requires the network to be highly over-parameterized to prevent task interference during training. However, weight regularization approaches are often combined with other approaches to improve performance because of their simplicity.

A regularization term $\mathcal{L}_{reg}$ is added to the model's classification loss function $\mathcal{L}_{cls}$ from Equation (2.6) as

$$\mathcal{L}(\theta) = \mathcal{L}_{cls}(\theta) + \lambda \mathcal{L}_{reg}(\theta), \tag{2.20}$$

where λ is the regularization coefficient. For weight regularization, $\mathcal{L}_{reg}$ is parameterized as

$$\mathcal{L}_{reg}(\theta) = \sum_{i=1}^{P} \Omega_i (\theta_i - \theta_i^*)^2, \tag{2.21}$$

where Ω_i is a function that calculates the *parameter importance* for the i-th parameter, θ_i^* is the i-th weight learned from the previously seen tasks, and P is the number of weights (parameters) in the neural network model.

Ω_i quantifies the importance of the θ_i^* parameter, and it may be calculated in a variety of different ways. Parameter freezing ($\Omega \rightarrow \infty$) can be considered as one extreme in regularization. In EWC, Ω_i is calculated as the diagonal of the Fisher information matrix [33], Calculating the importance requires storage of a diagonal matrix and an additional loop through all the parameters, which could be constrained by memory since a typical feature extractor contains millions or billions parameters. On the other hand, Ω_i in MAS[2] is calculated as the l_2 gradient of the neural network output given the inputs. MAS reduces the need for storing the Fisher information matrix but requires an additional backward pass to calculate the l_2 gradients.

Distillation

The idea of using knowledge distillation [24, 47, 81] for continual learning originates from transfer learning [47, 79, 96]. Knowledge distillation is a technique for transfer learning, where knowledge gained from solving one task is applied to a different but related problem. In other words, the knowledge of model A is *transferred/distilled* to model B by imposing model B to have the same output as model A, given the same input. Thus, implicitly regularizing the weights of model B and enabling model B to learn new tasks faster.

Similarly knowledge distillation can be used for continual learning. After model A is trained for a task, the knowledge is distilled to model B while model B is learning a new task. In [81], the authors use mean-squared error to distill the knowledge of an object detector in the incremental learning setting in an effort to minimize catastrophic forgetting. In [47], the authors regularize the divergence between logits of models A and B for incremental learning. A common drawback in distillation is the need to preserve a teacher model for the student to learn.

2.5 Architecture Expansion

The architectural approach to continual learning involves changing the DNN's architecture dynamically to enable the model to learn new data [12, 45, 66, 73]. These approaches can be separated into explicit expansion and implicit expansion.

Explicit Expansion

The learning algorithm for explicit expansion methods [45, 73, 95] will modify the DNN architecture explicitly to combat catastrophic forgetting. In this family, the algorithm expands the model as the model encounters more data or as the learning proceeds to successive tasks.

One such example is Progressive Neural Networks [73] (PNN), where for each new task a new model is created and connected to the previously learned model. The idea is to use the previously learned representation while training the new model on a new task. In PNN, the number of weights grows at a quadratic rate with respect to number of tasks. Thus, PNN is only limited to short task sequences. Another example that explicitly modifies the DNN's architecture is the Dynamically Expandable Networks (DEN) in [95], where neurons are added or pruned dynamically for the new tasks. DEN tackles the parameter growth problem by pruning unimportant parameters. The Neural Dirichlet Process Mixture Model in [45] attempts to reduce the quadratic rate of increasing parameters by modelling the model's capacity through a probabilistic model.

These methods often require well-defined task boundaries during training and inference in order to select the right model output. Further, these methods cannot be easily scaled to scenarios having long learning sequences [8]; for both reasons we do not consider these approaches further.

Implicit Expansion

A continual learning algorithm is considered an implicit expansion method when it adapts the model without explicitly changing the architecture. This is usually achieved through freezing a subset of the weights [78] in the DNN during learning and changing the forward pass path [65].

Freezing weights consists of choosing some weights at the end of a task that will not change in the future. The backward pass will not be able to tune them anymore; however, they can still be used in the forward pass. For example in Hard Attention to Task [78] and Piggyback [54], the authors define a mask for each task to prevent the weights from changing when learning new tasks and using the masks to choose the weights that will be used for forward pass during inference. In Random Path Selection (RPS) [65], weights were regularized with distillation techniques instead of freezing and the method dynamically selects the optimal forward paths within the DNN.

This family of methods ensures that the frozen weights will not be modified, and tries to keep enough free parameters to learn in the future. The difficulty lies in freezing enough weights so that catastrophic forgetting is minimized, but not so much that the model is not able to learn from new tasks.

2.6 Replay-based Approaches

Replay-based methods [50, 67, 85] aim to enable continual learning by repeatedly "rehearsing" memories of past tasks; the memories can either be raw input or processed input. To be specific, a replay-based approach either requires an external data buffer for sample storage or an addition model to reconstruct past information.

The approach of using a data buffer is termed a *rehearsal* approach [12, 67, 70]; this approach aims to model the past with only few examples. The examples are stored in a data buffer called a *coreset*. The coreset data are carried forward to the next task to serve as a memory for the model. Although randomly choosing samples will work for rehearsal, it is sub-optimal, and ideally the samples are chosen carefully in order to maximize their representation capability of past memory.

An alternative approach to using a coreset is *generative replay* [61, 80], where a generative model is trained on the data distribution and used to sample the past data in the future task. Samples from past data distributions are mixed into the current training data, enabling the model to rehearse from previous data distributions and minimizing catastrophic forgetting.

Rehearsal

In rehearsal approaches [50, 67], the algorithms aim to find examples that are representative of the current task and store the representative examples into a coreset. The data in the coreset will be mixed into the training data in future tasks. In addition, the coreset data can be used for regularization as to prevent the model from forgetting the previous task information. As such, rehearsal approaches enable the models to continually learn new tasks without explicitly freezing weights, giving the model more plasticity.

A prominent example is incremental classifier and representation learning (iCaRL) [67], which uses a greedy algorithm to choose which raw-input images will be stored in a coreset, and the coreset is then included in training subsequent tasks. In addition, iCaRL also uses the coreset data for distillation when learning new tasks. The LUCIR algorithm in [25], uses cosine normalization, a less-forgetting constraint, and inter-class separation constraint with a coreset to rebalance the class incremental training process. The algorithm in [50] trains a model on top of LUCIR to pick the examples that are most representative of the current task to be included into the coreset. The TPCIL in [85] models the topology of the feature embedding and selects the raw-input images to the coreset for replay.

Replay methods generally outperform other methods, and even naïve replay yields a strong baseline performance in comparison to other approaches, as shown in [26]. A challenge with rehearsal approaches is the management of an additional buffer requirement for storing examples. Storage of raw input data is not efficient, as often raw input contains more information than necessary. In addition, having a fixed buffer size could limit the quality of replay as the buffer may not be sufficient to represent previous tasks as more data are received over time and that class imbalance problems could occur.

Generative Replay

Generative (pseudo)-replay approaches train a generative model to construct past images or to approximate their distributions. By learning on current data and generating past data, the algorithm ensures that the knowledge and skills from the past is not forgotten. The generative model is usually implemented through GAN [61, 93] or auto-encoder [29].

In this framework, the classification model and generative model are usually trained alternately, in a *dual-model* framework. In other words, the generative model will be frozen when the classification model is being trained with past experience sampled from generative model. After the classifier is trained, the generative model will then be updated. In this thesis we focus on conditional generators where the generator is required to learn and generate specific conditions. A potential application of generative replay methods is in situation where there is privacy constrains [14, 80] making raw-input replay not possible.

The idea of using generative models for incremental learning has been explored in deep generative replay (DGR) [80] and deep generative memory (DGM) [61] where the authors focus their effort at generating images for for replay. However, both DGR and DGM are limited to small images and to small datasets. Later, conditional adversarial network (CAN) [93], Generative Features Replay (GFR) [49], and FearNet [29] employed generative models at the image embedding level, enabling the methods to be scaled to larger images for incremental learning. Further, this allows them to achieve state of the art performance in several benchmark scenarios. In this thesis, we will focus on the comparison against CAN, GFR and FearNet as the proposed method in this thesis is a generator moment matching model for incremental learning.

Chapter 3

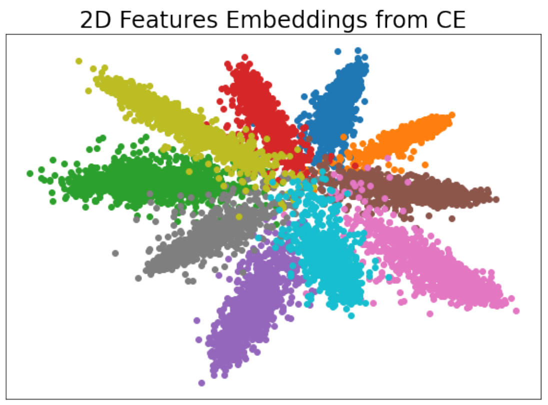
Method Development

In this chapter, we develop the Incremental Generative Replay Embedding (IGRE) framework for the Class Incremental Learning (CIL) scenario. In Section 3.1, we will visualize the feature embedding and provide intuition to understand the dynamics of catastrophic forgetting in the CIL scenario. In Section 3.2, we describe the use of feature extractors to reduce the dimensionality of the input. In Section 3.3, the alternating back-propagation method is employed as a method to train the generative model. Finally in Section 3.4, the incremental learning framework will be presented.

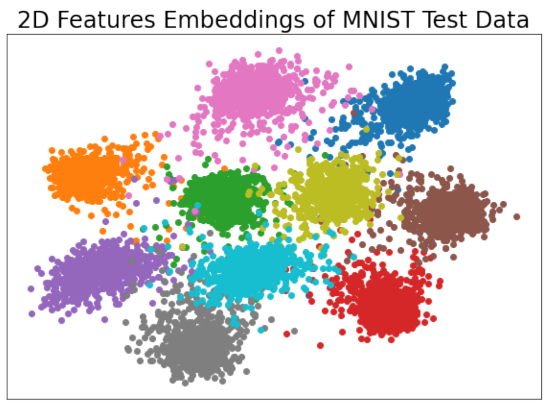
3.1 Problem Understanding

In this section, we aim to develop the intuition to understand the CIL problem [26] introduced in Section 2.3. We will visualize the feature embedding extracted from a deep neural network (DNN) trained in a relatively simple two-task CIL scenario without continual learning strategies. We aim to provide the intuition from which we develop our continual learning strategy from the visualization of feature embedding.

As introduced in Section 2.1, the feature extractor of DNNs is able to extract features from an image to solve a classification task. The work of [91] suggests that adjusting the bias at the classification layer is a simple way to combat catastrophic forgetting because the feature extractor remains stable under the CIL scenario. Inspired by [91], we visualize the feature embedding in order to have an idea about how task progression in CIL scenario impacts DNNs. We trained LeNet-5 [40], a DNN for classification, on the entire MNIST dataset [42] using cross-entropy (CE) loss from Equation (2.6), and extracted the feature embedding of the test data.



(a) Feature embedding from CE loss



(b) Feature embedding from triplet loss

Figure 3.1: Feature embedding of the MNIST test data when the model is trained with two different loss functions. (a) shows the feature embedding of data where the model is trained with cross-entropy loss. (b) shows the feature embedding of data where the model is trained with triplet loss.

The first obstacle is that the feature embedding is 128-dimensional which is difficult to visualize. We used t-SNE [87], an unsupervised learning dimensional reduction technique, to reduce the 128-dimensional feature embedding to 2-dimensional for visualization[1]. The dimensionality reduced feature embedding for subset of data can be seen in Figure 3.1-(a). Although the feature embedding trained with CE loss looks separable, many of the feature embeddings are overlapping at the center. Further, given that the validation accuracy on the test data point is 99%, the feature embeddings of the data are not well separated into clusters as is desirable. At this point there are two issues preventing us from further understanding the behaviour of DNN: the cross-entropy loss function and use of t-SNE for dimensionality reduction.

First, notice that although the CE loss from Equation (2.6) is good at training models for classification, it does not explicitly regularize the feature embedding of the same classes to be clustered together [76, 90]. A DNN trained with CE focuses on separating the logits of the model and implicitly regularize the feature embedding. Second, the use of t-SNE adds an additional layer of abstraction that may have caused the visualization to present more overlapping than there is. These two issues are solved with the use of the *triplet loss* [76, 90] from metric learning, where the loss encourages the model to learn image clusters

[1]The idea of using t-SNE to visualize feature embedding is reproduced by many open-sourced project. Similar visualization of feature embedding through t-SNE can also be found at this github page.

semantically in the feature embedding space. The use of triplet loss is to train the model is purely for visualization.

The triplet loss in [90] tries to bring the example, called the anchor, close to the positive images (images of the same label), and away from negative images (images of different label). The loss is given by

$$\mathcal{L} = \sum_{i=1}^{N} \left[\|\mathbf{f}_i^a - \mathbf{f}_i^p\| - \|\mathbf{f}_i^a - \mathbf{f}_i^n\| + \alpha \right]_+, \tag{3.1}$$

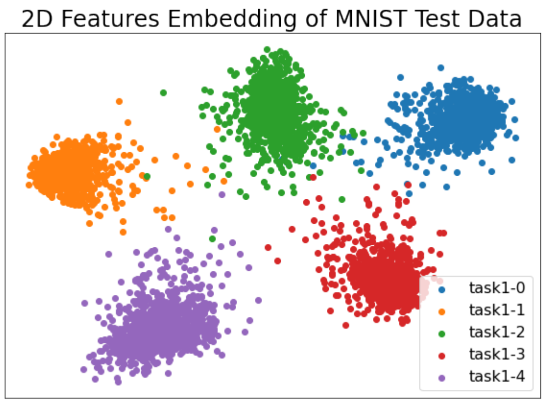
where $\mathbf{f}_i^a$ is the feature embedding of the anchor i, $\mathbf{f}_i^p$ and $\mathbf{f}_i^n$ is the feature embedding of the positive and negative pair for anchor i, and $\alpha > 0$ is a margin hyperprameter. As can be seen from Equation (3.1), the loss directly regularizes the feature embedding $\mathbf{f}$. More importantly, if we assume $\mathbf{f} \in \mathbb{R}^2$, we can directly visualize the learned feature embedding; no dimensionality reduction is needed after the training for visualization. We assert the assumption that $\mathbf{f} \in \mathbb{R}^2$ purely for visualization purpose.

The feature embedding of training data from the model trained with triplet loss are shown in Figure 3.1-(b). We can see the cluster formations are more prominent in the feature embedding space in Figure 3.1-(b) compared to Figure 3.1-(a). This verifies our intuition that the images are projected into the feature embedding space as clusters by the feature extractor of a DNN.

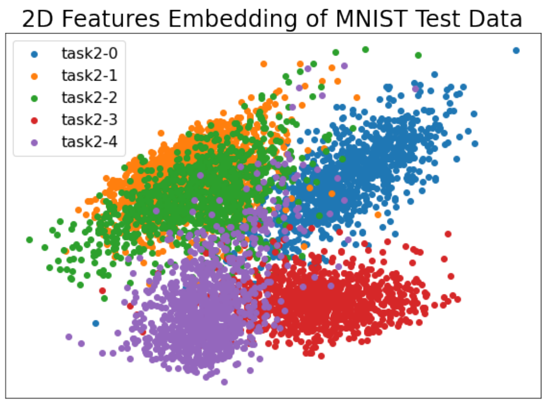
In order to visualize how the clusters change under the CIL scenario, we created two tasks from the MNIST digits dataset [42] by splitting the 10 classes into two groups, each group containing same number of disjoint classes. That is, task 1 consists of digits in $\{0, 1, 2, 3, 4\}$, and task 2 consists of digits in $\{5, 6, 7, 8, 9\}$. We present the groups separately to a LeNet-5 [40] model for learning. To maintain the interpretability of the feature embedding, the DNN is trained with triplet loss with 2-dimensional feature embedding.

We keep track of the test data from task 1 and analyse how the learned clusters evolve from task to task. In Figure 3.2-(a), we see each class has their own cluster embedding after task 1 training. The model is then presented with task 2 data for learning without any continual learning algorithm. We see in Figure 3.2-(b) that the clusters are grouped closer together. In particular, the clusters corresponding to digit 1 and digit 2 overlap; this suggests that the model suffers from catastrophic forgetting.

Intuitively, figures 3.1 and 3.2 showed that a DNN learns to group images from the same class together into the same cluster. When a new task is introduced in the CIL scenario, the DNN model suffers from catastrophic forgetting as indicated by the merging of clusters. An ideal continual learning algorithm is able to organize the clusters at feature



(a) Feature embedding after task 1.



(b) Feature embedding after task 2.

Figure 3.2: Feature embedding of the MNIST test data in the CIL scenario. (a) shows the feature embedding after the model is trained on task 1. (b) shows the feature embedding after the model is trained on task 2. This suggests that the model suffers from catastrophic forgetting of task 1 as the model fails to project task 1 test data back to the original cluster location.

embedding space such that inter-cluster interference is minimized when new clusters are added.

Next, we develop our intuition on how DNNs change under the CIL scenario into continual learning algorithm.

Problem Breakdown

Continuing from the intuition that the feature extractor of a DNN loses its ability to project images of previously seen classes to previously identified cluster locations, we develop two requirements. First, we need a model to describe the clusters in feature embedding space. Second, we need the cluster model to be able to generate the data distribution. Here, we will present the IGRE framework at a high-level, setting the stage for the detailed descriptions later.

The first requirement is that we need a model to describe the clusters. We satisfy this requirement by training a generator neural network to learn a map from a smooth latent space to the feature embedding space. That is, as introduced in Section 2.2, given an element from the smooth latent space, the generator neural network should be able to reconstruct the feature embedding in the feature embedding space. Therefore, we model the clusters in the feature embedding space through the latent space and generator neural network.

Second, we also require the generator to be able to model the data distribution in the CIL scenario. This requirement is satisfied through the conditioning the generator neural network with class labels. Using conditional generator, we can generate feature embedding of all classes as a generative replay mechanism for the classifier. As a result, we have a generative model that models the feature embedding data distribution.

3.2 Learning Feature Embedding

In this section, we describe our generative model for learning feature embedding of a single task in the CIL scenario. There are two main benefits for training a generative model for feature embedding instead of raw input. The first benefit is that the feature embedding has significantly fewer dimensions compared to raw input, which facilitates efficient generative model training. The second benefit is that it is (intuitively) easier to learn to separate the classes at feature embedding space as illustrated in Figure 3.1.

Recall in Section 2.3, in particular Equation (2.19), the training samples associated with the classes seen in task t in the CIL scenario:

$$\mathcal{X}^t = \Big\{ (\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^D,\, y_i \in \mathcal{Y}_t \Big\}_{i=1,\dots,m_t}. \tag{3.2}$$

The class label y_i is encoded in the class characteristic $\rho \in \mathbb{R}^K$, and that we define the conditional generative model G as

$$G : \mathbb{R}^d \times \mathbb{R}^K \to \mathbb{R}^D, \tag{3.3}$$

such that G takes in latent variable $\mathbf{z} \in \mathbb{R}^d$ with $d \ll D$ and $\rho \in \mathbb{R}^K$ and generates a new realistic sample $\tilde{\mathbf{x}}$ belonging to the set of samples $\mathcal{X}^t$.

The use of a generator neural network [46] to learn the data distribution through Equation (2.9) was introduced in Subsection 2.2.1. Here, we extend the moment matching framework to a conditional generator, thus the log joint likelihood becomes

$$\log P_\theta(\mathbf{x}, \mathbf{z} \mid \rho) = -\frac{\|\mathbf{z}\|_2^2}{2} - \frac{\big\|\mathbf{x} - G_\theta\big(\mathbf{z}, \rho\big)\big\|_2^2}{2\sigma^2} + c. \tag{3.4}$$

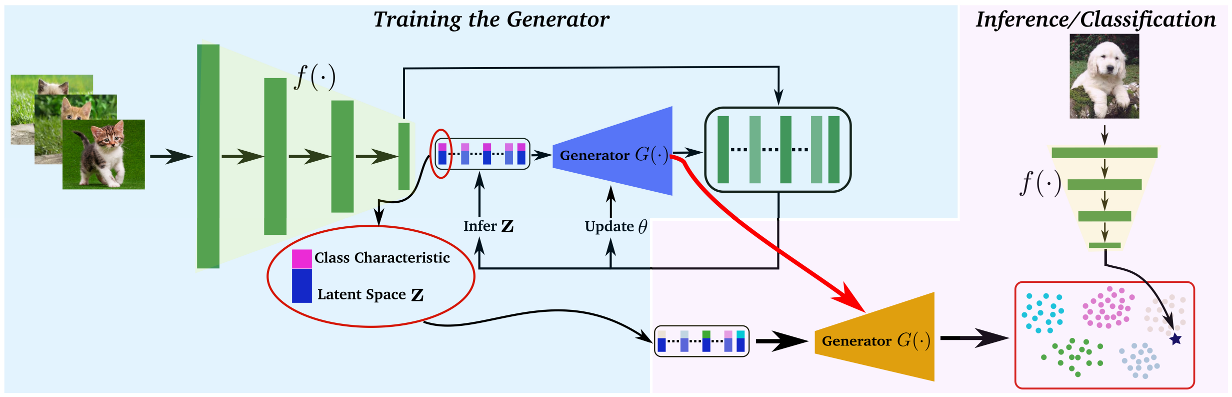


Figure 3.3: The proposed IGRE framework: The associated representation of each input sample is calculated by the feature extractor $f(\cdot)$. The generator model $G(\cdot)$ learns the feature embedding conditioned on class labels.

We can train a conditional generator, G_θ, by integrating out the latent variables and maximizing Equation (3.4) through gradient ascent with respect to θ, hence learning the conditional observed data distribution $P_\theta(\mathbf{x} \mid \rho)$. However, training G_θ is still computationally intensive because the gradient,

$$\frac{\partial}{\partial\theta}\log P_\theta(\mathbf{x} \mid \rho) = \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x},\rho)}\left[\frac{\partial}{\partial\theta}\log P_\theta(\mathbf{x},\mathbf{z} \mid \rho)\right], \tag{3.5}$$

is intractable in real-world applications due to the high dimensionality of $\mathbf{x}$. That is, we require the training data to contain enough variations to describe the space spans by $\mathcal{X}$.

The difficulty in learning G_θ associated with the high dimensionality of $\mathbf{x}$ can be mitigated through the use of a dimensionality reduction function,

$$f : \mathcal{X} \to \mathbb{R}^n. \tag{3.6}$$

Here f projects high-dimensional $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ into a modest n-dimensional embedding space, such that $n \ll D$. This f is realized in practice through the feature extractor of a DNN, and we call the space $f(\mathcal{X}) \subseteq \mathbb{R}^n$ the *embedding* space. The process can be seen in Figure 3.3, where f projects input images to embedding space. Thus, the difficulty in learning G_θ is reduced and G_θ can be learned efficiently.

The ideal $f(\cdot)$ is a many-to-few function which maps all samples belonging to a given class/task to a unique neighbourhood location in $\mathbb{R}^n$. However, this idealization is not possible, given the continual learning setup where all data are not seen simultaneously, making it difficult to partition the space properly [34]. As such the realization of clusters illustrated in Figure 3.1 is not feasible. Hence, we use a generator neural network to support the feature extractor by learning the data distribution at the embedding space incrementally through generative replay.

The key is that the replay step takes place in the *embedding* space, and not in the much larger original space. Thus the generator generates feature embedding for replay in the subspace of $\mathbb{R}^n$ which is far simpler than $\mathbb{R}^D$. As a result, the joint conditional probability density which we aim to learn is approximated by

$$P\left(\mathbf{x}, \mathbf{z} \mid \rho\right) \approx P\left(f(\mathbf{x}), \mathbf{z} \mid \rho\right), \tag{3.7}$$

where $f(\mathbf{x})$ is essentially a generic feature extractor that encodes the feature embedding associated with sample $\mathbf{x}$. Following the approximation in Equation (3.7), the conditional log joint likelihood for the model becomes

$$\log P_\theta(f(\mathbf{x}), \mathbf{z} \mid \rho) = -\frac{\|\mathbf{z}\|_2^2}{2} - \frac{\left\|f(\mathbf{x}) - G_\theta\left(\mathbf{z}, \rho\right)\right\|_2^2}{2\sigma^2} + c, \tag{3.8}$$

where c is independent of $\mathbf{x}, \mathbf{z}$ and θ. This means the conditional generator, G_θ, now learns to generate the feature embedding $f(\mathbf{x})$. In practice, we integrate out the latent variables $\mathbf{z}$ as they are not directly observed. Similar to the marginalization done at Equation (2.11), we integrate out the latent variables in Equation (3.8),

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{m} \log \int_z P_\theta(f(\mathbf{x}_i), \mathbf{z} \mid \rho_i)\, d\mathbf{z} && (3.9) \\ &= \sum_{i=1}^{m} \log P_\theta(f(\mathbf{x}_i) \mid \rho_i), && (3.10) \end{aligned}$$

where m is the number of data samples. Equation (3.10) is interpreted as the conditional observed data likelihood function which we want to maximize with respect to the generator's parameters θ, that is,

$$\arg\max_\theta \sum_{i=1}^{m} \log P_\theta(f(\mathbf{x}_i) \mid \rho_i). \tag{3.11}$$

In summary, we simplified the learning problem from $\mathbb{R}^D$ to $\mathbb{R}^n$ and adapted the generator for the lower dimension through Equation (3.8). The next section describes the learning algorithm to solve Equation (3.11).

3.3 Alternating Back Propagation

In this subsection, we describe the learning algorithm for the maximizing Equation (3.11). The algorithm is a simple alternating back propagation scheme [22] where the generative model is trained through the conditional sampling of latent variables, $\mathbf{z}$, and gradient ascent on θ. The gradients for learning the generator and inferential backpropagation are first derived, then we will describe the procedure to update the generator and latent variables iteratively.

Recall the conditional observed data log-likelihood, Equation (3.10), is obtained by integrating out the latent variables in Equation (3.8), that is,

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{m} \log \int_z P_\theta(f(\mathbf{x}_i), \mathbf{z} \mid \rho_i) P(\mathbf{z})\, d\mathbf{z} \\ &= \sum_{i=1}^{m} \log P_\theta(f(\mathbf{x}_i) \mid \rho_i). \end{aligned} \tag{3.12}$$

In other words, maximizing $L(\theta)$ with respect to θ is maximizing the likelihood of observing the data under θ. The maximization is done through alternating back propagation in a manner similar to expectation-maximization algorithm [58].

Generator Parameters θ

First, we describe the gradient calculation and update step for the generator parameters θ. The parameters are learned through gradient ascent. The gradient of Equation (3.12) with respect to θ, for a single observed pair[2] $(\mathbf{x}, \rho)$, is,

$$\frac{\partial}{\partial \theta} \log P_\theta(f(\mathbf{x}) \mid \rho) = \mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)} \left[\frac{\partial}{\partial \theta} \log P_\theta(f(\mathbf{x}), \mathbf{z} \mid \rho) \right]. \tag{3.13}$$

To see Equation (3.13) holds, we observe that,

$$\begin{aligned} &\mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)} \left[\frac{\partial}{\partial \theta} \log P_\theta(f(\mathbf{x}), \mathbf{z} \mid \rho) \right] \\ =&\mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)} \left[\frac{\partial}{\partial \theta} \log P_\theta(\mathbf{z} \mid f(\mathbf{x}), \rho) + \frac{\partial}{\partial \theta} \log P_\theta(f(\mathbf{x}) \mid \rho) \right] \\ =&\mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)} \left[\frac{\partial}{\partial \theta} \log P_\theta(\mathbf{z} \mid f(\mathbf{x}), \rho) \right] + \mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)} \left[\frac{\partial}{\partial \theta} \log P_\theta(f(\mathbf{x}) \mid \rho) \right]. \end{aligned} \tag{3.14}$$

[2] We omit the subscript i without loss of generality.

The first quantity in Equation (3.14) is zero because,

$$
\begin{aligned}
&\mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)}\left[\frac{\partial}{\partial\theta}\log P_\theta(\mathbf{z} \mid f(\mathbf{x}),\rho)\right] \\
=&\int p_\theta(\mathbf{z} \mid f(\mathbf{x}),\rho)\left[\frac{\partial}{\partial\theta}\log P_\theta(\mathbf{z} \mid f(\mathbf{x}),\rho)\right] d\mathbf{z} \\
=&\int \frac{\partial}{\partial\theta} P_\theta(\mathbf{z} \mid f(\mathbf{x}),\rho)\, d\mathbf{z} \\
=&\frac{\partial}{\partial\theta}\int P_\theta(\mathbf{z} \mid f(\mathbf{x}),\rho)\, d\mathbf{z} = \frac{\partial}{\partial\theta}1 = 0.
\end{aligned}
$$

Further notice that the function in the second quantity in Equation (3.14) is independent of $\mathbf{z}$ thus we have,

$$
\mathbb{E}_{p_\theta(\mathbf{z}|f(\mathbf{x}),\rho)}\left[\frac{\partial}{\partial\theta}\log P_\theta(f(\mathbf{x}) \mid \rho)\right] = \frac{\partial}{\partial\theta}\log P_\theta(f(\mathbf{x}) \mid \rho), \tag{3.15}
$$

which proves Equation (3.13) holds.

Since the expectation in (3.13) is analytically intractable, Monte-Carlo approximation is used to estimate the expectation [22, 100] with respect to current $\mathbf{z}$,

$$
\begin{aligned}
\frac{\partial L(\theta)}{\partial\theta} &\approx \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta}\Big[\log P_\theta\big(f(\mathbf{x}_i),\mathbf{z}_i \mid \rho_i\big)\Big] && (3.16)\\
&= \frac{1}{m}\left[-\sum_{i=1}^{m}\left[\frac{\partial}{\partial\theta}\frac{1}{2\sigma^2}\left\|f(\mathbf{x}_i) - G_\theta\Big(\mathbf{z}_i,\rho_i\Big)\right\|_2^2\right]\right], && (3.17)
\end{aligned}
$$

where Equation (3.8) is used to go from (3.16) to (3.17). The correctness of Equation (3.13) is guaranteed from the expectation-maximization algorithm [58]. It is important to note here that Equation (3.17) is formulated for batched learning with m examples, hence not scalable to large datasets with millions of examples as it is slow and costly for the algorithm to scan through the entire training set before taking a single step. Therefore, we use stochastic gradient with mini-batches [71] to estimate the gradient in Equation (3.17) as alternative optimization procedure so we can scale the method to large-scale datasets.

Once the gradient in Equation (3.17) is obtained, the parameters are updated by performing gradient ascent,

$$
\theta_{t+1} = \theta_t + \eta_t\frac{\partial L(\theta_t)}{\partial\theta}, \tag{3.18}
$$

where η_t is the learning rate at step t. Repeating (3.18) multiple times will guide the generator to generate a set of features that maximizes Equation (3.11) with a fixed set of latent variables $\mathbf{z}$. Since this is a Monte-Carlo approximation, the latent variables have to be inferred to prevent the algorithm from getting stuck at a local minima/maxima. We will describe how to infer the optimal $\mathbf{z}$ next.

Inferential Back Propogation

In addition to the generator, we need to also infer the latent variables to maximize (3.11). The continuous latent variable $\mathbf{z}$ is sampled based on Langevin dynamics [59], and updated iteratively:

$$\mathbf{z}_{j+1} = \mathbf{z}_j + \frac{s^2}{2}\frac{\partial}{\partial \mathbf{z}} \log P_\theta\Big(f(\mathbf{x}), \mathbf{z} \mid \rho\Big) + sN_j, \tag{3.19}$$

where s is the Langevin's step size and j is the time step for Langevin inference. $N_j \sim N(0, I_d)$ is the random noise projected in each step preventing the chain from being trapped.

Note that generating independent high-dimensional $\mathbf{z}$ from scratch for the Monte-Carlo approximation of Equation (3.17) is computationally expensive [59, 100]. In practice, we take L Langevin's steps for each data point at each iteration in order for the algorithm to search for an optimal latent variable representation given a fixed generator. That is, take L-steps according to Equation (3.19) to update $\mathbf{z}$, then at the next iteration, we do a warm restart for the sampling of $\mathbf{z}$ from previously updated $\mathbf{z}$. After inferential back-propagation for latent variable $\mathbf{z}$, the generator's parameters θ are then updated with the inferred latent variable with Equation (3.18).

The generator $G_\theta\big(\mathbf{z}, \rho\big)$ generates a new feature embedding given the latent variable $\mathbf{z}$ and class characteristic ρ. The ultimate goal is that the generated feature embeddings that follow the same distribution as $f(\cdot)$. The learning algorithm alternates between (3.19) and (3.18) for the model to learn the log joint likelihood in Equation (3.8). It has been shown [59] that the proposal step in (3.19) guides the latent variable $\mathbf{z}$ to a stationary distribution of our target, $P_\theta(f(\mathbf{x}), \mathbf{z} \mid \rho)$.

Having described the alternating back-propagation and gradient calculation to maximize (3.11), the next section summarizes the steps for class incremental learning.

3.4 Continual Learning Model

The alternating back propagation scheme described in Section 3.3 maximizes Equation (3.11). However, the maximization problem shown in Equation (3.11) is only applicable for $\mathcal{X}^t$ where t is fixed. In this section, we describe the IGRE framework, in which we adapt the alternating back propagation algorithm for the Class Incremental Learning (CIL) scenario. We will describe the steps of IGRE to combat catastrophic forgetting, that is, extending to $\mathcal{X}^t$ where $t \in \{1, \ldots, \tau\}$.

As described in Section 2.3, $\mathcal{X}^t$ is presented to a continual learning algorithm sequentially. More importantly, a model in the CIL scenario is allowed access to $\mathcal{X}^t$ as many times as it needs at task t for learning, but then never again thereafter. That means the training set $\mathcal{X}^t$ will not be accessible at future tasks $s > t$ unless the algorithm decides to store a subset of $\mathcal{X}^t$ for replay, as such the model will suffer catastrophic forgetting. The IGRE framework consists of a class conditional generator neural network and a single-headed classifier. The generator neural network generates class feature embedding for replay to combat catastrophic forgetting, while the classifier is trained with the generated feature embedding for classification. The steps are summarized in Algorithm 1.

At task t, the raw inputs in $\mathcal{X}^t$ are first projected into the embedding space via feature extractor $f(\cdot)$ which is fixed (e.g., an ImageNet pre-trained model) for all tasks/classes. We denote the resulting projected embedding as D^t. The previously trained generator, G^{t-1}, will be used to generate features from previously seen tasks, that is $D^{1,\ldots,t-1}$. The new feature embedding dataset, the combination of D^t and $D^{1,\ldots,t-1}$, is used to train the generator at task t via the alternating back-propagation, switching between (3.19) and (3.18). In other words, we incrementally maximize

$$\arg\max_{\theta} \log P_{\theta}(f(\mathbf{x}) \mid \rho), \tag{3.20}$$

where $\mathbf{x} \in \mathcal{X}^t$, $t \in 1, \ldots, \tau$, and the $\mathcal{X}^t$'s are presented sequentially.

In the final step, the classifier is updated by the generated feature embedding conditioned on different class characteristics ρ's through G^t. This is achieved through formulating the generator as a class conditional generator. In Section 4.5, we show experimentally that the classifier can be a linear model (e.g., linear-SVM or logistics regressor) since the embedding space of the learned generator is sufficiently separable and efficient.

Algorithm 1: Alternating back-propagation within the IGRE framework.

```
Input  : Training set at task $t$ $\mathcal{X}^t = \{(\mathbf{x}_i^t, \rho),\ \ i = 1, \ldots, m\}$
         Langevin's step size, $s$
         Model's standard deviation, $\sigma$
         Number of Langevin's step, $L$
         Number of training epochs, $I$
Output : The parameters $\theta$ in generator $G^t(\cdot)$
         Inferred latent variable $\mathbf{z}$
if $t == 1$ then
    Generate embedding train set $D^t$ via $f(\cdot)$.
    for $i \leftarrow 1$ to $I$ do
        for $j \leftarrow 1$ to $L$ do
            Update the latent variable $\mathbf{z}$ via (3.19).
        end
        Update $\theta$ via (3.18).
    end
end
for $t \leftarrow 2$ to $\ldots$ do
    Generate embedding train set $D^t$ via $f(\cdot)$.
    Create train set $D^{1,\ldots,t-1}$ by generating embedding for tasks/classes
      $\{1, \ldots, t-1\}$ via the learnt generator $G^{t-1}(\cdot)$.
    Combine the train sets $\left\{D^{1,\ldots,t-1} \cup D^t\right\}$.
    for $i \leftarrow 1$ to $I$ do
        for $j \leftarrow 1$ to $L$ do
            Update latent variable $\mathbf{z}$ via (3.19).
        end
        Update $\theta$ via (3.18).
    end
end
```

Chapter 4

Experimental Results

In this chapter, we will present our analysis of the proposed Incremental Generative Replay Embedding (IGRE) framework. IGRE was evaluated with common benchmark datasets in machine learning literature, and ablation studies were conducted on different components of the IGRE framework.

4.1 Datasets and Evaluation

In this section, we describe the datasets and the evaluation metrics we used for the experiments. As mentioned in Section 2.3, we split a dataset into disjoint groups of classes and present the groups sequentially to the model to simulate the class incremental learning scenario.

Datasets

Three different publicly available datasets are used to evaluate the proposed method. The datasets are CIFAR-10/100 [37], Caltech-UCSD Birds-200-2011 (CUB) [89], and ImageNet2012 [13, 72]. These datasets are common benchmarks within the continual learning community, thus allowing us to compare our proposed method against other methods.

CIFAR-10/100 [37] is a small scale dataset consisting of $60,000$ 32×32 RGB-images over 10 or 100 classes. The data are separated into $50,000$ training samples ($5,000$ per class label) and $10,000$ test samples for the 10-class variant, whereas the 100-class variant has only 500 training samples per class label.

CUB [89] is a medium scale fine-grained dataset consisting of 224×224 images of 200 bird species with 312 class-level attributes. This dataset is included to evaluate the proposed IGRE framework when class attributes are used as the conditional variables, ρ, rather than the class labels. In addition to having more class attributes, this dataset also contains imbalanced numbers of examples per-class, allowing us to evaluate IGRE's robustness in the imbalanced class situation.

ImageNet2012 [13] is a popular large scale dataset within the computer vision community, consisting of 13 million natural images with 1000 classes. The large number of classes in this dataset allows us to evaluate IGRE in large-scale long learning sequences. In addition to using all of the classes, we also used a subset of the classes to form SubImageNet [49, 85] which is used to compared against other CIL algorithms.

Evaluation

All algorithms are evaluated in the class incremental learning scenario for image classification. Half of the classes in a dataset[1] are randomly selected as the base task, task 0, for model initialization, and the remaining half is split into τ disjoint groups. The groups are then presented to the model sequentially for the model to learn incrementally. Two evaluation metrics are used to measure the performance of the proposed method and competing algorithms:

1. Average accuracy across all tasks, denoted as A_t, and
2. Knowledge retention of the base task (task 0) at task t, denoted as B_t.

These two metrics are similar to those used in [29, 93]. Let $a_{t,j}$ denotes the accuracy of the model for task j, evaluated when the model is trained for task t, where $j \leq t \leq \tau$:

- The average accuracy at task t is calculated as

$$A_t = \frac{1}{t} \sum_{j=1}^{t} a_{t,j} \, , \tag{4.1}$$

  which measures how effective a continual learning method is at classification across a sequence of tasks. The reason the calculation for $a_{t,j}$ starts at task 1 is that the learning sequence actually starts at $t = 1$ and not $t = 0$.

[1]This base task initialization is used for all datasets except CIFAR10.

Table 4.1: A hypothetical scenario with two models and two tasks. Model A and B both achieved $A_2 = 0.5$ but Model B has less forgetting than Model A as Model B has higher $a_{2,1}$.

Model	$a_{2,1}$	$a_{2,2}$	A_2
A	0	1	0.5
B	0.5	0.5	0.5

- The knowledge retention of base task is calculated as

$$B_t = \frac{1}{t}\sum_{j=1}^{t} a_{j,0}\,, \tag{4.2}$$

  measuring the model's knowledge retention of the base or initialization task over the next t further learning sessions.

Since A_t is an averaged value, we are not able to evaluate how well the learning algorithm is retaining knowledge from earlier tasks. For example, consider a hypothetical scenario with two models with $t = 2$, and the models performance listed in Table 4.1. We see both models achieved $A_2 = 0.5$. Since model A obtained a score of $a_{2,1} = 0$ due to catastrophic forgetting of task 1, we can infer that model A also obtained $a_{2,0} = 0$. In this case, model A is not retaining any knowledge from previous task yet achieving the same score as model B. Therefore, we use one additional metric B_t to give us an indication how well the learning algorithm is able to retain knowledge from the base initialization task.

All experiments are repeated 5 times to get the mean and variance of the performance metrics. Note that the numbers reported for IGRE is always the average over 5 runs. All results from competing methods are obtained from the original publications.

4.2 Class Incremental Learning

We will first present the results for split-CIFAR-10, split-CIFAR-100, SubImageNet and split-ImageNet where class labels are used as conditional variables. These four benchmark scenarios for class incremental learning evaluation are listed in increasing difficulty.

Split-CIFAR-10

In split-CIFAR-10, the 10 classes are split into groups containing equal numbers of disjoint classes and presented to the model; there is no base task in this scenario because there are many examples per class. We compared the proposed method to conditional channel-gated network (CGN) [1] and dynamic generative memory (DGM) [61] in Split-CIFAR-10. We will briefly describe CGN and DGM and then present the comparison in Table 4.2.

The CGN of [1] is a replay method where the algorithm jointly optimizes for class and task predictions, that is,

$$\max_{\theta} \mathbb{E}_{t\sim\mathcal{T}} \left[\mathbb{E}_{(\mathbf{x},y)\sim T_t} \log \mathbb{P}_\theta(y, t \mid \mathbf{x})\right] = \tag{4.3}$$

$$\mathbb{E}_{t\sim\mathcal{T}} \left[\mathbb{E}_{(\mathbf{x},y)\sim T_t} \left(\log \mathbb{P}_\theta(y \mid \mathbf{x}, t) + \log \mathbb{P}_\theta(t \mid \mathbf{x})\right)\right]. \tag{4.4}$$

The term $\log \mathbb{P}_\theta(t \mid \mathbf{x})$ computes the log-probability that an image is from task t. The predicted task t is then used to compute $\log \mathbb{P}_\theta(y \mid \mathbf{x}, t)$, the log-probability of the class label of input image $\mathbf{x}$.

In this joint objective, the term $\mathbb{P}_\theta(t \mid \mathbf{x})$ represents a task classifier which predicts the task from observations. As such, the model is able to choose the class prediction from the appropriate task head, thus reducing the prediction difficulty. A replay mechanism with coreset is used to rehearse the task classifier. However, CGN requires well-defined task information during training to learn the appropriate classification head.

In DGM [61], the authors proposed two masking mechanisms for the training of generative models in the CIL setup. The first mechanism is called layer activation masking (DGMa), where an activation mask is learned to find a subset of parameters optimal for the task. The second mechanism is a binary mask imposed on the weights of the generator (DGMw). In both variants of DGM, a GAN was used to generate images from past tasks to combat catastrophic forgetting and consolidate the model's past knowledge.

In Table 4.2, we see that the proposed IGRE framework outperforms the other algorithms in both A_5 and A_{10} scenarios. IGRE outperforms CGN despite CGN being a task-aware method; this suggests that IGRE is able to utilize the class information in a more efficient manner compared to CGN. DGMa and DGMw use a GAN with an encoder to generate past image examples; this requires DGM to need a wide variety of images in order to learn the data distribution. That is DGM learns to generate replay examples at $\mathcal{X}$ which has much larger dimensionality than $f(\mathcal{X})$, hence needing more examples in $\mathcal{X}$ to learn better. Thus, it is challenging to scale both CGN and DGM to the more difficult scenario of split-CIFAR-100.

Table 4.2: Comparisons for split-CIFAR-10: The proposed IGRE is compared with DGM [61], a generative replay method, and CGN [1], a task-aware replay method. Our proposed IGRE outperforms these state-of-the-art methods in both 5-task (A_5) and 10-task (A_{10}) scenarios. Assessments of competing methods are extracted directly from their original papers.

Method	A_5	A_{10}
DGMw [61]	0.725	0.562
DGMa [61]	0.719	0.518
CGN [1]	-	0.701
IGRE	0.829	0.816

Table 4.3: Comparisons for split-CIFAR-100: The proposed IGRE framework significantly outperforms competing methods in longer sequences of tasks, while providing comparable results for shorter sequences.

Method	B_{25}	A_{25}	B_{10}	A_{10}	B_5	A_5
EWC [33]	0.102	0.154	0.144	0.179	0.163	0.187
iCARL [67]	0.506	0.528	0.554	0.572	0.571	0.595
FearNet [29]	0.547	0.569	0.613	0.625	0.648	0.662
CAN [93]	0.562	0.580	0.619	0.631	0.644	0.671
IGRE	0.662	0.643	0.675	0.653	0.688	0.659

On the other hand, IGRE generates past features with only a class conditional generator showing the benefits of not having an encoder and replaying at the feature embedding space. This allows IGRE to be more data efficient and scale to more difficult scenarios.

Split-CIFAR-100

In split-CIFAR-100, we split the classes evenly into groups of 2, 5 and 10 to simulate longer learning sequences while keeping the first 50 classes as a base task, as described in [93]. The same pre-trained ResNet-50 network architecture is used as the feature extractor $f(\cdot)$. At test time, we generate 300 samples from each class and train a single-layer classifier for all classes.

The split-CIFAR-100 scenario is a more difficult scenario compared to split-CIFAR-10 because there are longer task sequences and fewer training samples per class, thus all methods generally achieve lower accuracy. The proposed IGRE framework is compared against EWC [33], iCARL [67], FearNet [29], and CAN [93], with a particular interest in FearNet and CAN, as they similarly replay on an image embedding, but with a different mechanism.

As seen in Table 4.3, despite using only feature descriptors, IGRE is able to produce results comparable to other state-of-the-art features generative replay methods with an average accuracy of 64.3% for a longer (25-task) sequence. While IGRE and CAN [93] both perform modeling at the embedding space, the proposed IGRE framework outperforms CAN in longer learning sequences. The main advantage of IGRE is that it employs a generative model without an encoder, thus lessening the catastrophic forgetting of the entire system and enabling the system to learn longer task sequences.

In addition, IGRE shows better knowledge retention rate compared to other methods. This is seen from IGRE having higher B_τ compared to all other methods; IGRE is able to achieve B_{25} of 66.2% while the next highest is 56.2% from CAN. However, IGRE is not as plastic for the shorter learning sequence as IGRE achieved only 65.9% for A_5 while CAN and FearNet achieved 67.1% and 66.2% respectively.

We also compare IGRE with SOTA exemplar based methods and with a different backbone architecture (ResNet-18). Topological Preserving Class Incremental Learning (TPCIL) [85] is an algorithm which models the topology of feature embedding for class incremental learning. Learning Unified Classifier Incrementally via Rebalancing (LUCIR) [25] and LUCIR-M [50] are both exemplar based algorithms with LUCIR-M optimized on the exemplar selection. Similarly, we see in the split-CIFAR-100 column of Table 4.4 that IGRE is able to consistently outperform the exemplar CIL methods in longer task sequence scenarios, A_{25} and A_{10}, because the generator is able to generate feature embeddings that are highly discriminative.

In addition to examplar methods, we compared IGRE to Generative Features Replay (GFR) [49] designed for class incremental learning. GFR uses a GAN model for feature replay with a smaller backbone compared to CAN. We see IGRE consistently outperforms GFR in all scenarios suggesting IGRE is more resilient to data distribution shift compared to GFR, hence able to achieve higher overall average accuracy.

ImageNet

In this section, we present results using the ImageNet2012 [13] dataset. The first scenario is SubImageNet [50, 85], where only a subset of the classes in ImageNet is used for the experiment. The SubImageNet scenario is gaining popularity in the continual learning community [49, 50, 91] because it poses the challenge of learning from images of different

Table 4.4: Comparison with exemplar based methods using a ResNet18 backbone. iCARL, TPCIL, LUCIR and LUCIR-M are exemplar based CIL methods, and GFR is generative feature replay method. All competing results are extracted from respective original papers.

Method	Split-CIFAR-100			SubImageNet			ImageNet		
	$\mathbf{A_{25}}$	$\mathbf{A_{10}}$	$\mathbf{A_{5}}$	$\mathbf{A_{25}}$	$\mathbf{A_{10}}$	$\mathbf{A_{5}}$	$\mathbf{A_{25}}$	$\mathbf{A_{10}}$	$\mathbf{A_{5}}$
iCARL [67]	0.482	0.527	0.571	0.529	0.599	0.654	0.431	0.469	0.515
TPCIL [85]	-	0.636	0.653	-	0.748	0.763	-	0.629	0.649
LUCIR [25]	0.575	0.602	0.632	0.614	0.683	0.708	0.566	0.613	0.643
LUCIR-M [50]	0.610	0.623	0.633	0.697	0.714	0.726	0.610	0.630	0.645
GFR [49]	0.400	0.520	0.530	0.581	0.623	0.656	0.535	0.546	0.553
IGRE-ResNet18	0.621	0.625	0.629	0.650	0.675	0.681	0.538	0.550	0.559

resolutions yet not as many classes as full ImageNet. In both SubImageNet and full ImageNet, we trained a ResNet-18 from scratch with base task data to have a fair comparison with other methods.

In the SubImageNet scenario, we select 100 classes from the ImageNet [13] dataset and split these 100 classes to simulate CIL scenario with 5, 10 and 25 learning sessions. Half of the selected 100 classes were used as the base task where we train our feature extractor with these 50 classes, and the remaining 50 were split up into groups for incremental learning. The results are summarized in Table 4.4.

Using ResNet-18 as the feature encoder in IGRE framework, we were able to achieve A_{25} of 65.0%, which is 6.9% higher than GFR which also uses generative features replay. We see that IGRE outperforms GFR in all learning sequence lengths given similar amount of computing resources. This shows that IGRE is capable of learning feature embedding of higher resolution images and able to outperform similar generative features replay algorithms.

We also included comparison with state-of-the-art CIL algorithms [25, 50, 67, 85] which use raw input images for coreset replay for comparison. These raw input replay algorithms are listed above the solid line in Table 4.4. It is important to note that the IGRE framework achieved higher A_{25} than some raw input replay methods (LUCIR and iCARL) while using less memory resources. Although the raw input algorithms are able to achieve much higher A_{τ}, they require much larger disc memory for the coreset.

The results for large scale experiments with full ImageNet are listed at the right most column of Table 4.4. We see that the IGRE framework is able to achieve comparable performance to GFR in all learning sequence length, yet having simpler training complexity

Table 4.5: Average accuracy on the CUB dataset: The proposed method outperforms the state-of-the-art algorithms in all task sequence lengths.

Method	B_{50}	A_{50}	B_{20}	A_{20}	B_{10}	A_{10}
iCARL [67]	0.400	0.443	0.473	0.496	0.495	0.505
FearNet [29]	0.440	0.478	0.504	0.527	0.553	0.533
CAN [93]	0.468	0.497	0.527	0.549	0.562	0.576
IGRE	0.664	0.583	0.665	0.600	0.657	0.611

compared to GFR. This shows the effectiveness of the framework for generative replay. On the other hand, through using more memory for data buffer, raw inputs replay methods can outperform generative replay methods in all scenarios when using full ImageNet. Detailed analysis on the memory footprint of IGRE can be found in Section 4.6.

4.3 Class Attribute Learning

The conditional variables in the IGRE framework are able to be extended beyond class labels. In this section we present experimental results using class attributes as conditional variables instead of class labels. We use the CUB dataset and simulate the split-CUB scenario for class attribute learning, demonstrating that the IGRE framework can also be used with class attributes as the conditional variables.

The features are extracted from ResNet101 pre-trained on ImageNet and class attributes are obtained as described in [92]. There are 312 attributes in the CUB dataset, with the attributes annotated by human experts [89]. Some examples of the attributes are bird size, bird crown colour, and bird head pattern. Similar to split-CIFAR, we simulate split-CUB by creating 10, 20 and 50 tasks with 10, 5 and 2 disjoint classes in each task, respectively, with the first 100 classes as the base task. IGRE then learns the classes incrementally without any task boundary knowledge. All results are obtained as an average over 5 experimental runs.

We observe that A_t is generally decreasing in Figure 4.1 as classes are added, but IGRE consistently outperforms both CAN and FearNet. The results for the split-CUB class incremental learning scenarios can be seen in Table 4.5. The results suggest that IGRE is able to effectively use class attributes for continual learning, which we believe stems from the class attributes allowing IGRE to generalize to unseen examples of the same classes better than competing methods.

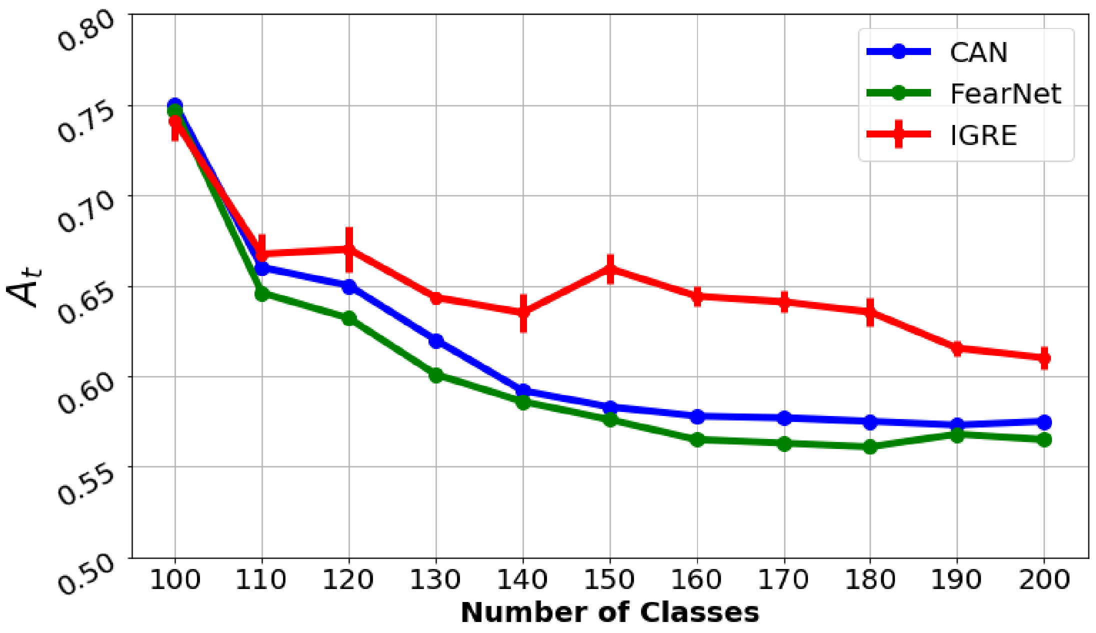


Figure 4.1: Average accuracy for all seen classes (A_t) comparing the proposed IGRE framework and competing methods CAN [93] and FearNet [29] on the split-CUB dataset. The proposed IGRE framework better maintains accuracy over tasks, and one benefit of IGRE is to reinforce knowledge and even improve the average accuracy while seeing more tasks.

Table 4.6: The hyper-parameters used for CIFAR10/100 and CUB learning scenarios.

	CIFAR-10		**CIFAR-100**			**CUB**		
	A_5	A_{10}	A_5	A_{10}	A_{25}	A_5	A_{10}	A_{25}
Epoch I	30	30	70	110	110	70	70	70
Optimizer	Adam		Adam			Adam		
Learning Rate for θ	0.0002		0.0002			0.0002		
Learning Rate for $\mathbf{z}$	0.0001		0.0001			0.0001		
σ	0.1		0.1			0.1		
s	0.3		0.3			0.3		
L	10	10	10	15	20	20	20	20

4.4 Implementation Details

To allow for reproducibility of results, in this section we list all the hyperparameters that were used to produce the results in Tables 4.2, 4.3, 4.4 and 4.5. The hyperparameters in the IGRE framework are the number of epochs I for each incremental task, learning rates for θ and $\mathbf{z}$, σ from Equation (3.8), Langevin's step size s from Equation (3.19) and the number of Langevin steps L. All experiments are conducted in the PyTorch [63] framework with the gradients calculated through their automatic differentiation implementation [63], and the Adam [30] optimizer was used for the optimization procedure.

We present the hyperparameters based on complexity of the dataset. The hyperparameters for experiments based on CIFAR10/100 and CUB are presented together as a group in Table 4.6. Images were resized to 224×224 before extracting feature embedding to ensure consistent usage of the feature extractor. The feature embedding is 2048-dimensional and we designed the generator neural network to be a three layers fully connected network with 2048 neurons in the hidden layer. Leaky ReLU [94] with parameter 0.2 was used as the non-linear activation function for the outputs of each layer.

In SubImageNet and ImageNet experiments, we trained a ResNet-18 [23] from scratch using half the total number of classes in the experiment as base task. In addition to the standard ImageNet training augmentation of random resized crop and horizontal flip [13, 23, 82], we also use Mix-up [98] augmentation, starting learning rate of 0.001 with cosine annealing with warm restart [52] and Adam optimizer were used to train the ResNet-18 from scratch. The trained feature extractor is then used for the IGRE framework.

Similarly, the images from ImageNet dataset were resized to 224×224 before extracting feature embedding to ensure consistency with the training procedure. The generator neural

Table 4.7: The hyper-parameters used for different ImageNet learning scenarios.

	SubImageNet			**ImageNet**		
	A_5	A_{10}	A_{25}	A_5	A_{10}	A_{25}
Base task epoch	80	80	80	120	120	120
Epoch I	80	80	80	90	100	100
Optimizer		Adam			Adam	
Learning Rate for θ		0.0002			0.0005	
Learning Rate for $\mathbf{z}$		0.0001			0.0002	
σ		0.1			0.1	
s		0.3			0.3	
L	40	40	40	30	25	30

network in this case is a four layers fully connected network with 1024 neurons in the hidden layers. Leaky ReLU [94] with parameter 0.2 was used as the non-linear activation function for the outputs of each layer. In order for the generator to converge faster we start off the learning rate for θ at 0.0005 and reduce it to 0.0003 and to 0.0002 after 20% and 60% of total iterations, respectively.

Table 4.7 shows the hyperparameters used in different scenarios to train the generator through the IGRE framework. Due to the large number of classes in task 0 for ImageNet, we needed to train the generator for more epochs for the generator to learn the feature embedding, this is reflected as one additional row for base task epoch in Table 4.7. It is important to note that although fewer Langevin steps L were taken for ImageNet experiments, there are more epochs I for each incremental step compared to SubImageNet experiments. The main reason for taking less L is that we want to reduce the run time per experiment to within five days.

4.5 Classifier Module

All experiments presented up to this section were conducted via a linear classifier trained end-to-end with stochastic gradient descent [71]. Recall from Section 2.6 that the generator is trained separately in the dual-model framework, we show that the generated feature embedding can be used with other kind of classifiers such as logistic regression and Support Vector Machine (SVM). This section aims to show that the generated features are highly transferable.

Table 4.8: A test on different choices of classifier applied to the IGRE framework on split-CIFAR-100 with 10 learning sequence scenario. The performance of the IGRE framework is not particularly dependent on the choice of classifier, demonstrating the effectiveness of the embedding space learned by the generator model.

Classifier	A_{10}
Linear Classifier	$0.653 \pm (0.007)$
Linear-SVM	$0.656 \pm (0.002)$
Logistic	$0.652 \pm (0.002)$
MLP (single hidden layer)	$0.646 \pm (0.003)$

Here we investigate the impact of the choice of classifier on IGRE's performance. To this end, we evaluate different linear and non-linear classifiers in the split-CIFAR-100 scenario with a learning sequence of 10 tasks. Similar to the split-CIFAR-100 experiments (Table 4.3), a pre-trained ResNet50 is used as the feature extractor $f(\cdot)$ for all classifiers.

We see in Table 4.8 that the linear classifier, SVM with a linear kernel, and logistic regression all achieve similar performance. This demonstrates the flexibility of the learned embedding space as it is not constrained to one type of linear classifier.

In addition, we trained a single hidden layer multi-layer perceptron (MLP) with ReLU as non-linear activation function to demonstrate that a non-linear classifier can also be used for the task. We do see a slight performance drop for the non-linear classifier as it overfits to the training data during cross-validation. Despite overfitting, the non-linear classifier was able to achieve similar performance ($\sim$65%) for A_{10}. This implies that the embedding space learned by the generator model is highly effective and sufficiently discriminating such that a linear classifier can categorize the test samples with accuracy better than using a non-linear classifier.

4.6 Computational Resources

One of the motivations for using the IGRE framework for CIL is that this framework is less memory intensive compared to the other raw input replay methods. We will compare IGRE's memory requirements to typical raw input replay methods, which store around 20 to 30 images per class [49, 50, 85] for replay.

It is important to recall from Chapter 3 that training a generator to generate a feature embedding in $\mathbb{R}^{2048}$ is significantly easier compared to generating an image in $\mathbb{R}^{3\times224\times224}$.

Table 4.9: Memory and computation resources requirements for various state-of-the-art methods. Epochs for replay methods are used to train a model end-to-end, but epochs for generative replay methods are used only for training the generative model.

Method	Dataset	Coreset	Backbone	Generative Model	Epochs
Replay	SubImageNet	2000 (400 Mb)	44 Mb	-	90
	ImageNet	20000 (3.8 Gb)	44 Mb	-	90
CAN [93]	Small	-	90 Mb	8.0 Mb	90
FearNet [29]	Small	-	90 Mb	8.0 Mb	80
GFR [49]	All	-	44 Mb	4.5 Mb	201
IGRE	SubImageNet	-	44 Mb	8.0 Mb	80
IGRE	ImageNet	-	44 Mb	14 Mb	90

Further, a feature vector with 2048-dimensions takes up approximately 8 kilobytes of memory, whereas a 224×224 RGB-image (having 150k pixels) takes up approximately 80-150 kilobytes, approximately an order of magnitude more than a feature vector. As such, a given system capable of storing a few thousand images (via conventional replay), would require around 400 megabytes of disc space storage whereas the generator we use requires a maximum of 18 megabytes.

In the large scale ImageNet and SubImageNet experiments, TPCIL [85], LUCIR [25] and LUCIR-M [50] outperformed IGRE in A_{10} and A_5 as seen in Table 4.4, but they require approximately 2-4 gigabytes of memory for their data buffer alone. On the other hand, IGRE requires around 14 megabytes for the generator neural network, 1 megabyte for the latent vectors, and 44 megabytes for the ResNet-18 feature encoder, a total of 59 megabytes of memory, which is significantly less than a replay method requires. The breakdown of memory usage of different methods[2] can be seen from Table 4.9. The small dataset in Table 4.9 refers to CIFAR-10/100 and CUB, and the 90 megabyte backbone refers to ResNet-50. In addition to using less memory, IGRE was able to produce comparable performance in the longer learning sequences of A_{25}.

On the other hand, the downside of using generative replay methods is the long training time needed for the generative model to converge. This is a general characteristic of deep generative models as introduced in Section 2.2. All generative replay methods investigated in this thesis, IGRE, CAN [93], FearNet [29] and GFR [49], reported significantly higher number of epoch compared to conventional replay. We see in Table 4.9 that generative

[2]The numbers are estimated through PyTorch, the actually memory usage may vary for different systems.

Table 4.10: Average accuracy of the proposed IGRE on split-CIFAR-10 (A_5) and split-CIFAR-100 (A_{10}) when different numbers of generator replay samples are used in training. Increasing the number of samples can improve the performance of the model in both datasets.

Number of Samples	CIFAR-10	CIFAR-100
50	$0.786 \pm (0.004)$	$0.550 \pm (0.012)$
100	$0.803 \pm (0.003)$	$0.621 \pm (0.004)$
300	$0.829 \pm (0.001)$	$0.653 \pm (0.007)$
500	$0.835 \pm (0.003)$	$0.661 \pm (0.003)$
600	$0.837 \pm (0.002)$	$0.662 \pm (0.005)$

replay methods used 80-201 epochs to train the generative model alone, whereas replay-based methods used around 90 epochs to train the model end-to-end. We observed the trade-off between generative replay methods and replay methods being the memory usage and compute time.

Number of Replay Examples

In this section, we show our results on the number of generated replay examples. Since generative replay methods use less memory, we investigate whether the performance of IGRE will improve along with more generated replay examples. We conducted the experiment in the split-CIFAR-10 and split-CIFAR-100 scenarios.

We see from Table 4.10 that the more samples we generate for replay, the better the performance. However, the performance saturates beyond 500 examples. We speculate that is the upper limit which IGRE can achieve for the given scenario and setup. Even with 600 generated replay examples, IGRE is still using less memory than a conventional replay method as seen in Table 4.9. This shows IGRE has the capacity to reconstruct the data distribution at the image embedding level and be a competitive CIL method.

4.7 Dependency on Feature Extractor

In the IGRE framework, generative replay is performed at the feature embedding space, where ResNet50 was used to produce the results presented in Table 4.2 and 4.3. We now investigate the effect of feature extractor $f(\cdot)$ on IGRE performance. We use VGG [82], ResNet [23] and EfficientNet [84] as a basis of comparison.

Table 4.11: Average accuracy of the proposed IGRE as a function of architecture for the generator G. The use of a pre-trained network architecture with better performance on the original task (i.e., pre-trained on ImageNet) can improve the performance.

Feature Extractor $f(\cdot)$	ImageNet Top-1 Acc	CIFAR-10 A_5	CIFAR-100 A_{10}
VGG16 (4096D)	0.704	0.69	0.46
ResNet18 (512D)	0.718	0.80	0.63
ResNet50 (2048D)	0.771	0.83	0.65
ResNet101 (2048D)	0.789	0.87	0.69
EfficientNet-b7 (2560D)	0.840	0.89	0.70

We simulate the split-CIFAR-10 and split-CIFAR-100 scenarios, leading to A_5 and A_{10}, respectively. Table 4.11 shows the results, from which it is worth noting that larger / more generalized networks used as feature extractors can significantly improve the IGRE performance. Despite the increase in Top-1 accuracy from ResNet101's 79% to EfficientNet-b7's 84%, IGRE's performance in split-CIFAR-100 yielded only marginal improvement. This suggests IGRE becomes invariant to the choice of feature extractor as the feature extractor becomes specialized in the pre-train task.

Table 4.11 also shows a strength of the proposed IGRE, that it is capable of using a variety of different feature extractors and feature embeddings to mitigate the catastrophic forgetting problem. The framework is able to learn the feature embedding space imposed by different feature extractors.

In addition to the ablation study on the feature extractor, we also investigate how the dimensionality of the feature descriptor from a feature extractor affects IGRE's performance. We conducted an experiment using the split-CIFAR-100 scenario and feature embedding from ResNet50. We calculate the principal components of the training features and run ten tests, keeping features in 10% increments from 10% through 100%.

The results for A_{10} are reported as a function of the percentage of features in Figure 4.2. The results are monotonic, as expected, and we observe that A_{10} starts to plateau after $70 - 80\%$ of the components, suggesting that, with care, an appropriately selected dimensionality reduction may be able to maintain our current results in a lower dimensional feature embedding space. This is further supported by the observation that the feature embedding from ResNet50 is 2048 dimensional, and the feature embedding from ResNet18 is 512 dimensional, yet IGRE achieved A_{10} of 62.5% using ResNet18's feature descriptor, while 30% of ResNet50's feature descriptor only achieve A_{10} of 33% as seen in Figure 4.2.

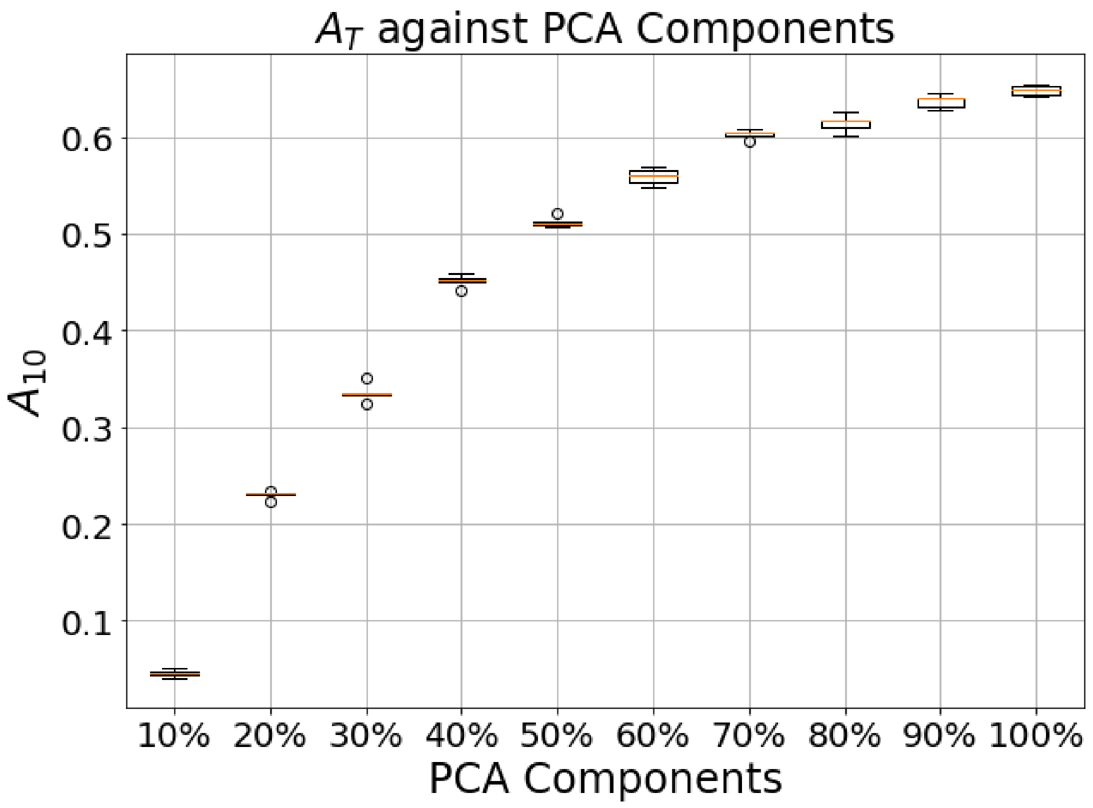


Figure 4.2: The effect of feature embedding dimensionality: The average accuracy (A_{10}) is plotted as a function of the percentage of principal components of the feature extractor. The number of features used to represent each sample is varied by applying PCA on the output of the feature extractor and is passed to the generator model.

Chapter 5

Conclusions

In this chapter, we summarize the thesis, review the contributions and list some potential areas for future work.

5.1 Thesis Summary and Contributions

In this thesis we explored the Class Incremental Learning (CIL) scenario in deep neural networks. We introduced the limitation of batched learning and motivated the need for continual learning algorithm in Chapter 1.

In Chapter 2, we reviewed two kinds of deep learning models and introduced the class incremental learning problem. First of the two deep learning models are discriminative models. These models are used to differentiate data from different classes. We also described the training methodology for discriminative models in a batched learning setting through cross-entropy loss function. The second kind of deep learning models are generative models, where they are used for learning data distributions. We introduced three different frameworks for using deep neural networks as generative models, one of which IGRE is based on. Then we described formally the class incremental learning problem for deep neural networks and reviewed three main approaches which were used to combat catastrophic forgetting.

In Chapter 3, we develop our proposed methodology by presenting our intuition on CIL through visualizing the feature embedding. We then gave a detailed description of alternating back propagation for training a generator neural network. The alternating back propagation algorithm is then adapted to the CIL problem for the IGRE framework.

This thesis presented the Incremental Generative Replay Embedding (IGRE) framework for Class Incremental Learning. The IGRE framework is a generative replay framework which does not store data, thus can be used for privacy constrained applications.

Chapter 4 of the thesis compared IGRE against wide variety of CIL methods. In particular, we focused on CAN and FearNet, which used GAN and VAE, respectively, to model the feature embedding space. We presented the benefits of using IGRE in CIL as the framework has fewer parameters so it is less susceptible to catastrophic forgetting and distribution shift. In addition, IGRE is able to perform at a level comparable to raw input replay methods despite using less memory. Although IGRE shows superior performance, we discussed the limitation of IGRE that is its dependency on the pretrained feature encoder.

Through generative feature embedding replay the framework enabled a simple and efficient way for CIL scenario. Experimental results show that this framework outperforms similar generative replay methods in longer learning sequences and IGRE has fewer parameters in the system.

5.2 Future Work

This thesis opens up a variety of future research directions, of which we will discuss three.

5.2.1 Training Feature Encoder

The feature encoder in the IGRE framework is pre-trained and assumed to be able to extract useful feature embeddings for a task. The generator is trained to do generative replay on the extracted feature embedding. Thus, if the extracted feature descriptor is not suitable for a particular task, then incremental learning on that task with IGRE may not be optimal.

Although pre-trained feature encoders are publicly available, pre-trained encoders are not guaranteed to be optimal for solving every task. For example, feature encoders pre-trained on ImageNet may not be suitable to be used with biomedical images as the colour space of natural images and biomedical images is very different. The experimental results in Section 4.7 also suggest that the performance of IGRE is dependent on how specialized the feature encoder is on the pre-trained task. That is, the more specialized the feature encoder, the better the transfer learning capability. This dependency on a pre-trained feature encoder is a limitation of the IGRE framework. Thus, it is the natural next step for us to consider incorporating the training of the feature encoder into the framework.

5.2.2 Unsupervised Continual Learning

The second future work direction is aimed at extending the IGRE framework to unsupervised continual learning. The IGRE framework is considered as a supervised classification model because it requires class labels for the conditional generator. The conditional generator then generates the appropriate features to train the classifier. Generative models can be adapted for unsupervised learning [75] where the models infer class labels, this includes the generative model in the IGRE framework. For example the Continual Unsupervised Representation Learning (CURL) method in [66], applied a variational autoencoder framework to infer the class labels for unsupervised continual learning. The generator in the IGRE framework can similarly be adapted for unsupervised learning. In addition to not needing labelled data, designing an algorithm for unsupervised continual learning can easily be used in streaming settings which has far broader engineering applications.

5.2.3 Computation Matter

The third future work direction is on improving the computational resources requirement of the IGRE framework. In Section 4.6 we seen that training generative models requires significantly more training time for each incremental task. Further research in ways to reduce training complexity for generative models is necessary to reduce the training time gap between generative replay methods and raw input replay methods. The significant training time may be a hindrance for deploying the model into production scenarios where near real-time prediction is necessary.

References

[1] Davide Abati, Jakub Tomczak, Tijmen Blankevoort, Simone Calderara, Rita Cucchiara, and Babak Ehteshami Bejnordi. Conditional channel gated networks for task-aware continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3931–3940, 2020.

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 214–223. PMLR, 2017.

[4] Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

[5] GA Carpenter. Self organization of stable category recognition codes for analog input patterns. *Applied Optics*, 3:4919–4930, 1987.

[6] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

[7] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[8] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[9] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and P. Abbeel. Variational lossy autoencoder. In *International Conference on Learning Representations*, 2017.

[10] Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. In *5th International Conference on Learning Representations*, 2017.

[11] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[12] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2(6), 2019.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[14] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[15] Robert M French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Proceedings of the 13th annual cognitive science society conference*, pages 173–178, 1991.

[16] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[17] Hongchang Gao and Heng Huang. Joint generative moment-matching network for learning structural latent code. In *IJCAI*, 2018.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] Andreas Griewank et al. On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107, 1989.

[21] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[22] Tian Han, Y. Lu, S. Zhu, and Y. Wu. Alternating back-propagation for generator network. In *AAAI*, 2017.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[25] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

[26] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In *NeurIPS Continual Learning Workshop 2018*, 2018.

[27] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[28] A Jordan et al. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14(2002):841, 2002.

[29] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[31] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[32] Diederik P. Kingma and M. Welling. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, 12:307–392, 2019.

[33] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[34] Jeremias Knoblauch, Hisham Husain, and Tom Diethe. Optimal continual learning has perfect memory and is np-hard. In *International Conference on Machine Learning*, pages 5327–5337. PMLR, 2020.

[35] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[36] Zhifeng Kong and Kamalika Chaudhuri. The expressive power of a class of normalizing flow models. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3599–3609. PMLR, 26–28 Aug 2020.

[37] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[38] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[39] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[40] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[41] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[43] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, page 9–50, Berlin, Heidelberg, 1998. Springer-Verlag.

[44] C. Ledig, L. Theis, Ferenc Huszár, J. Caballero, Andrew Aitken, Alykhan Tejani, J. Totz, Zehan Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017.

[45] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. In *International Conference on Learning Representations*, 2020.

[46] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015.

[47] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[48] Shiyu Liang and R. Srikant. Why deep neural networks for function approximation? In *ICLR*, 2017.

[49] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D Bagdanov, Shangling Jui, and Joost van de Weijer. Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 226–227, 2020.

[50] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12245–12254, 2020.

[51] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017.

[52] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[53] Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, 2016.

[54] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[55] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[56] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[57] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[58] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[59] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[60] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

[61] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11321–11329, 2019.

[62] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[63] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[64] Jary Pomponi, Simone Scardapane, Vincenzo Lomonaco, and Aurelio Uncini. Efficient continual learning in neural networks with embedding regularization. *Neurocomputing*, 2020.

[65] Jathushan Rajasegaran, Munawar Hayat, Salman H Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for continual learning. In *Advances in Neural Information Processing Systems*, pages 12669–12679, 2019.

[66] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, pages 7647–7657, 2019.

[67] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[68] Yong Ren, J. Zhu, J. Li, and Y. Luo. Conditional generative moment-matching networks. In *NIPS*, 2016.

[69] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538. PMLR, 07–09 Jul 2015.

[70] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin Austin, Texas 78712, 1994.

[71] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[72] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[73] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[74] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[75] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[76] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[77] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[78] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*, 2018.

[79] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[80] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[81] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3400–3409, 2017.

[82] K. Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[83] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

[84] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[85] Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. Topology-preserving class-incremental learning. In *ECCV*, 2020.

[86] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[87] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[88] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[89] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[90] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

[91] Y. Wu, Yan-Jia Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 374–382, 2019.

[92] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning - the good, the bad and the ugly. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[93] Ye Xiang, Ying Fu, Pan Ji, and Hua Huang. Incremental learning using conditional adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6619–6628, 2019.

[94] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[95] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[96] J. Yosinski, J. Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

[97] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[98] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[99] Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253, 2018.

[100] Yizhe Zhu, Jianwen Xie, Bingchen Liu, and Ahmed Elgammal. Learning feature-to-feature translator by alternating back-propagation for generative zero-shot learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9844–9854, 2019.