Politecnico di Torino

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING
Master of Science – Computer Engineering

GANS FOR INCREMENTAL LEARNING

Supervisors:
Prof. Elisa Ficarra
Francesco Ponzio

Candidate:
Tabriz Nuruyev

October 2021

Summary

Classical machine learning models require all the data to be available prior to the training. Once a model is trained on certain tasks, it is limited to perform on those tasks and cannot be adopted to solve a different problem. But in the dynamic world that we are living, there is no end to a stream of data coming from different sources. Incremental learning or lifelong learning is a hot research area that studies the ability of neural networks to continuously learn new knowledge while retaining old experiences, imitating the human way of learning. One of the proposed approaches is to rehearse previously learned information when learning a new class of samples. In this work, we extract features embeddings from trained samples to use them for rehearsal instead of images themselves, or generated images from those classes. The benefit of this strategy is that feature embeddings have less memory footprint, which is an essential attribute for memory critical applications. Furthermore, the feature vector of an image preserves considerably less sensitive data in itself than a raw image, subsiding privacy concerns. For this representation learning task, we propose a combination of three neural network paradigms, namely BiGAN, StyleGAN, and CGAN. BiGAN is a bidirectional GAN model in which the discriminator takes input from both the generator and encoder network. While the generator model serves its original task of generating realistic images, the encoder learns the inverse mapping from real data distribution to the feature representation. In the original implementation of BiGAN authors used DCGAN architecture which fails to generate higher-resolution images. In this project, more advanced network architectures, StyleGAN and ResNet have been adopted for generator and encoder sub-networks, respectively. Conditional GAN addresses the stochastic nature of generative adversarial networks and gives the possibility of generating samples from the desired class. At the end of the training process, the encoder sub-network is detached from the model and used for representation learning. Learned low-dimensional representations are later used for a classification task in a class incremental scenario. By its nature GANs are generative models, therefore they do not learn discriminant function. Naturally, the feature extractor of our model was not as effective as classical feature extractor networks in a classification task. However, the model showed a strong tendency to extract high-level information from the input images. Furthermore, our model is able to generate any number of feature vectors for a given class, effectively handling class imbalance problem that is very common in real-world scenarios. We believe that with a more extensive model optimization and training, our approach can serve to produce more distinctive features that can eventually keep up with classical discriminative models.

Table of Contents

1. Introduction 1
1.1 Machine learning overview 1
1.2 Generative adversarial networks 1
1.3 Representation learning 3
1.4 Synopsis 4
2. Problem Statement 5
2.1 Classical ML approach 5
2.2 Incremental Learning 7
2.2.1 Definition 7
2.2.2 Desiderata 8
2.2.3 Stability-Plasticity Dilemma 10
2.2.4 Challenge – Catastrophic forgetting 10
3. Related Work 12
3.1 Worth noting GAN models 12
3.2 Comparison of related ML paradigms 16
3.3 Main methods against CF 19
3.4 Architecture based 19
3.5 Regularization based 21
3.5.1 Weight regularization 21
3.5.2 Data regularization 23
3.6 Rehearsal Based approaches 24
3.6.1 Exemplar memory size 24
3.6.2 Sampling strategy 25
3.6.3 What to rehearse? 25
3.6.4 Incremental Classifier and Representation Learning 26
3.7 Feature rehearsal 27
3.7.1 What is it? 27
3.7.2 How to generate features? 28
3.7.3 How to evaluate generated features? 29

4. Datasets 30
4.1 CIFAR-10 30
4.2 CIFAR-100 30
5. Methodologies 32
5.1 BiGAN 32
5.1.1 Overview 32
5.1.2 Architecture 32
5.1.3 Limitations (DCGAN based) 34
5.2 StyleGAN 34
5.2.1 Overview 34
5.2.2 StyleGAN generator design 35
5.2.3 Adaptive Instance Normalization 37
5.2.4 Stochastic variations 38
5.2.5 Truncation trick 40
5.2.6 StyleGAN issues 40
5.3 StyleGAN2 42
5.3.1 Weight Demodulation 42
5.3.2 Removing progressive growing 44
5.3.3 Lazy Regularization 45
5.4 StyleGAN2-ADA 45
5.5 Conditional GAN 47
5.6 BiStyleGAN 48
5.6.1 Architecture 49
5.6.2 Loss function 51
5.6.3 Hyperparameters 52
5.6.4 Evaluation metrics 53
6. Experiments & Results 56
6.1 Overview 56
6.2 Finetuning 57
6.3 BiStyleGAN outputs 59
6.4 Classification with generated features 61
6.5 Transfer learning 65

7. Conclusion 66
8. References 68

Table of Figures

Figure 1.1 Generator & Discriminator. 3
Figure 2.1 Process of incremental learning. 8
Figure 3.1 CycleGAN image-to-image translation between domains 14
Figure 3.2 Architecture of ProGAN 15
Figure 3.3 Comparison of related ML approaches by Biesialska et al. [5]. 18
Figure 3.4 Architectural and Regularization Strategies 20
Figure 3.5 Overview of a rehearsal strategy 26
Figure 4.1 CIFAR-10 Dataset 31
Figure 5.1 Architecture of BiGAN. 33
Figure 5.2 Architecture of StyleGAN generator. 36
Figure 5.3 Synthesis network of StyleGAN generator 38
Figure 5.4 Effects of adding noise to the different layers of StyleGAN generator network. 39
Figure 5.5 Bloblike artifacts in StyleGAN outputs 41
Figure 5.6 Flaws of progressive growing 42
Figure 5.7 Improvements in StyleGAN2 44
Figure 5.8 Conditional generative adversarial networks 48
Figure 5.9 The design of the BigBiGAN network 51
Figure 5.10 Loss variations during GAN training 54
Figure 5.11 Generating images in VAE-like architecture 55
Figure 6.1 Our training procedure 59
Figure 6.2 Outputs images of BiStyleGAN after 10, 25, 50 epochs 61
Figure 6.3 Training feature embeddings in various settings 63
Figure 6.4 ResNet-50 and ResNet-101 feature embeddings 64
Figure 6.5 Feeding CIFAR-10 images to the pre-trained BigBiGAN network. 65

1. Introduction

1.1 Machine learning overview

The whole concept of machine learning is exploring various ways to teach the computer to perform a certain task without direct supervision. The process can be seen as an attempt to develop an “intuition” in a computer. To instill the ability to perform a certain task effectively, we need to provide relevant knowledge to the computer which is the data that comes in many forms. Once we select the model, a learning framework, we preprocess our data according to the “needs” of our model and feed the processed data to our model. We tune our model parameters with the preprocessed data to optimize its performance. The final step is to evaluate the performance of the model with data the model has not seen before during training known as test dataset. The objective and settings behind this process define the family of approaches our model belongs to. A typical machine learning problem involves a task of prediction where a model is trained on some samples and associated labels and is expected to make predictions on similar, unseen samples. It is an iterative process where samples are shown to the model along with their labels, and if necessary, the model is corrected to make a prediction close to the expected output. The family of these approaches is called discriminate models, in which the aim is to find a discriminant function that can map the given input to the correct label.

1.2 Generative adversarial networks

There is one another family of models called generative models, where the goal is to generate new samples from given input distribution. Those models achieve so by successfully summarizing input distribution and create new samples within that distribution. Different models, such as Naïve Bayes, Latent Dirichlet Allocation, and Variational Autoencoder, have been adopted to tackle this problem, but one that shines most among them is Generative Adversarial Networks.

Proposed by Ian Goodfellow back in 2014, Generative Adversarial Network [1] was a breakthrough, described as "the most interesting idea in the last 10 years in Machine Learning" by one of the prominent researchers in the Machine Learning field Yann LeCun. The idea is to put two network sub-models, namely generator, and discriminator, in an adversarial training process. The goal of the generator model is to generate realistic images. In contrast, the discriminator model tries to classify whether input data is authentic, from the domain, or fake, composed by the generator network. It is a clever approach where authors integrate a generative task with a discriminative task. The generator model takes a random noise vector drawn from a Gaussian distribution, generates an image from that noise vector. In the beginning, the generator does not produce effective results, but as it takes constant feedback from the discriminator network, the generator can eventually fool the discriminator.

The training process is a zero-sum game in the sense of game theory, described in Figure 1.1. Once the generator produces a batch of samples and presents it to the discriminator, the discriminator classifies them as real or fake. If the generator successfully fools the discriminator, the discriminator labels a generated image as real, the latter is penalized by a change in network parameters. Instead, if the generator fails to fool the discriminator, the network parameters of the generator are subject to modification.

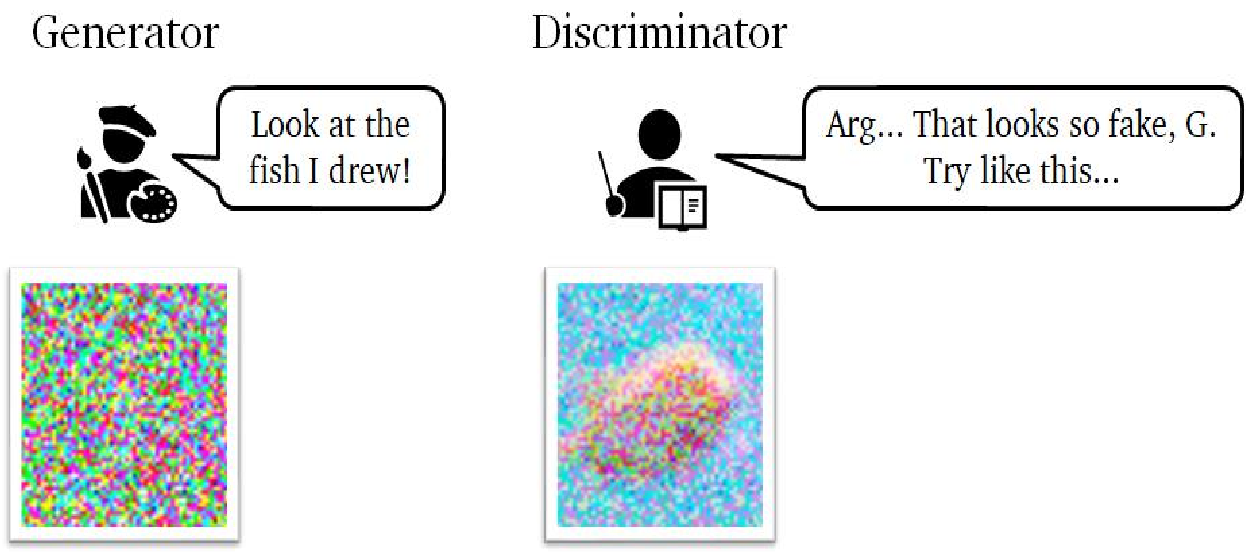


FIGURE 1.1 GENERATOR & DISCRIMINATOR.

By getting constant feedback from the discriminator network, the generator network manages to generate authentic images that the discriminator fails to tell apart from actual samples.

1.3 Representation learning

Representation learning is a process of generating low-dimensional representation from high-dimensional data to be exploited in machine learning tasks. Transforming raw data to a lower dimension reduces the noise, facilitates the decision-making process. In the context of deep learning, as we move forward input data through the layers of CNN networks, a more compact and complex representation is learned. The last hidden layer presents the final feature vector to the classifier, and the classifier outputs a prediction. Classic network architectures like AlexNet, ResNet, Inception, VGG are generally used for the representation learning task. In this work, a different approach, GAN architecture, has been exploited for feature extraction. The goal of the representation learning in this work is to generate low-dimensional feature descriptors to replace real samples in an incremental learning task.

1.4 Synopsis

In this work, we aim to develop a robust model that can gradually learn to excel on new tasks, in a way, imitate the human way of learning. In our hybrid machine learning approach, our first task is a generative one where we try to produce representative feature embeddings which can effectively summarize real data distribution. Low-dimensional feature embeddings have the advantages of being memory-efficient and not being subject to privacy concerns as opposed to raw images. Learned feature embeddings are then used for a classification task in an incremental learning scenario, where a new class of data becomes available over time. So for each new batch of classes, we aim to learn an effective feature extractor to summarize the data in an efficient way and use the generated features to classify incoming image samples. In a nutshell, our approach is a combination of generative and discriminative methods.

2. Problem Statement

2.1 Classical ML approach

In 1997 remarkable event for the status of artificial intelligence happened. For the first time in history, artificial intelligence beat a human in a game of chess. Deep Blue, a supercomputer developed by IBM, managed to take a win against Garry Kasparov, one of the most prominent chess players in history. Deep Blue relied on brute force strategy to analyze all the available moves. Later, humans developed more intelligent robots with more sophisticated techniques. Go is an ancient Chinese game that has more straightforward rules but far more possible moves. Before 2015, computers could only beat amateur players in a game of Go, since the game requires high intuition, and the brute force approach is not a winning strategy for the game. However, yet another spectacular event happened in 2015, and Google DeepMind's AlphaGo managed to take a win against 18-time world champion Lee Sedol in a five-round game of Go. The robot was far superior to IBM's Deep Blue and was developed using deep learning and reinforcement learning techniques [2].

Many similar examples have given hints of the fact that machine learning models exhibit comparable and even superior performance against human beings on specific tasks. Nevertheless, there is one matter that should be stressed upon. Machine learning models are great at what they do such that they even surpass humans, but they are incapable of adapting their behavior to a new task or environment. They do not possess human cognition that can continually obtain, distill, transfer knowledge and skills through their lifespan. With a massive amount of data generated every day, computational devices are expected to learn multiple tasks from the diverse data distribution. In a dynamically changing external world, classical machine learning models become inadequate in handling diverse tasks simultaneously for several reasons.

- Data should be available beforehand

In classic ML tasks, all the data should be available to the model prior to the training process. Once a model is trained on available data, the generated model is fixed to perform the defined task on that data. If new, additional data with different class labels is available after the training, either a new model should be developed trained on the combination of old and new data, or a performance drop of the model on the new data should be expected. Considering the fact that there is no end in the availability of the new data in real-world scenarios, the second option is not an optimal approach for the general use case.

- Batch training is a lengthy process

Batch training is the method used for training neural networks with big datasets. The data available at a certain point is split into chunks and is feedforwarded to the neural network sequentially, not once but several times, defined by a hyperparameter called an epoch. The number of epochs needed to train the model effectively can be as high as hundreds. Given these circumstances, training a new model from scratch whenever a new class becomes available and repeating this process in the model's lifespan is not a desirable strategy at all.

- Memory limitations

We do not use flagship computers to train a neural network and make predictions in every sector. Instead, we have devices with limited computational resources and memory restrictions. In a real-world with full of memory-restricted devices, it is not feasible to retain the whole dataset within them. However, if a new class of input data becomes available, those devices will need to train a model with new data along with old data. Additionally, the classic batch training property of machine learning causes severe performance issues on those devices. Therefore, the classical machine learning approach is not suitable for the majority of the end devices.

- Privacy concerns

The property of retaining the whole dataset in memory is a cause for privacy concern in itself. The idea of a model processing sensitive data of the users dozens of times in its lifespan does not quite sound safe. A neural network that does not need to preserve a dataset for later use is a more preferable property from a user's point of view. But again, the trained model can only perform on the learned dataset in the standard approach.

2.2 Incremental Learning

2.2.1 Definition

In a nutshell, the classical ML approach is not suitable for scaling up a single model when the number of tasks increases over time or deploying a model in real-world use cases where there is a limitation of computational power and resources. To tackle the problem as mentioned above, a new research area in the Machine Learning field has emerged. Incremental Learning (or Lifelong Learning in some other literature) is a subfield of Machine Learning in which the goal is to develop methods to perform the classification task when the model continuously receives input data. The form of input can vary from a single sample to thousands of images. However, the main attribute of incremental learning that separates it from its counterpart is incremental learning models requires seeing the training data only once. In literature, new data with a different class label is generally referred to as a new task, so from now on it will be referred to in the same manner. The ultimate goal is to retain the performance of the model at an acceptable level (or even increase it, if possible) as new data becomes available.

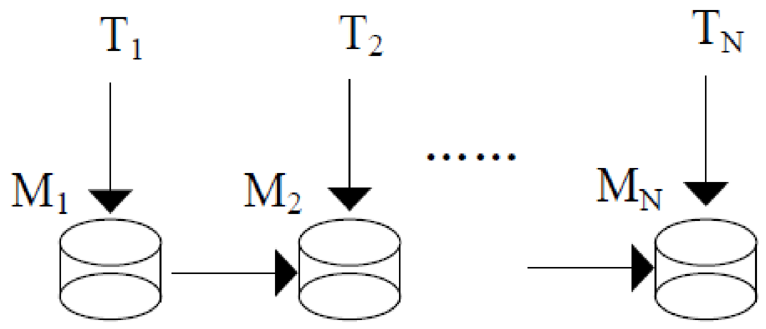


FIGURE 2.1 PROCESS OF INCREMENTAL LEARNING.

2.2.2 Desiderata

- Knowledge retention

Knowledge retention is defined as the ability of the model to preserve previously learned knowledge when it is trained on the new data however, it is not an easy task to do. Basically, when a neural network is trained on certain data, the model weights are modified in a way that when an unseen sample arrives it can classify the input correctly. But if the model is tried to be optimized for a different task from which it is trained for, its model weights can be changed in a way that the model cannot perform the original task anymore. Therefore, knowledge retention is the core objective in Incremental Learning, or rather knowledge retention essentially defines the Incremental Learning task.

- Online Learning

As stated previously, in lifelong learning rather than having all the data available priorly, we would like to have the ability to train the model on a continuous stream of input data. Which in terms means that a model can observe each training sample only once (or maybe only a few times) instead of receiving thousand times like in batch training.

- Forward Transfer

A forward transfer is the positive influence of the knowledge learned from current tasks on the learning of future tasks. If the forward transfer is high enough, it can enable zero-shot learning which is classifying input data based on a few or no labeled samples. Forward transfer can also be negative, and it is when the ability of a model to learn new concepts deteriorates as the number of tasks increases.

- Backward Transfer

As the name suggests, a backward transfer is the reverse process of the forward transfer. Positive backward transfer improves the model performance on previous tasks, while a negative backward transfer worsens the model performance on previous tasks, and if high, enables catastrophic forgetting.

- Fixed Model Capacity

As incremental learning models are generally designed to operate in real-world use-cases, from self-driving cars to humanoid robots, increasing model size is not a desired property, especially considering the fact some devices have very limited memory capabilities. That's why an increasing number of tasks should not lead to increasing model size rather more advanced techniques should be adopted to dedicate resources for incoming tasks.

- Graceful Forgetting

Graceful forgetting is one of those techniques to let the model make space for new tasks without the need to increase the model size. The idea is to selectively forget the information which is deemed either less important or trivial for the classification of a task.

- No Task Boundaries

Having no task boundaries implies having no requirement neither on the form of the data (for example, the model does not require the data to be i.i.d) nor on the task division. There are two main approaches in the task division in incremental learning. In task continual learning (TCL), each task is a separate classification problem (e.g., one classifying different breeds of dogs and one classifying different types of birds). TCL builds a set of classification models (one per task) in one neural network. In testing, the system knows which task each test instance belongs to and uses only the model for the task to classify the test instance [3]. This setting was generally deployed in the previous experiments since the overall task is relatively easy. While in the testing stage under the class continual learning setting, the model has no access to the information about the task. Class continual learning is a more attractive setting for the researchers and this setting aligns with the desiderata of no task boundaries.

2.2.3 Stability-Plasticity Dilemma

Incremental learning methods try to find the optimal trade-off between the ability to learn new concepts while retaining the knowledge learned previously. Given the fact that the model should perform the classification task for any previously learned task at any given time, learning a new task should not deteriorate the performance of the model on previous tasks. However, as the number of learned tasks grows, learning new information for the model gets more difficult. And the problem arises when the model is at its limit to accommodate new knowledge in its network. In that case, the model either cannot learn a new task or lose its ability to classify previous tasks effectively. In the worst-case scenario, it can forget all the relevant information about previous tasks, a phenomenon known as catastrophic forgetting. This trade-off is called the stability-plasticity dilemma and incremental learning approaches try to find the optimal trade-off between these two, contradicting aspects.

2.2.4 Challenge – Catastrophic forgetting

Catastrophic is the main challenge in incremental learning, that is performance drop on the previous task as the new tasks arrive. Since incremental learning aims to keep the size of the classifier constant, catastrophic forgetting is a very natural byproduct of this process. A model with a limited size can only retain limited information. There are several causes of catastrophic forgetting:

- Weight drift

When learning a new task, the weights of the model are modified in a way that it can correctly classify input data from the new task. However, this change in model weights can affect the model performance negatively on previous tasks, since the changed weights were previously optimized for those tasks.

- Activation drift

Activation drift has a cause-effect relationship with the weight drift. Changed model weights cause changes in activation function, and consequently misclassification of the input data. Focusing on activations rather than on weights can be less restrictive since this allows weights to change as long as they result in minimal changes in layer activations (Masana et al., 2020).

- Inter-task confusion

Since in incremental learning setting tasks are never jointly trained, the model cannot discriminate all the learned tasks in the same manner.

- Task-recency bias

It is only obvious that a model will perform better on a recent task since a recent task is less subject to weight drift and activation drift than an older task.

3. Related Work

3.1 Worth noting GAN models

As mentioned earlier, we adapt a GAN network for representation learning to be used in a lifelong learning setting. Picking the right model for a specific use case can be tricky, so several popular GAN models have been revised. Here we will point out some of them that we think are worth mentioning.

- Generative Adversarial Nets

Ian Goodfellow can be considered the founder of generative adversarial networks. In his paper "Generative Adversarial Nets" [1], presented in 2014, he was the first one to propose the training of two neural networks in an adversarial fashion. Discriminator and generator networks were comprised of solely fully connected layers and did not include convolutional layers, which are crucial for image tasks. Nevertheless, the model performed successfully in MNIST and CIFAR-10 datasets.

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)\right] \tag{3.1}$$

- Deep Convolutional Generative Adversarial Networks (DCGAN)

DCGAN is a successor of the original GAN where Radford et al. use deep convolutional networks instead of the multi-layer perceptron to make the network more suitable for image generation tasks. DCGAN paper highlighted the constraints on the training of the generator and proposed the appropriate solutions, such as removing fully connected layers, using batch normalization in both sub-networks which is crucial for stable training.

- Wasserstein Generative Adversarial Network (WGAN)

Wasserstein GAN tackles the mode collapse problem by introducing a new sub-network, "critic", that replaces the discriminator model. Instead of producing binary results (SoftMax activation), the critic outputs an unbounded real number. During training, the critic tries to give scores to real images which are far apart from scores it gives to fake ones. And the generator on its side aims to close the gap between the scores.

Furthermore, previous GAN networks suffer from vanishing gradients problems because of the deployed loss function. As the discriminator gets better in the classification task, the feedback it provides to the generator gets worse and the generator struggles to improve. This phenomenon is called the vanishing gradient problem, and the authors pinpoints the root of the problem, Jensen–Shannon Divergence.

WGAN introduces Wasserstein Loss described in Eq. (3.2) to avoid the vanishing gradient problems. There is no constraint such as loss should be between 0 and 1 as in Jensen-Shannon Divergence, therefore cost function can grow based on the scores from two distributions. However, Wasserstein loss uses the 1-Lipschitz Continuous function which enforces the norm of the gradient to be at most one anywhere. To achieve so, the authors use a very simple approach, weight clipping, which is clipping network weights that surpass the predefined threshold. Authors choose this method for its simplicity and good performance, but they also highlight the fact that it is not an optimal solution that can lead to the vanishing gradient problem if the threshold range is not chosen correctly. From one perspective, the method can serve for weight regularization purposes however, in some cases, it prevents the network from learning high-level knowledge.

$$\min_{G} \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}}))] \tag{3.2}$$

WGAN-GP is an extension of WGAN where gradient penalty is used to satisfy the 1-Lipschitz Continuous function instead of weight clipping. Although the introduced method adds computational complexity, it also results in higher-quality images.

- CycleGAN

CycleGAN performs image-to-image translation, converting an image from one domain to another in a reversible way. It deploys two generator and two discriminator networks, one generator translating an image from domain A to B and another doing the opposite. Discriminators distinguish translated images from the real ones from their domain. The novelty of CycleGAN is it does not require the images from two domains to be presented in a pair-wise fashion, which is the constraint in another notable GAN network, Pix2Pix.

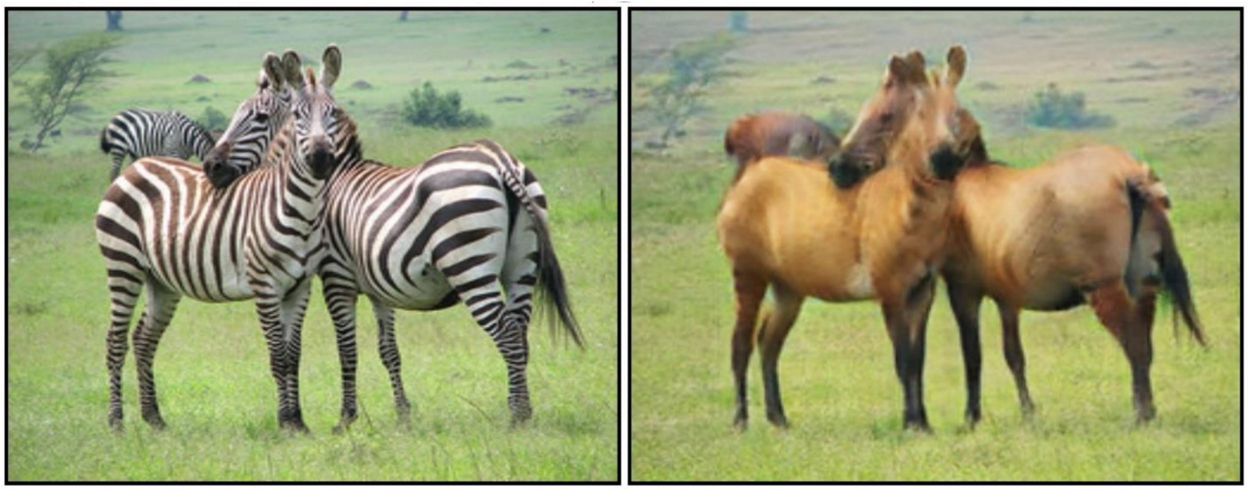


FIGURE 3.1 CYCLEGAN IMAGE-TO-IMAGE TRANSLATION BETWEEN DOMAINS

- Progressive GAN

Progressive GAN presents the idea of progressively increasing model depth, generating the image details step by step, from general features to finer details. It starts with generating low-resolution images such as low as 4x4. Once the model is stabilized, another layer with the double size is added and the training starts again. In each stage previously learned network weights are transferred but not frozen. This procedure proceeds until the desired resolution is achieved. Of course, in low resolutions, we do not expect to come up with nice samples. The output images will be a just blur, therefore in those stages general shape, distinct attributes of desired samples are considered. For example, if we generate face samples, we would like to end up with an oval shape at least.

Generators usually struggle to generate samples with high variety. To address this issue, the authors insert the "minibatch standard deviation" layer into the discriminator network in which pixel-level standard deviation for a given batch is calculated. The generator sub-model tries to produce images of which batch statistics are similar to the one calculated from real data samples. Finally, for the loss function, they deploy WGAN-GP loss explained in Eq. (3.2).

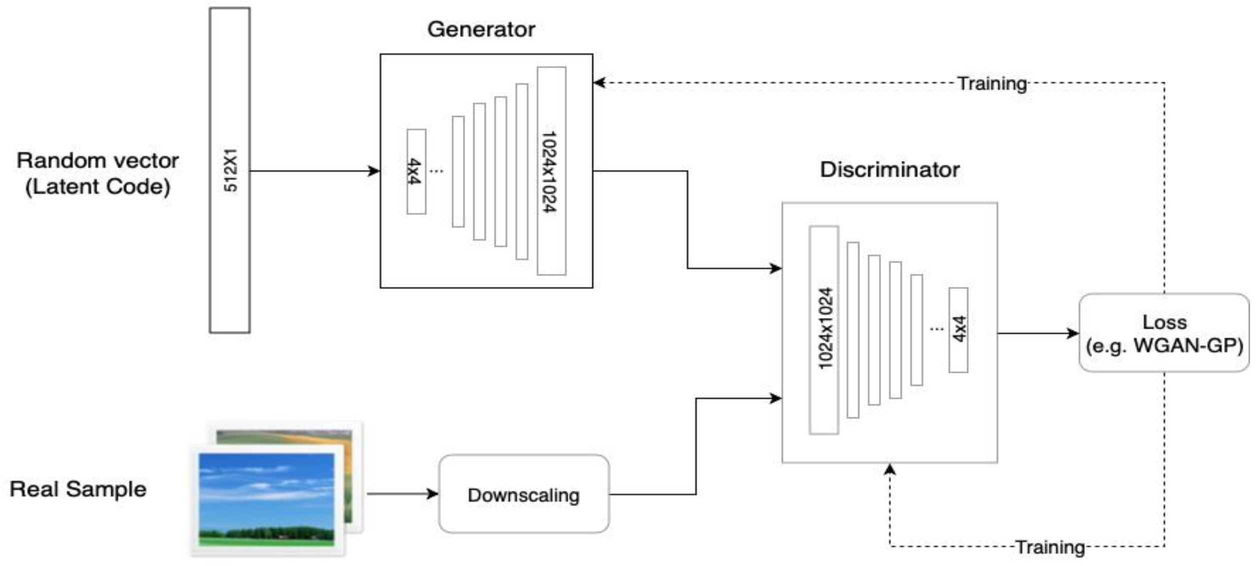


FIGURE 3.2 ARCHITECTURE OF PROGAN

- BigGAN

Big Generative Adversarial Networks (BigGAN) [4] is an approach build upon existing best practices in the field of GAN. The method follows a simple idea of significantly increasing network capacity and batch size to achieve high-fidelity output images (Figure 3.2). Backbone architecture is self-attention generative adversarial networks (SAGAN) with hinge loss function that is the general choice for the training of support vector machines. On the contrary to the common practice, BigGAN updates the discriminator network twice before updating generator weights. Moving average of the network weights, the idea proposed in ProGAN paper, is maintained to produce generator outputs at any given time during the training process. Skip connection inspired by the ResNet paper is also introduced, directly connecting the noise vector to any specific layer in the generator sub-network.

3.2 Comparison of related ML paradigms

Different machine learning approaches are closely connected in one way or another. In this section, similarities and differences among lifelong learning and other dominant ML approaches will be analyzed.

- Transfer Learning

Transfer learning can be the most fundamental method of knowledge transfer in the ML field. When we have a pre-trained model for one specific task, and we would like to use the same model for a very similar task, the transfer learning method comes in handy. Using the pre-trained model, and finetuning network parameters of that model for the new task we can obtain faster convergence and good performance on the task. Generally, in transfer learning, the initial layers of the transferred model are not modified, which is called freezing the layers. The reason is initial layers are good to capture basic shapes, while later layers try to obtain more advanced information. So, when applying transfer learning, we finetune the later layers of the transferred model. Transfer learning provides the property of the forward transfer, same with one of the objectives incremental learning tries to achieve. However, transfer learning does not provide any other advantages among the desiderata of incremental learning.

- Multi-task Learning

Multi-task learning aims to learn multiple, related tasks at once. Instead of training an independent model for each task, multi-task learning develops a single model and optimizes loss function for all similar tasks. By doing so, multi-task learning aims to achieve generalization, achieving a single model which is not bound to solve one, well-defined task rather a model that can perform well enough in a set of related tasks. The keyword here is "related": unlike lifelong learning, multi-task learning does not provide "no task boundaries" property. Furthermore, multi-task learning sometimes suffers from the negative transfer. Co-training the set of tasks affects the performance of the model on those tasks, and an equivalent model trained specifically for a single task performs better.

- Meta-learning

Meta-learning is defined in many works of literature as "learning to learn", that is adapting to the new concepts and environments with limited exposure to the task configuration. Meta-learning models try to imitate the human way of learning, learning new concepts in a faster and efficient manner. It uses the metadata of classical learning algorithms as an input, provides better generalization, improved performance, and faster learning when trained on the new task.

- Curriculum Learning

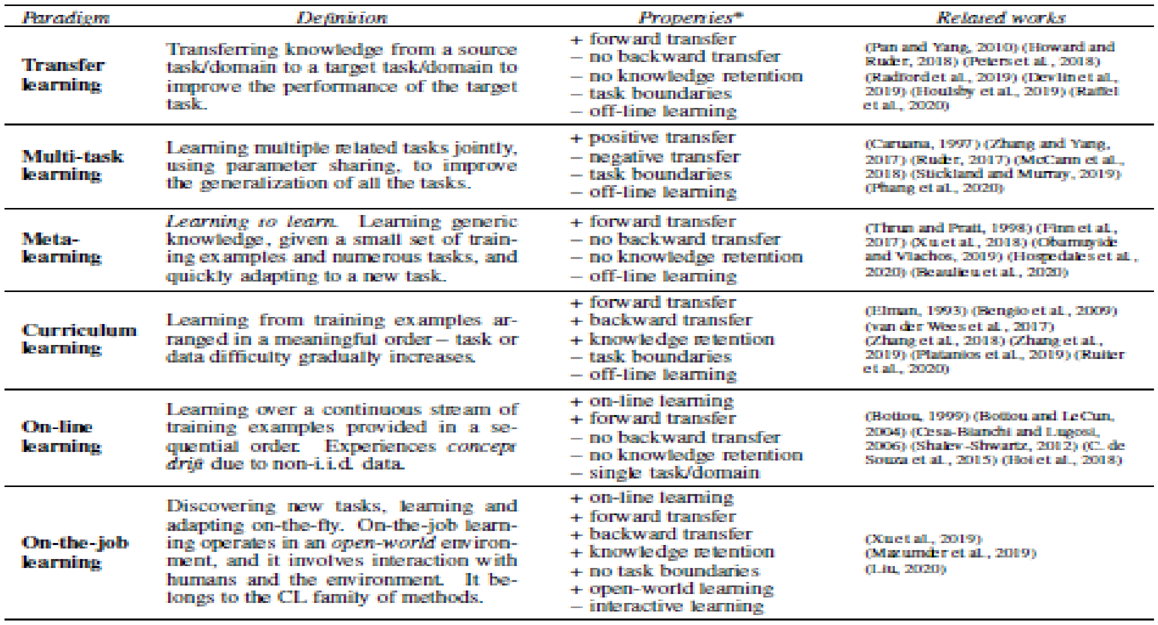
The idea behind curriculum learning is starting to train the model with a relatively easy task and increase the difficulty of tasks as time goes on. Curriculum learning provides both forward and backward transfer. Again, curriculum learning is against the "no task boundaries" property of lifelong learning since tasks or data are fed to the model in a meaningful order.

- Online Learning

Online learning can be considered a subset of incremental learning since it aims to achieve one of the objectives of incremental learning, as obvious by its name. But online learning does not provide other properties like knowledge retention, backward transfer and it is limited to work on a single task.

- On-the-job Learning

On-the-job learning is a new, very interesting field that deviated from lifelong learning itself. As the name suggests, it involves adapting to a new environment or a new challenge on the fly. On-the-job learning operates on real-world use cases and involves interaction with the user itself and the environment. The model can take feedback from the user or environment when faced with an unfamiliar task and use that feedback as a classification label. In this way, when faced again with the same task, the model can perform its job without the need for intervention by the user or interaction with the environment. Self-driving car is one of the use cases where on-the-job learning fits well for the goal.



Paradigm	*Definition*	*Properties**	*Related works*
Transfer learning	Transferring knowledge from a source task/domain to a target task/domain to improve the performance of the target task.	+ forward transfer – no backward transfer – no knowledge retention – task boundaries – off-line learning	(Pan and Yang, 2010) (Howard and Ruder, 2018) (Peters et al., 2018) (Radford et al., 2019) (Devlin et al., 2019) (Houlsby et al., 2019) (Raffel et al., 2020)
Multi-task learning	Learning multiple related tasks jointly, using parameter sharing, to improve the generalization of all the tasks.	+ positive transfer – negative transfer – task boundaries – off-line learning	(Caruana, 1997) (Zhang and Yang, 2017) (Ruder, 2017) (McCann et al., 2018) (Stickland and Murray, 2019) (Phang et al., 2020)
Meta-learning	*Learning to learn.* Learning generic knowledge, given a small set of training examples and numerous tasks, and quickly adapting to a new task.	+ forward transfer – no backward transfer – no knowledge retention – off-line learning	(Thrun and Pratt, 1998) (Finn et al., 2017) (Xu et al., 2018) (Obamuyide and Vlachos, 2019) (Hospedales et al., 2020) (Beaulieu et al., 2020)
Curriculum learning	Learning from training examples arranged in a meaningful order – task or data difficulty gradually increases.	+ forward transfer + backward transfer + knowledge retention – task boundaries – off-line learning	(Elman, 1993) (Bengio et al., 2009) (van der Wees et al., 2017) (Zhang et al., 2018) (Zhang et al., 2019) (Platanios et al., 2019) (Ruiter et al., 2020)
On-line learning	Learning over a continuous stream of training examples provided in a sequential order. Experiences *concept drift* due to non-i.i.d. data.	+ on-line learning + forward transfer – no backward transfer – no knowledge retention – single task/domain	(Bottou, 1999) (Bottou and LeCun, 2004) (Cesa-Bianchi and Lugosi, 2006) (Shalev-Shwartz, 2012) (C. de Souza et al., 2015) (Hoi et al., 2018)
On-the-job learning	Discovering new tasks, learning and adapting on-the-fly. On-the-job learning operates in an *open-world* environment, and it involves interaction with humans and the environment. It belongs to the CL family of methods.	+ on-line learning + forward transfer + backward transfer + knowledge retention + no task boundaries + open-world learning – interactive learning	(Xu et al., 2019) (Mazumder et al., 2019) (Liu, 2020)

FIGURE 3.3 COMPARISON OF RELATED ML APPROACHES BY BIESIALSKA ET AL. [5].

3.3 Main methods against CF

In the literature, incremental learning approaches are mainly divided into three categories namely, architecture-based, regularization-based, and rehearsal-based approaches. The difference among these families of approaches comes from the way task-specific information is stored and utilized during the whole training lifecycle. Here we will discuss the strength and weaknesses of those approaches and have a glimpse of the training procedure of one method from each family.

3.4 Architecture based

Architecture-based approaches try to mitigate the catastrophic forgetting by making changes to the network architecture of the model. The change can be freezing network units related to previous tasks, increasing the model size in order to make space for a new task, keeping a separate classifier unit for each unique task, and so on. Model architecture size can be fixed or dynamic, which is the size of the model increases as new tasks arrive. Having a fixed model size is memory efficient, computationally cheaper, and aligns with the desiderata of lifelong learning, however, scalability issues arise when a model is at its limit to accommodate a new task. For example, some approaches dedicate a certain number of network units for each task and train the model using solely those units dedicated for the task, freezing the rest of the trainable parameters. It is obvious that with this approach, a model can only accommodate only a limited number of tasks. For example, if a model has 100K of trainable units and for each task, we assign 5K trainable units, the maximum number of tasks a model can classify properly is twenty. 21st task cannot be added to the model since the model has no resources to dedicate for that task.

PackNet network model is an architecture-based incremental learning approach that uses a similar technique mentioned above. The training process consists of two stages: in the first stage, a model is trained with all the network units that are not yet assigned to any other task. After the first stage, used network units are ranked by importance using the magnitude of change, the higher the magnitude more important a network unit is for the trained task. Subsequently, some fraction of units deemed less important are pruned from the training process and the model is trained again with the important units. The number of units to be pruned can be decided based on information about the total number of tasks the model is expected to solve in its lifetime. "Unimportant" units are later used in future tasks, and again some of them are deemed important for the task, and some are reserved for later use.

Architecture-based approaches generally do not align with the desiderata of incremental learning. The reason is related approaches require more resources (let it be memory, increased model size) as the number of tasks grows. Fixed-memory architecture-based approaches, on the other hand, do not scale well in the incremental learning setting.

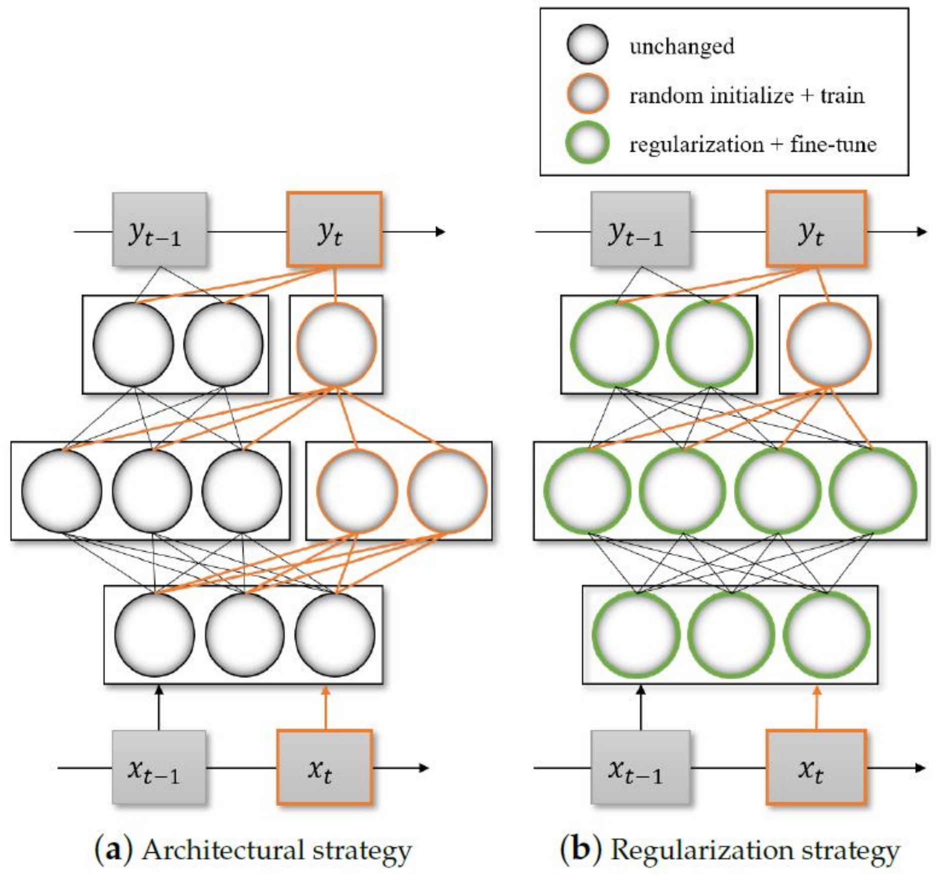

unchanged
random initialize + train
regularization + fine-tune
y_{t-1}
y_t
x_{t-1}
x_t
(a) Architectural strategy
(b) Regularization strategy

FIGURE 3.4 ARCHITECTURAL AND REGULARIZATION STRATEGIES

(a) Architectural strategy. When new data becomes available, the neural network dedicates new resources for the task, either by expanding the existing model or dedicating network units within the model. X_t *and* Y_t *represents the input and the output of the model at the time t, respectively.*
(b) Regularization strategy. Regularization-based strategies instead aim to exploit existing model weights without the need to increase the model capacity.

3.5 Regularization based

Instead of making modifications to the model architecture, regularization-based methods impose limitations on the update of model weights through various ways. This family of methods introduce additional loss term to the loss function. Consequently, the loss function consists of classification loss, we are familiar with from standard classification models and additionally, distillation loss. This loss term penalizes abrupt changes in network weights and by limiting weight changes it helps to alleviate catastrophic forgetting. Regularization-based approaches can be divided into two families of techniques.

3.5.1 Weight regularization

Weight regularization methods try to prevent weight drift, one of the four reasons behind catastrophic forgetting, by introducing additional loss. In order to prevent weight drift, the importance of all network parameters for prior tasks is estimated, a weight of importance for each network unit is generated. During the training of later tasks, modifications to each network unit are penalized by the factor of their weight. In this way, a change to a network unit important to previous tasks is penalized more harshly.

Elastic Weight Consolidation is one of the pioneer weight regularization methods introduced in 2016 [6]. EWC is exactly doing the same process described above, introducing additional loss for the network weights. When training a new task, the importance of all network parameters related to the previous task is calculated through Fisher Information Matrix. Changes to the parameters are penalized quadratically. Eq. (3.3) describes the loss function EWC uses for the training process.

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_j \frac{\lambda}{2} F_j \left(\theta_j - \theta_{A,j}^*\right)^2 \tag{3.3}$$

$\mathcal{L}_B(\theta)$ is classification loss of current task B, $\theta_{A,j}^*$ is the optimal network parameters found for previous task A, θ_j are current values of the network parameters, F_j is Fisher Information Matrix explained earlier, and finally λ is a stability-plasticity factor. If the factor is set high, the model will prioritize performance on the older task rather than on the new, in other words, stability over plasticity, and if set low, plasticity over stability. Again, the choice of the stability-plasticity factor is dependent on the use case, the goal of the implementation.

There are several drawbacks of the EWC. First of all, the method requires keeping Fisher Information Matrix for each task and makes it unfeasible to deploy a model on an end device. Furthermore, the method is required to do an extra pass over the task it is trained for which is not a desired property for lifelong learning applications.

EWC++ is an upgraded version of standard EWC where instead of calculating Fisher Information Matrix after each task, a single FIM is kept which is calculated as a moving average over all the trained tasks.

3.5.2 Data regularization

Data regularization methods, on the other hand, try to prevent activation drift. This family of methods is based on the Knowledge Distillation technique. The original implementation of Knowledge Distillation aims to learn to a more compact student network from a bigger teacher network with a minimal performance drop so that generated models can be used in systems with limited memory and computational resources. Specifically, a student network takes soft labels generated by a teacher network as input and tries to imitate the behavior of the teacher model, in technical terms tries to output the same labels from the last activation function.

Learning without forgetting (LwF) [7] is the first method that utilizes Knowledge Distillation in an incremental learning context. In this case, both the teacher and student model is the used model itself, specifically, the teacher model is the model after being trained on the last task and the student model is the model being trained on the current task.

$$\mathcal{L}(\theta) = \left(\lambda_o \mathcal{L}_{\mathrm{KD}}\left(Y_o, \hat{Y}_o\right) + \mathcal{L}_{\mathrm{CE}}\left(Y_n, \hat{Y}_n\right) + \mathcal{R}(\theta)\right) \tag{3.4}$$

In Eq. (3.4) above, $\mathcal{L}_{\mathrm{KD}}$ signifies knowledge distillation loss, which ensures stability in the training process while $\mathcal{L}_{\mathrm{CE}}$ describes cross-entropy loss for the currently trained task. λ_o is again the stability-plasticity factor, can be set high or low depending on the applications. $\mathcal{R}(\theta)$ is the regularization term, utilized generally in neural network training to constraint the drastic increase of network weights and in this way to prevent overfitting. One worth mentioning property of LwF is it does not require seeing old data when learning a new task.

LwF has some serious drawbacks. First of all, it is designed for applications in multi-head settings, which means it is not suitable for class continual learning. Furthermore, the performance of the approach is heavily dependent on the relatedness of the tasks.

While being an obsolete approach for the current time, LwF is a pioneer work in the field of incremental learning and has contributed to many advanced approaches.

3.6 Rehearsal Based approaches

Rehearsal-based approaches retain a subset of samples from each previous task in its exemplar memory and co-train them with new task samples in order to alleviate catastrophic forgetting. Feeding samples from older tasks to the model during training prevent the model from forgetting representation about previous tasks. There are several configurations in which rehearsal-based approaches can be deployed.

3.6.1 Exemplar memory size

Exemplar memory can be deployed in fixed or dynamic memory configurations. If a model has an increasing size of the memory of exemplars, the model can easily dedicate resources for each incoming task by simply adding a new subset of samples to the exemplar memory. This option leads to a linear increase in memory size, and in a long run, it might not be suitable for some applications.

If a model has a fixed maximum size for all the tasks, it may need to remove some samples from the exemplar memory in order to make space for new task samples. As the number of tasks grows, the model will hold a fewer number of samples for each task in its memory, which can lead to an overall performance decrease among all the learned representations. To prevent this issue from happening, an optimal sample removal strategy should be adopted. For example, some algorithms record the order of samples picked for the rehearsal by decreasing order of importance, and when some exemplars are needed to be removed to make space, samples with lower importance are removed first [8].

3.6.2 Sampling strategy

The sampling strategy is another important factor that affects the performance of the model. Exemplars can be selected randomly without any additional overhead, which is proven to be an effective strategy by Masana et al [9]. However, there exist some other methods that adopt more advanced sampling strategies. The herding strategy presented in iCaRL [10] picks a subset of samples whose mean of latent features are closest to the mean of all the samples in the relevant class, in this way representing original distribution in the best way possible. Meanwhile, Maximal Interfered Retrieval by Aljundi et al. [11] retrieves samples that are subject to the highest loss increase by incoming network parameters update which is estimated by virtual parameter update. There are numerous other approaches for the sample selection, but it is generally advised to use random sampling since it is computationally cheap and produces similar results against its advanced rivals.

3.6.3 What to rehearse?

What form of information to keep about classes is another question to be answered. There are three main options to consider: keeping raw images, keeping generated images, and keeping features generated from images.

- Raw images

Keeping raw images is the simplest approach, but this choice has serious drawbacks. First of all, raw images have a much bigger memory footprint, which makes the method not suitable for memory-critical applications. Secondly, this method is subject to the class imbalance problem. If one class has much fewer training samples than the other classes, the model will not perform well on that task. Finally, keeping raw images can cause privacy concerns in real-world applications.

- Generated images

Instead of keeping real images, artificially generated images can be also kept for rehearsal purposes. Generated images are not subject to privacy concerns as in the case of raw images. Class imbalance is not an issue in this approach since thousands of images can be generated for classes that lack enough samples. Nevertheless, generating real-like images is an expensive operation and also, as in the case of raw images, generated images are memory-inefficient, which left us with the final option, feature rehearsal.

- Generated features

Feature rehearsal is a promising approach in the incremental learning field. This option has much less memory footprint, minimizes privacy concerns, eliminates the class imbalance problem, is a better choice for big datasets. The only drawback of feature rehearsal over other approaches is it is not easy to evaluate how well generated features represent the real images.

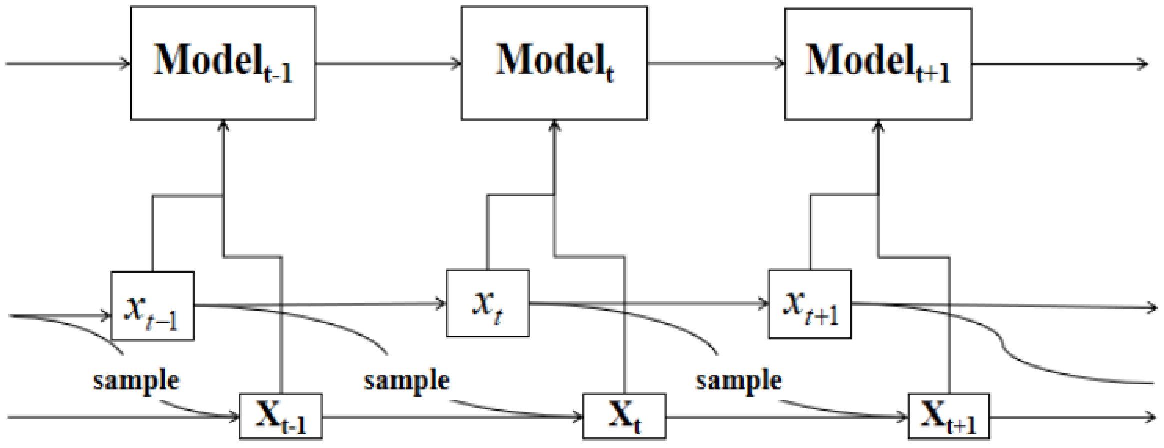


FIGURE 3.5 OVERVIEW OF A REHEARSAL STRATEGY

3.6.4 Incremental Classifier and Representation Learning

Despite being introduced in 2016, iCaRL [10] is one of the most successful methods still now. The method utilizes two loss functions for representation learning, namely classification loss, and distillation loss. Distillation loss prevents abrupt changes in feature mapping while training a new class. As mentioned previously, iCaRL uses herding as a sample selection strategy. It picks a subset of real samples whose mean of latent features is closest to the mean of all the samples in the related class. Furthermore, it records the order of the selection so that it can evaluate which samples are more important for the representation of the related class. If fixed exemplar memory is applied, images with less importance will be removed first to make space for the new task samples.

The drawback of the iCaRL method is actually the proposed sampling strategy. Since herding needs to have access to all samples of the class to select a representative set, the method passes over samples that have been processed before. This is against the online learning property from the desiderata of incremental learning. Instead of herding, reservoir sampling strategy can be deployed for online scenarios.

3.7 Feature rehearsal

3.7.1 What is it?

Feature rehearsal is one of the hottest research fields in incremental learning. The goal is to retain feature maps instead of real or generated images for rehearsal and in this way, bypass the privacy concerns and drastically decrease the memory requirements. Furthermore, it is pretty common in practice that some classes have a very limited number of samples and as a result, the deployed model fails to perform well enough on those tasks. This phenomenon is known as the class imbalance problem. Feature rehearsal strategy can solve this problem by generating any number of feature maps from real samples of those classes. Rehearsing generated images can also handle the class imbalance problem however generated images are not always optimal, generating real-like images is harder than generating feature vectors.

3.7.2 How to generate features?

Feature generation is done by stacking up several convolutional layers together, passing raw images through them, and in this way, decreasing the dimensionality of those images. There are myriad network architectures in the literature that serve this purpose. Feature extractor part of those networks has been utilized for generating representation for given images.

Some other works tried to utilize GAN architecture for feature generation although the objective of GAN methods is totally different. Xian et al. [12] use WGAN for feature generating rather than image. To make the network suitable for the classification task, authors add classification loss on the top of the WGAN generator. Makhzani et al. [13] propose an approach called adversarial autoencoder in their work, which is a combination of GAN and VAE. Mao et al. [14] test the performance of GAN discriminator as a feature generator against classic CNN classifier and concludes that CNN classifier is generally a better option for generating feature maps. Nevertheless, he also highlights the discriminator's ability to exclude the noise.

Iscen et al. introduce feature adaptation network to alleviate forgetting in the feature extractor part of a classification model in their work [15]. They deploy feature rehearsal strategy in their research, rehearsing features instead of images. The authors state that as new tasks arrive the model changes in a way that previously generated features become obsolete. In order to avoid that, authors present feature adaptation strategy in which previously learned feature descriptors are transformed to a new feature space for each new task. However, there is an obvious task recency bias in this approach since the feature vectors of prior tasks will be transformed most and ultimately lose the accurate representation of the images.

Keeping statistics of feature vectors is yet another method in this field. So instead of keeping feature descriptors for prior tasks, statistics like covariance matrix or mean vector are preserved in the memory, and in this way, authors gain further reduction in memory requirements. When required, new feature descriptors are generated from relevant statistics for each task. However, the performance of this approach is heavily dependent on the pre-trained network used.

3.7.3 How to evaluate generated features?

It is not an easy task to evaluate how well the generated feature vectors capture given images. Generated features can be fed to a deep upconvolutional neural network, like a GAN generator, to create images and check whether feature descriptors summarize all the relevant information effectively. Another strategy could be a performance comparison of the model using feature rehearsal against image rehearsal.

4. Datasets

4.1 CIFAR-10

In order to evaluate the performance of the model, we start with a relatively easy dataset. CIFAR-10 is one of the benchmark datasets we have used in this work. CIFAR-10 consists of 60K RGB images with 10 different classes, namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Image resolution is 32x32 for this dataset therefore training process should not take too long. Furthermore, models with low-resolution data can be trained on Google Colaboratory. Therefore, initial tests of the developed model have been tested using the CIFAR-10 dataset.

4.2 CIFAR-100

CIFAR-100 is another dataset that has been utilized in the experiments. As in CIFAR-10, the dataset consists of 50K training and 10K testing images. However, it has 100 classes each has 500 training samples which makes it harder to generate good quality images with respect to CIFAR-10.

FIGURE 4.1 CIFAR-10 DATASET

5. Methodologies

5.1 BiGAN

5.1.1 Overview

Bidirectional GAN [16], proposed by Donahue et al., is a very interesting approach that exploits GAN as a feature extractor. In classical GAN architecture, there are two main network parts, namely generator, and discriminator. The role of the generator is to create realistic images from noise vectors and fool the discriminator and the role of the discriminator is to distinguish real images from generated ones and not be fooled. This adversarial training leads to the development of a robust generator that can create images that are indistinguishable from real samples, at least from the discriminator side. Technically speaking, the generator learns to transform a noise vector to the feature space of real samples. In BiGAN instead, the authors bring an additional "player" into the training phase, an encoder. The loss function is altered in a way that it considers the encoder network as well and the discriminator network plays an adversarial game against both the encoder and generator.

5.1.2 Architecture

As noted previously, BiGAN architecture consists of three network components, namely generator, encoder, and discriminator, described in Figure 5.1. As in other implementations, the generator is seeded with random input sampled from the Gaussian distribution and expected to generate an image. The encoder, on the other hand, is seeded with samples from real distribution and it strives to produce representative feature descriptors. Input/output pairs of both network components are fed to the discriminator and its role is to decide whether the input pair comes from the generator or the encoder, as described in the figure below. In an ideal world, we would like to produce an encoder network at the end of the training which is the inverse model of the generator sub-network.

$$\min_{\mathcal{G}\mathcal{E}} \max_{\mathcal{D}} \{\mathbb{E}_{\mathbf{x}\sim P_{\mathbf{x}},\mathbf{z}\sim\mathcal{E}_{\Phi}(\mathbf{x})}[\log(\sigma(\mathcal{D}(\mathrm{x},\mathbf{z})))] + \mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}},\mathbf{x}\sim\mathcal{G}_{\Phi}(\mathbf{z})}[\log(1-\sigma(\mathcal{D}(\mathrm{x},\mathbf{z})))]\} \tag{5.1}$$

It may not be obvious from the description above since the generator and the encoder do not directly communicate with each other in any means. As reflected in the Eq. (5.1), neither encoder network nor the generator network observes the output of each other such as ***E(G(z))*** or ***G(E(x))***. Therefore, the encoder network learning an effective inverse mapping from real data distribution to the representative feature vectors seems unlikely. However, the authors assert that to fool the discriminator and reach the global minima, encoder and generator should learn the inverse mapping of each other.

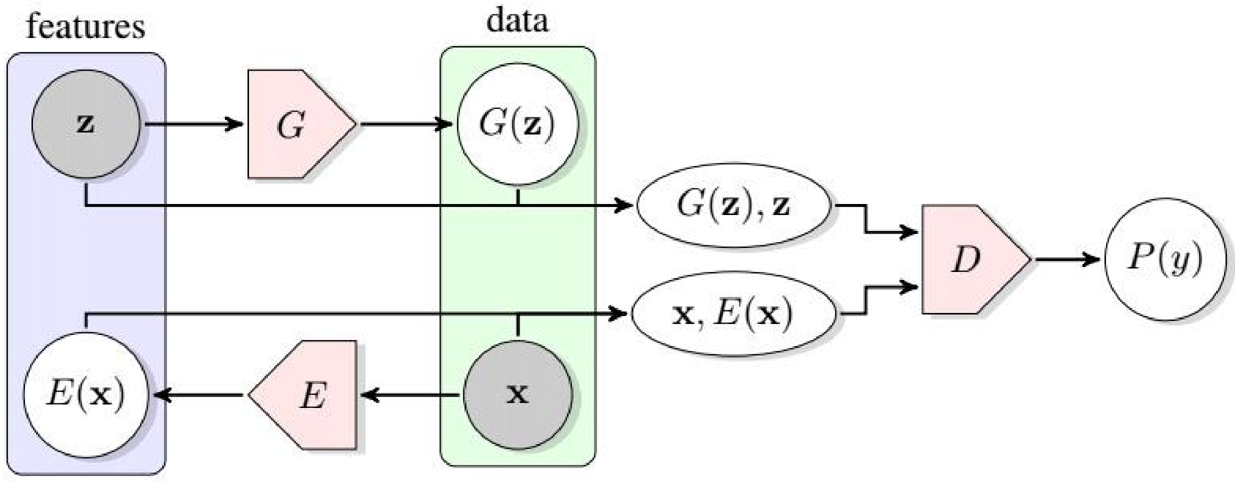


FIGURE 5.1 ARCHITECTURE OF BIGAN.

*Generator G learns to transform latent vector **z** to an image **G(z)** within feature space of real data distribution. Encoder E instead learns inverse mapping, from real distribution to latent feature space. Discriminator D receives input pair of **z** and **G(z)** from generator and input pair of **x** and **E(x)** from encoder, predicts whether input pair comes from generator or encoder.*

5.1.3 Limitations (DCGAN based)

The original implementation of BiGAN [16], proposed in 2016, is based on DCGAN architecture. While being a successful method in its time, it is an outdated architecture for the current time being. The architecture can only produce low-resolution images and fails to produce high-quality samples. The method is also prone to mode collapse in which the model can only generate images from a limited number of classes, or in the worst-case from only one class. Furthermore, a stronger generator network leads to a better encoder network in this implementation. To adapt the model for our use case, a more modern, solid GAN architecture should be adopted.

5.2 StyleGAN

5.2.1 Overview

StyleGAN architecture proposed by Karras et al. [17] is a revolutionary approach in the field of Machine Learning. StyleGAN approach aimed to bring some control over generated samples. Many approaches proposed before were able to successfully generate realistic images though they did not influence on details of the generated samples, like shape, color, pose, and so on. StyleGAN, on the other hand, tackles this problem by generating artificial images gradually, starting from a resolution as low as 4x4, increasing the resolution by the factor of two until the desired resolution, as high as 1024x1024.

In the early stages of image generation, the model learns base features that appear in low resolution, and as the resolution increases the model captures more specific and fine-grained features. The same strategy has been already adopted in ProGAN as described in Figure 3.2, so the method was not a breakthrough. However, ProGAN architecture does not have a strong influence on various details of the generated images. That's why StyleGAN brings some additional upgrades to make the network suitable for style control.

5.2.2 StyleGAN generator design

The generator of the StyleGAN (Figure 5.2) itself consists of 2 network components. Compared to other GAN methods, a noise vector is not fed directly to the generator network itself, rather it is processed by the mapping network initially. The goal of the mapping network is to minimize the phenomenon called feature entanglement. As the name suggests, feature entanglement is the state where some features are being entangled to each other, either being present together or not appearing in a generated image. If the dataset used is being dominated by the samples in which feature A and feature B appear at the same time, a GAN network will high likely fail to generate images in which only one of those features exists. For example, the task is to generate human face images, and the dataset consists of male portraits with short hair and female portraits with long hair predominantly. Feature of sex and feature of hair are entangled with each other in this state and the model will fail to produce an image of a male with long hair and an image of a female with short hair. Therefore, in typical GAN architecture, changing the seed vector to obtain a long hair in a generated image would also lead to a change in gender.

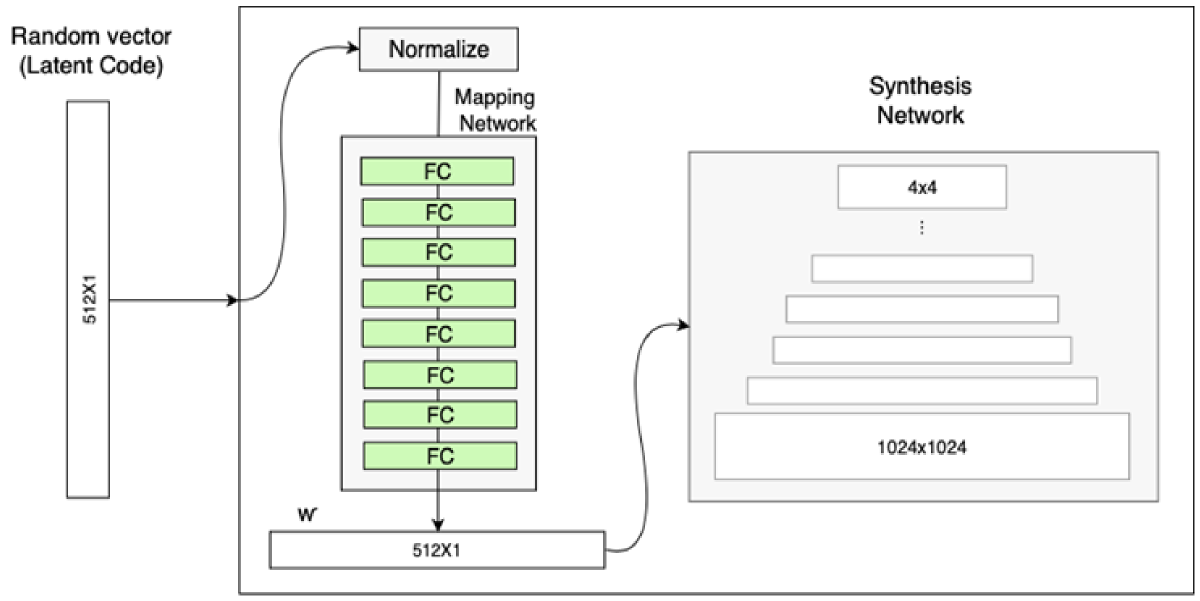


FIGURE 5.2 ARCHITECTURE OF STYLEGAN GENERATOR.

The first component of the generator networks serves to diminish the effect of feature entanglement, not to limit generated images with certain combinations of features, at the same time to avoid feature combinations that do not appear in the training dataset at all (a soldier with a long hair). The initial noise vector ***z*** is fed to this network called mapping network. The mapping network consists of eight fully connected layers, takes an input size of 512x1, outputs an intermediate vector ***w*** in the same size. Later, intermediate vector ***w*** is fed to the synthesis network to incrementally generate an image.

As mentioned previously, StyleGAN follows a progressive method for image generation. It starts with a very simple task in the beginning (creating a low-resolution image) and gradually increases the complexity of the problem (increasing the resolution step by step). The synthesis network consists of upconvolutional layers starting with a resolution of 4x4 and building its way up to 1024x1024, increasing the resolution by the factor of two in each stage. In lower resolutions, from 4x4 to 8x8, the model learns high-level features (coarse styles) such as pose of the subject, shape of the face, shape of the hair, even eyeglasses. In mid resolutions, from 16x16 to 32x32, the model inherits intermediate features such as small-scale changes in facial features and hairstyle, eyes open or closed, and so on. It is worth noting that, high-level features learned in previous stages like pose, general facial features are preserved in this stage. And finally, from 64x64 to 1024x1024, the network learns finer styles like the color of the eyes, tone of the skin, and other microstructures. This method of training is robust against mode collapse which is the major problem observed in many GAN architectures like DCGAN, as highlighted earlier.

5.2.3 Adaptive Instance Normalization

AdaIN (Adaptive Instance Normalization) is the style module applied to intermediate vector $\boldsymbol{w}$ in each resolution level. In each layer, an input feature map is passed through a convolutional layer and then transformed by the operation called AdaIN. First, AdaIN normalizes the input feature map and later adds a style vector as a bias term. Karras et al. explains the procedure as "Learned affine transformations then specialize $\boldsymbol{w}$ to styles $\boldsymbol{y} = (\boldsymbol{y_s}, \boldsymbol{y_b})$ that control adaptive instance normalization (AdaIN) operations after each convolution layer of the synthesis network g" in [17].

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i} \tag{5.2}$$

One may ask why not use the standard batch normalization technique rather than adaptive instance normalization. Actually, batch normalization does the reverse operation desired in this work. Batch normalization normalizes a batch of feature vectors instead of a single one, and consequently resulting in the batch of feature vectors centered around a single vector, preventing possible variations in styles of generated samples. AdaIN, on the other hand, operates on a single feature vector and different combination of $(\boldsymbol{y_s}, \boldsymbol{y_b})$ affine parameters result in variations in feature statistics and therefore, output images in diverse styles.

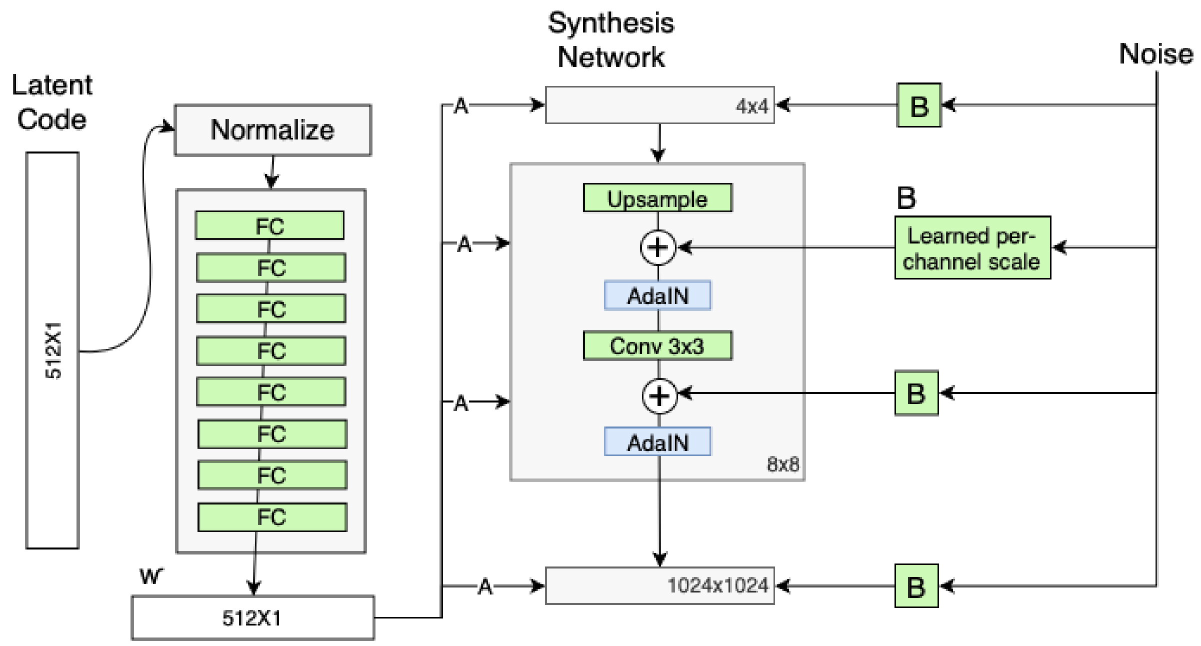


FIGURE 5.3 SYNTHESIS NETWORK OF STYLEGAN GENERATOR

5.2.4 Stochastic variations

After instilling styles to the generated images, we would also like to add small details to encourage variations in generated outputs and produce more realistic images. In the case of human face generation, we would not like to end up with plain portraits rather we would like to have stochastic variations such as freckles, wrinkles, skin pores. To achieve so, the authors add a noise vector in each resolution layer. In Figure 5.4, results of addition of noise to all layers (a), no noise in any layer (b), noise in only high-resolution layers (c), noise in only low-resolution layer (d) have been described by Karras et al. It is obvious from the images that not adding noise vector produces worst results. The difference between (c) and (d) is subtle, adding noise to coarse layers results in bigger curls in hair while in the case of fine layers, smaller curls are observed.

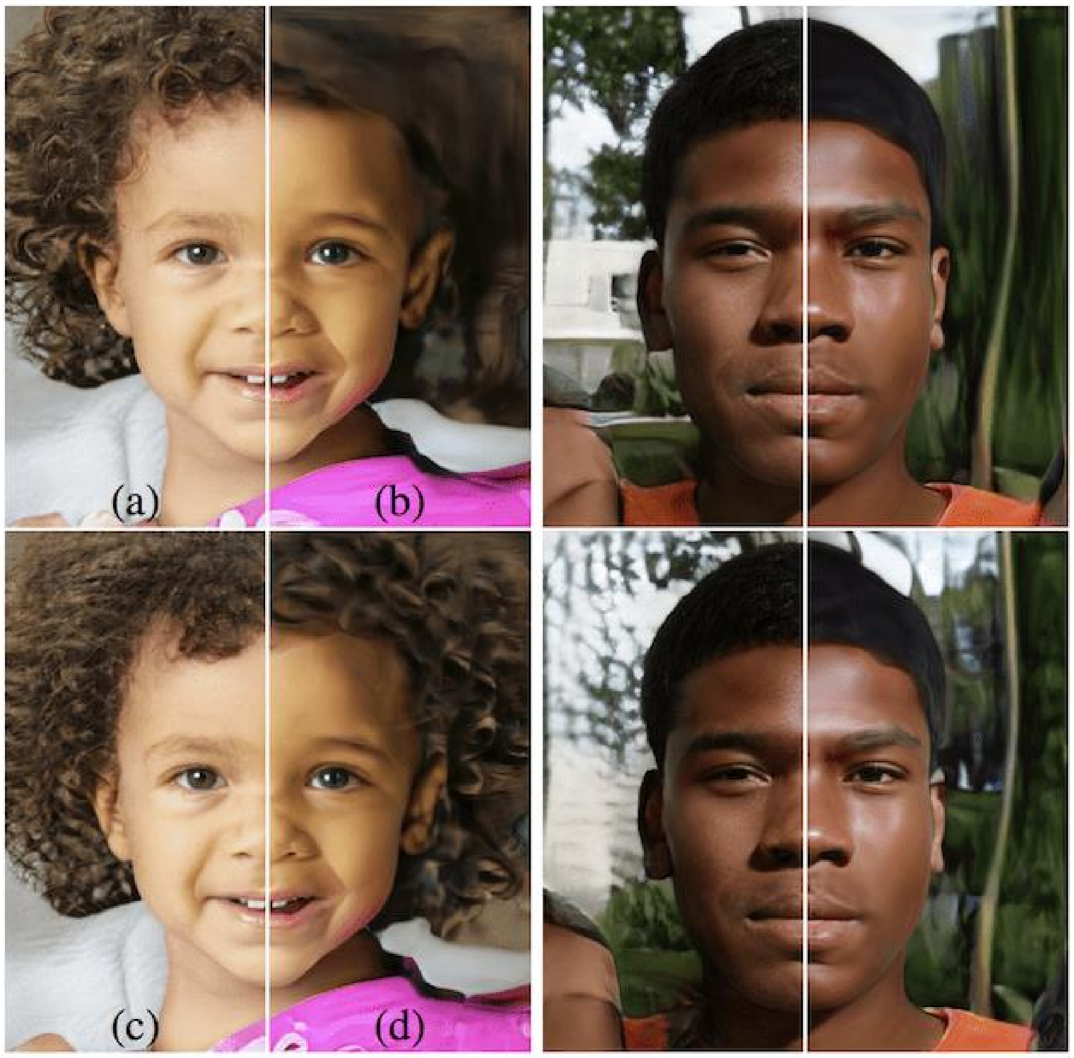


FIGURE 5.4 EFFECTS OF ADDING NOISE TO THE DIFFERENT LAYERS OF STYLEGAN GENERATOR NETWORK.

(a) adding noise vector to all layers (b) not adding noise vector to any layer (c) noise vector only in high-resolution layers (d) noise vector only in low-resolution layers

5.2.5 Truncation trick

This method involves using different distributions for sampling noise vectors in training and testing phases. Taken from BigGAN [4], the truncation trick helps to avoid low probability density regions, which can deteriorate the quality of outputs, in initial or intermediate noise vectors. "We call this the Truncation Trick: truncating a z vector by resampling the values with magnitude above a chosen threshold leads to improvement in individual sample quality at the cost of reduction in overall sample variety", authors of BigGAN explain. However, Karras et al. use the truncation trick only in low-resolution layers in order to avoid any changes to finer details.

5.2.6 StyleGAN issues

- Bloblike artifacts

Water droplet-like artifacts are observed in generated samples of original StyleGAN. Authors pinpoint the root cause of these artifacts in their paper, "Analyzing and Improving the Image Quality of StyleGAN". "We pinpoint the problem to the AdaIN operation that normalizes the mean and variance of each feature map separately, thereby potentially destroying any information found in the magnitudes of the features relative to each other. We hypothesize that the droplet artifact is a result of the generator intentionally sneaking signal strength information past instance normalization: by creating a strong, localized spike that dominates the statistics, the generator can effectively scale the signal as it likes elsewhere. Our hypothesis is supported by the finding that when the normalization step is removed from the generator, as detailed below, the droplet artifacts disappear completely", explain Karras et al in [18]. Nonetheless, authors do not hide their wonder from the fact that the discriminator cannot distinguish affected images from real data samples.

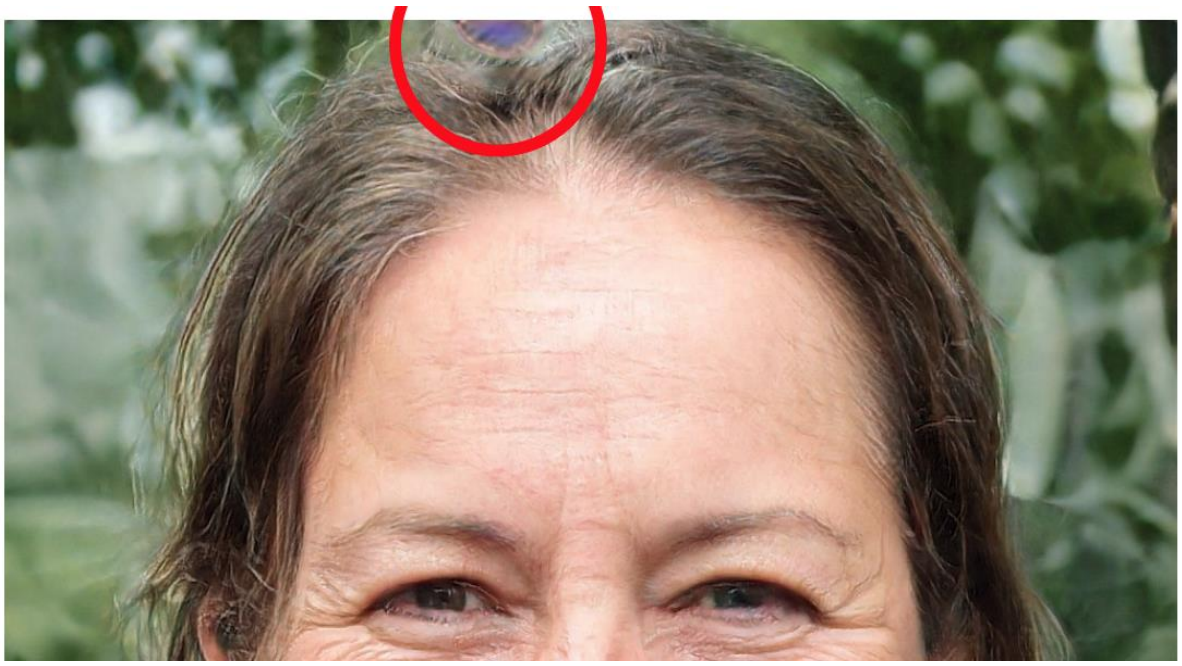


FIGURE 5.5 BLOBLIKE ARTIFACTS IN STYLEGAN OUTPUTS

- Progressive growing

Karras et al. deploy progressive growing methodology proposed in the ProGAN paper by himself in the StyleGAN model. The idea is to start with minimal resolution such as 4x4, train the model and generate images with that resolution and once the objective is reached append a new layer with twice resolution to the model. However, this feedforward design has its own drawbacks. Specifically, progressive growing models have a strong preference for the locality of the features in images. When playing with features of low-resolution layers, some details that are subject to change remain unaffected. As described in the Figure 5.6, while the face of the subject is rotated, the center locations of the teeth are not modified.

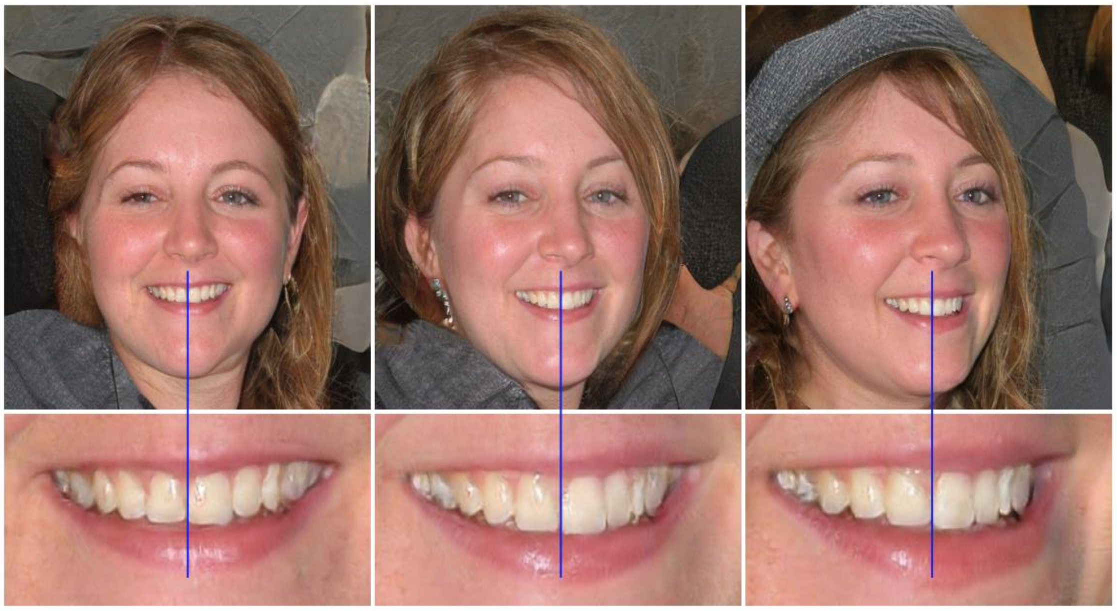


FIGURE 5.6 FLAWS OF PROGRESSIVE GROWING

5.3 StyleGAN2

To address the flaws of StyleGAN architecture, authors publish their new work under the name "Analyzing and Improving the Image Quality of StyleGAN" [18] where they bring some modifications and improvements to the existing method. Especially, the authors pinpoint the root cause behind undesired spots appears on some generated images. Furthermore, they further simplify some units from the generator network that are deemed unnecessary.

5.3.1 Weight Demodulation

As described previously, AdaIN instance normalization leaves unwanted artifacts on generated images, and surprisingly the discriminator network fails to label those images as fake.

The authors propose an alternative normalization technique called weight demodulation to eliminate those artifacts from output images. First, previous convolutional weights are scaled using the modulation stage followed by 3x3 convolution layer as described in Eq. (5.3) and this stage corresponds to the "mod" stage in the figure below. In the below equation, s_i corresponds to the scale factor for the ith feature map, w and w' the convolutional weights before and after modulation and *j* and *k* signify convolution and its spatial footprint.

$$w'_{ijk} = s_i \cdot w_{ijk}$$

(5.3)

After the modulation stage, convolutional weights are normalized with the demodulation layer, described in Eq. (5.4). Although weight demodulation is not mathematically equivalent to the instance normalization, it serves the same purpose and even achieves better results by removing unwanted artifacts. Apart from removing water droplet-like spots, the proposed technique brings a 40% training speedup, from 37 images to 61 images per second.

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} w'_{ijk}{}^{2} + \epsilon}$$

(5.4)

Additionally, the authors have done further modifications to the generator network. Intermediate latent vector entering to the synthesis network does not go through mean/standard deviation normalization and noise vector is not added to the latent vector at the beginning. Mean normalization is deemed unnecessary by the authors and is removed from all resolution layers. Furthermore, noise vector is added to the input outside style module.

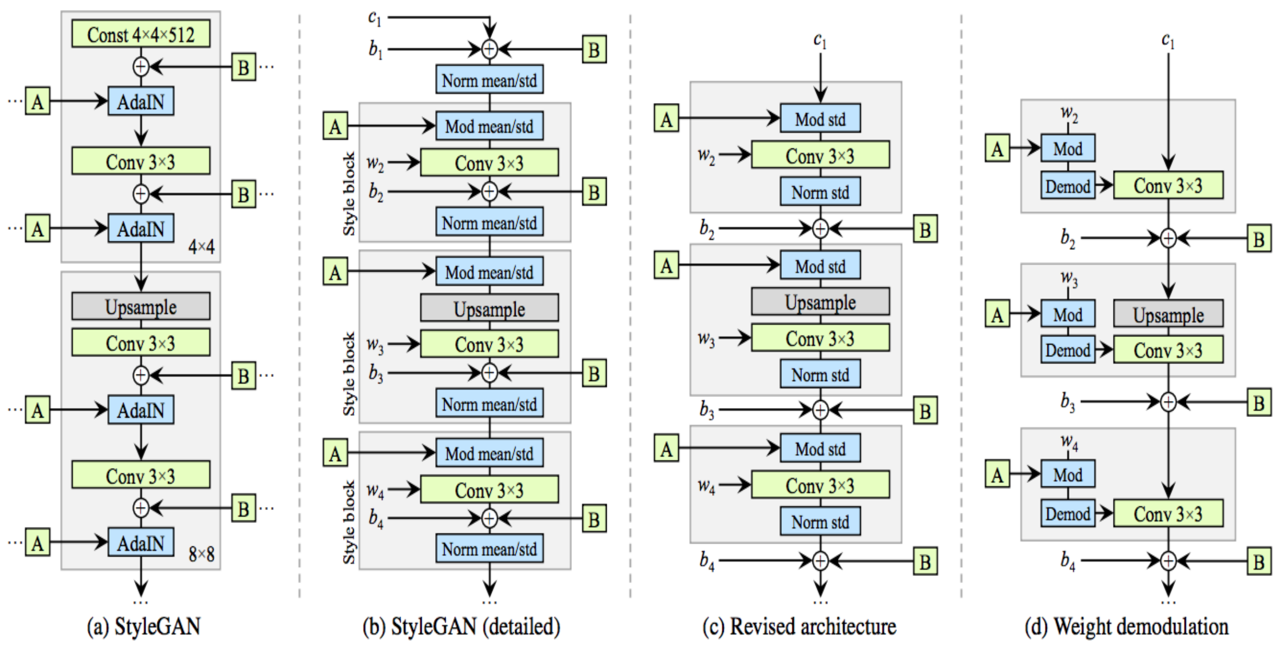


FIGURE 5.7 IMPROVEMENTS IN STYLEGAN2

5.3.2 Removing progressive growing

Progressive growing model property is another root cause for sub-optimal images generated by StyleGAN. Changing the pose of the subject does not influence the location of teeth in generated outputs. In StyleGAN2, authors do not train the resolution layers gradually because low-level features are fixated to the same location during training of early layers and show resistance to any change. Therefore, Karras et al. adopt another strategy inspired by residual connections of the RevNet network architecture. The idea is very simple and vastly utilized in the literature. Basically, the output of a certain network layer is fed directly to the input of another layer which is not subsequent. The technique is used to allow gradients to move forward through a deep neural network without passing through non-linear activation functions. The motive behind this is to prevent gradients from vanishing since non-linear functions by their nature cause information loss. In this work, the authors deploy bilinear filtering for upsampling and downsampling layers, an approach that resembles skip connections.

5.3.3 Lazy Regularization

The term "lazy evaluation" is the essence of modern applications such as big data, deep neural networks. For example, lazy evaluation is one of the main properties of the TensorFlow library. To be specific, the TensorFlow graph can be considered as a roadmap of computations, and each individual computation is called operations (Ops). Operations are not executed until we explicitly run a session. In this way, we avoid running unnecessary intermediate operations and consequently, gain performance increase. Karras et al. follow a similar method called "lazy regularization" in which they calculate cost function and update network parameters only once in 16 mini-batches. Empirical results showed that updating cost function in this frequency does not have a negative effect on model performance, however, a significant reduction in computational cost has been achieved.

5.4 StyleGAN2-ADA

Karras et al. came up with even further improvements to their already brilliant GAN design with their paper “Training Generative Adversarial Networks with Limited Data” in 2020 [19]. The authors try to adapt their StyleGAN2 network for use-cases where the training dataset has limited samples. In practice, GAN networks require 40~50K data samples to achieve stable training, to imitate real data distributions for high-quality, high-resolution outputs. However, not every dataset has a huge number of image samples, and collecting a vast number of images in certain fields is not possible. For example, GAN applications are becoming widespread in the medical field, nonetheless finding public datasets with a big number of samples is a challenging task due to privacy concerns.

When training a GAN architecture with a small number of samples, the discriminator sub-network tends to overfit to the training data units and the feedback quality it gives to the generator model deteriorates and, eventually becomes meaningless. As a result, the generator model fails to produce diverse, high-quality samples. Data augmentation is the technique to be referred for small dataset problems in almost all subfields of machine learning. Especially in classifications tasks data augmentation is used for both increasing the size of the training data and also prevent overfitting, by adding random noise to the samples. But when it comes to the GAN training, data augmentation technique can leave unwanted artifacts on generated samples. The phenomenon known as augmentation leaking happens when input data is augmented in the same fashion as in classification models. To further elaborate, the generator network ends up generating augmented samples such as rotated, flipped, blurred images when try to imitate data distribution, obviously a property that we would not desire from GAN architectures.

In this work, Karras et al. try to artificially expand the training dataset to avoid overfitting in the discriminator while preventing augmentation information from leaking to the generator network. The authors state that a transformation function for data augmentation should be invertible to achieve "non-leaking" augmentation. For example, pixel blitting, color transformations are among the invertible augmentation functions, the model can see through the affected images to understand the real distribution. However, unlike humans, a neural network model cannot discern true form if images flipped either 0°, 90°, 180°, 270° with equal probability and as a result, generated samples can come with different angles. However, if augmentation strength ***p*** is picked that is the probability of transformation within the range of 0 and 1, then training samples would be dominated with non-flipped samples and a GAN network would know the correct orientation of an image.

Therefore, performed augmentations should be invertible, the model should be able to differentiate transformed samples from plain samples. The authors categorize deterministic mappings, additive noise, image or color rotations and projections, cutout as "non-leaking" augmentation techniques. Authors develop an augmentation pipeline with fixed order 18 transformations and use the same value of augmentation strength ***p*** for all transformation functions.

They also highlight the fact the optimal value of augmentation strength ***p*** is dataset dependent. Several experiments done by the authors on image datasets with various sizes point out that any augmentation to a big dataset (more than 100K) is harmful, while for datasets with 10K size ***p*** value should be positive but not so high.

Finally, the authors propose an adaptive augmentation method to set ***p*** value dynamically. Specifically, the method monitors the discriminator network for any overfitting signs and adjusts the augmentation strength accordingly.

5.5 Conditional GAN

GAN models are capable of generating plausible samples by imitating the real data distribution. By feeding random noise a fully-fledged GAN generator, very realistic images can be obtained. Nevertheless, users have no option to control the class of data samples to be generated when using a classic generator model (like DCGAN generator) other than trying to figure out the complex relationship between input latent space and generated outputs. No need to say, image generation conditioned to a class label is a particularly desired property of generative adversarial networks. To achieve this property, additional information about the class label incorporated with input samples can be used in the training process and this addition can bring several advantages such as more stable training, faster convergence, generated images with higher quality. "Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. […] We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer", explain Mehdi Mirza and Simon Osindero in their paper [20].

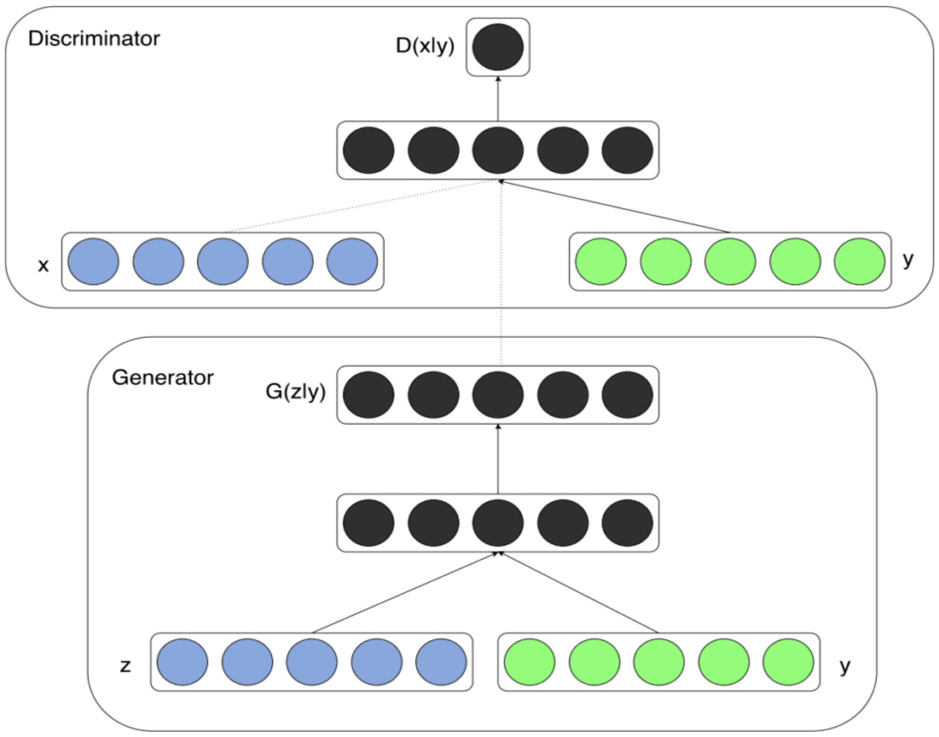


FIGURE 5.8 CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS

5.6 BiStyleGAN

In this work, we aimed to investigate the potential contributions of generative adversarial networks in the incremental learning field. Among three general strategies in incremental learning, models based on rehearsal strategy generally excel both in performance and align with the desiderata of incremental learning, do not require additional resources with the increasing size of the data, or do not exhibit resistance to learn, a trait seen in regularization-based incremental learning approaches. That is why we decided to stick to rehearsal strategy, especially feature rehearsal strategy that brings many advantages in real-world scenarios. Combining the general network design of the bidirectional GAN [16] and robust network units of Style-based GAN [19] we propose a hybrid approach for representation learning to be used for feature rehearsal strategy.

The proposed network model can further be extended for a conditional model which can generate images and features for any desired class of a given dataset.

5.6.1 Architecture

Donahue et al., published “Large Scale Adversarial Representation Learning” [21] which is the extension of the original bidirectional GAN [16]. In this work, they highlight the fact progress in image generation quality brings improvement in representation learning under BiGAN network design. Therefore, they bring some improvements over backbone architecture, deployed BigGAN generator for image generation, and extended the discriminator network into three sub-networks. For the loss function, they stick to the one they proposed in their work of bidirectional GAN with some modifications. However, as explained in section 3.1, BigGAN network architecture achieves high-quality images with a simple but costly strategy. The strategy of BigGAN is scaling up existing GAN models by increasing network capacity and batch size. This method is not convenient for incremental learning models that are designed for real-world applications.

We adopt the general network design of bidirectional GAN with the improvements proposed to the discriminator network in the BigBiGAN paper [21]. However, we change the backbone network architecture with the one that has higher quality image generation capabilities with a smaller network size namely StyleGAN2.

- **Generator**

The generator sub-network of our model is composed of the StyleGAN2 generator. This generator, developed by Karras et al., is adaptive to both low and high-quality image datasets. Compared to BigGAN generator StyleGAN2 generator produces better quality images and furthermore, the network is more compact with respect to BigGAN. While the original network only outputs a batch of image samples, we modified the network to return also intermediate latent vector formed after passing an input latent vector through the mapping network of the generator.

Because in the loss function of our model we utilize both input and output of the encoder and the generator sub-networks as proposed in bidirectional GAN paper.

- **Encoder**

For the encoder sub-network, we developed a ResNet model from scratch to be used for representation learning. ResNet network family is known for successful implementation of skip connections which addresses the vanishing gradient problem caused by non-linear activation functions. The size of the encoder model should be proportional to the generator model size. StyleGAN2 model is a brilliant model that already adapts to the image resolution of the dataset. As a starting point, we decided the depth of the ResNet model to be 101 for the CIFAR dataset. However, we later switched to more compact model. Multi-layer perceptron with skip connections is attached to the end of our ResNet model. Our encoder network takes a batch of real data samples as input and produces feature embeddings with various size, one of the hyperparameter we optimized during our experiments. Input and output dimensions of the generator and the encoder network are identical.

- **D_F**

D_F model can be thought of as a standard discriminator network in GAN models. It takes an input of real sample used in encoder ***E*** or generated sample from generator ***G***. For this discriminator sub-network, we again used the StyleGAN2 discriminator with adaptive discriminator augmentation. As explained previously, the feature is specifically designed for limited data applications, and in our experiments, the adaptive augmentation technique (section 5.4) was quite helpful.

- **D_H**

D_H network on the other hand takes feature vectors as an input. This network is relatively small compared to the ***D_F*** network, it consists of an 8-layers multi-layer perceptron with ResNet style skip connections. Since we only process low-dimensional feature embeddings in this model, we designed a network small in size. Also, there is no need for convolutional layers for processing feature vectors. It should be noted that ***D_F*** and ***D_H*** sub-networks receive the input data at the same time. The input to these sub-networks can come either from the encoder network (***x*** to the

D_F and ***E(x)*** to the ***D_H***) or from the generator network (***G(z)*** to the ***D_F*** and ***z*** to the ***D_H***) as described in Figure 5.9.

- **D_J**

D_J receives output score from ***D_F*** and ***D_H*** produces a joint score for these two sub-networks. ***D_F*** and ***D_H*** also output separate scores and all three are added together to calculate the loss function. The network architecture of the ***D_J*** sub-model is identical to the ***D_H*** network.

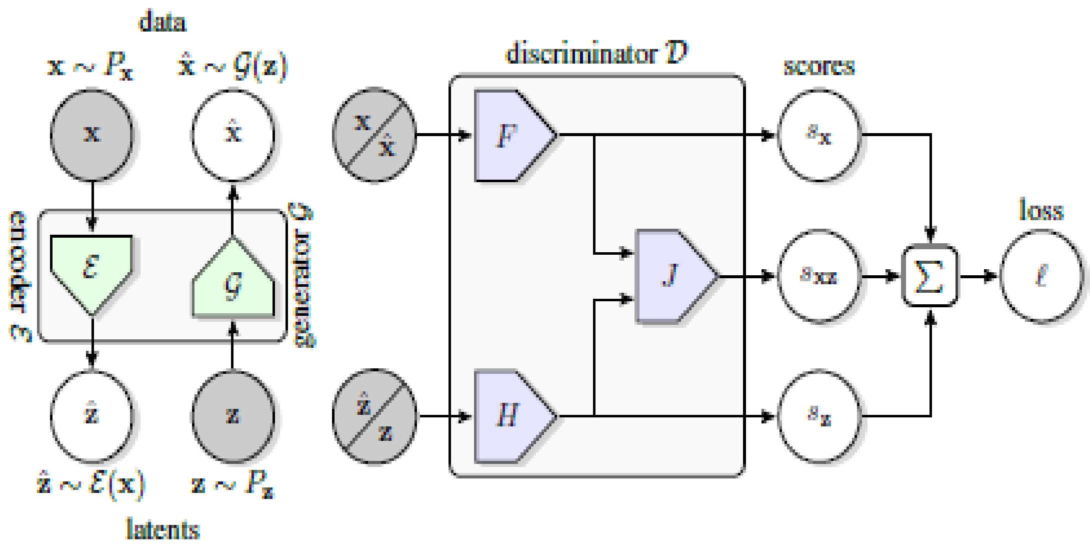

data
$\mathbf{x} \sim P_{\mathbf{x}}$
$\hat{\mathbf{x}} \sim \mathcal{G}(\mathbf{z})$
encoder $\mathcal{E}$
generator $\mathcal{G}$
$\hat{\mathbf{z}} \sim \mathcal{E}(\mathbf{x})$
$\mathbf{z} \sim P_{\mathbf{z}}$
latents
discriminator $\mathcal{D}$
F
J
H
scores
$s_{\mathbf{x}}$
$s_{\mathbf{xz}}$
$s_{\mathbf{z}}$
$\sum$
loss
ℓ

FIGURE 5.9 THE DESIGN OF THE BIGBIGAN NETWORK

5.6.2 Loss function

As mentioned in section 5.6.1, three parts of the discriminator network output three separate loss terms, namely S_x , S_z , and S_{xz} (described in Figure 5.9). The authors used only one loss term (coincides with S_{xz} of the new loss function) in the previous approach to tie the data and latent distributions together. In the improved loss function, separate outputs of data and latent discriminators are also taken into account. By doing so, we can have control to guide the training process towards desired direction. That is, assigning appropriate weights to the unary loss terms in the loss function can provide the ability to prioritize the task of representation learning over image generation or vice versa. Intuitively, giving higher weight to the S_x unary loss term would make more sense since the generated images are more representative than generated feature embeddings. However, in our training, we explicitly picked equal weights for S_x and S_z unary loss terms as the main objective of this research is to develop a robust neural network for feature rehearsal.

5.6.3 Hyperparameters

Since our model is a combination of two different network architectures designed from scratch, we cannot simply "transfer-learn" the optimal network hyperparameters from any of those models. Instead, we tried to check a different set of parameters to find the optimal group for our approach. Nevertheless, we decided to follow general patterns in those approaches, especially in the BigBiGAN since it's the main backbone architecture in our work.

The most fundamental hyperparameter in machine learning models is probably the learning rate. We update the network parameters of the generator and the encoder network jointly so the learning rate for these two sub-networks is the same. In the paper of the BigBiGAN, learning rate of the discriminator is also same with the other two, therefore, we follow that rule. However, in our experiments, we also checked the performance of our model with an unequal learning rate. For the batch size, we mainly picked higher numbers like 128, 256 since the image resolutions of the trained datasets are not high.

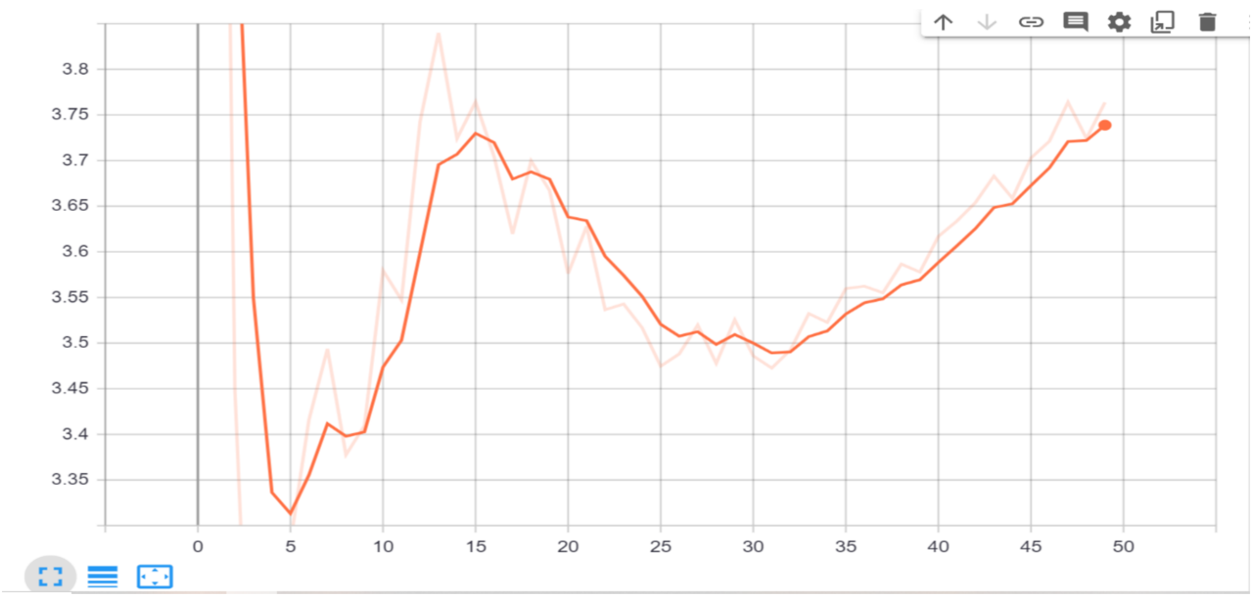
One another interesting hyperparameter is the frequency of updating network weights and the ratio of this value among the sub-networks. This value is contradicting between the StyleGAN2-ADA and the BigBiGAN approaches. In the former, the generator network is updated four times more frequently than the discriminator network while in the latter this value is two.

In StyleGAN paper, the size of feature embeddings presented to the generator network is 512x1, but this number is smaller in the BigBiGAN network. For our network design, the input of the generator and the output of the encoder should be same in size. And the outputs generated from the encoder network after the training are used for our classification task therefore the dimensionality of feature embedding is an important parameter.

There are also many other network-specific hyperparameters such as augmentation strength, lazy regularization, the ratio of importance between unary loss terms that are mentioned earlier as well as countless ways to design the sub-networks.

5.6.4 Evaluation metrics

In standard machine learning classification approaches, the loss function is a good indication of performance, and it is monitored constantly to detect any underfitting or overfitting issues. In GAN training, however, the loss function does not show the whole picture, it is not possible to objectively evaluate the quality of the model or the progress of the training from loss alone. Figure 5.10 depicts the loss values of the sub-networks recorded by Tensorboard in one of our experiments. As seen from the figure, there is a constant fluctuation in loss values. There is a general trend of decreasing loss value in generator & encoder and increasing loss in discriminator network and this trend was generally suggesting stable training in our experiments. Nonetheless, this information does not suffice to effectively evaluate the performance of the model, such as the quality of generated images or feature embeddings in our case.



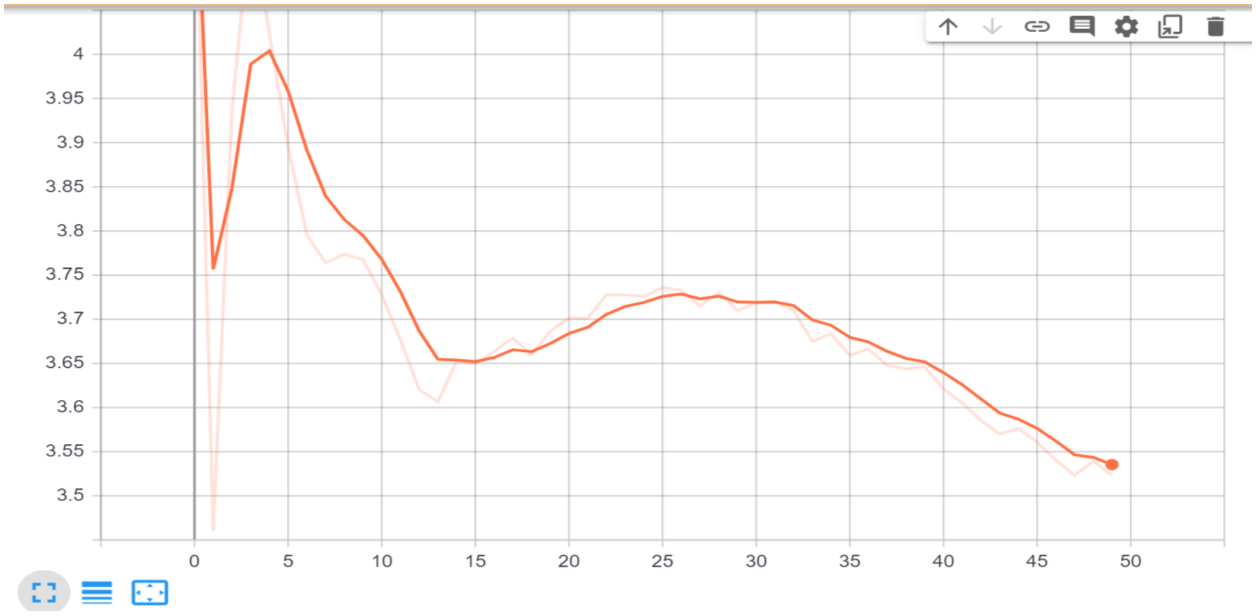


FIGURE 5.10 LOSS VARIATIONS DURING GAN TRAINING

The first image describes the variation of loss value in the discriminator network while the second image depicts joint loss value in generator and encoder networks after 50 epochs of training.

Several quantitative measures have been proposed such as inception score and Fréchet inception distance to provide a robust assessment of GAN models. Clearly, these metrics are designed for the main objective of the GAN architectures which is image generation however we exploit a GAN model for a different task, representation learning. Considering the hypothesis of Donahue et al. in their paper that to reach the optima the encoder network should learn the inverse mapping of the generator network; we take the images generated by our generator network as the main performance indicator of our encoder network during the training process. Figure 6.2 describes the generated images during different epochs of the training procedure. As it is seen from the figure, the quality of the generated images increases as the number of epochs raises. Therefore, monitoring the generated images helps us to keep track of the training and make an interception if necessary.

Furthermore, once we finish the training of generator and encoder models, we feed real images to our encoder network to produce feature embeddings and then use those feature embeddings as the input to our generator network. Images generated by this process provide valuable insights about the representation learning capability of our encoder model.

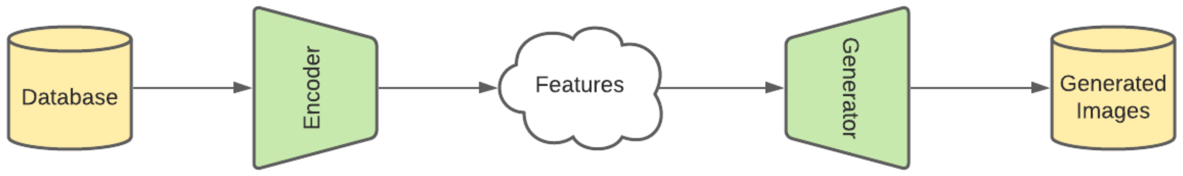


FIGURE 5.11 GENERATING IMAGES IN VAE-LIKE ARCHITECTURE

6. Experiments & Results

6.1 Overview

In the initial stages, we mainly did our experiments on the CIFAR-10 dataset, but later we also tested our model with the CIFAR-100 dataset. In generative adversarial networks training, a huge number of samples is required to generate satisfactory results. Karras et al. alleviated this requirement with their adaptive discriminator augmentation method however the model still needs around 10K samples. The number of classes in the dataset also contributes to the training process. The more class we have in our dataset, the harder to produce high-quality images. The number of samples on mentioned datasets is equal however the number of images per class is ten times more in the CIFAR-10 dataset. Therefore, we achieved better results with the CIFAR-10 dataset. Furthermore, since we did our early experiments all on the CIFAR-10, our model is finetuned to perform on CIFAR-10.

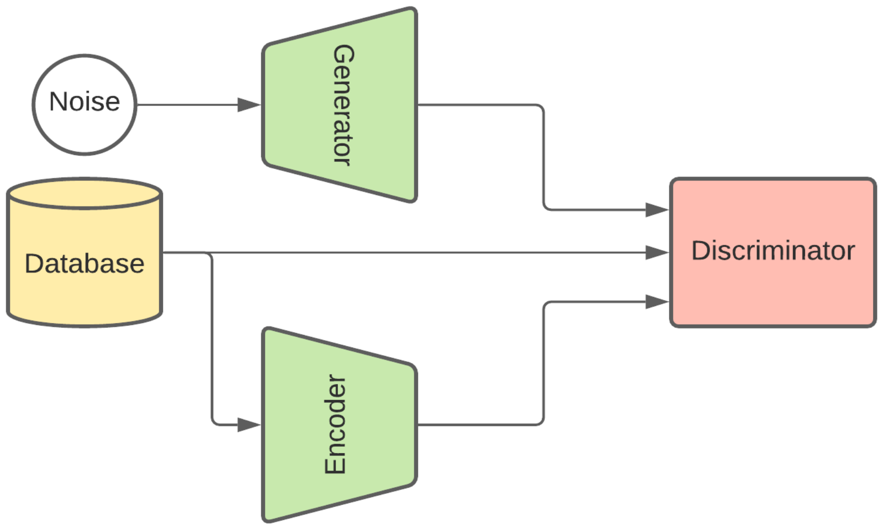
Figure 6.1 depicts the general procedure of our training approach. First, we train our proposed model for each task, observing generated images of the generator sub-network. Once we see "acceptable" output images, we stop the training process and detach the encoder sub-network from the model. Then we feed the samples of the trained task to the detached encoder network and produce a low-dimensional representation of the images. Then we feed the generated feature embeddings to the classifier (nearest mean classifier as in [10] or support vector machine). We train our proposed model for each subsequent class and present the feature embeddings to the final classifier.

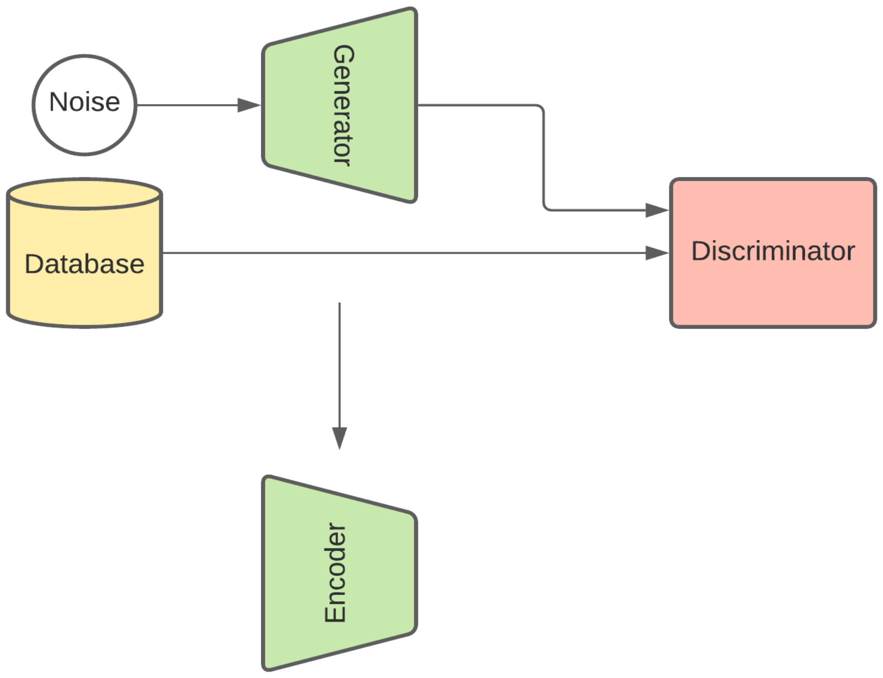
Furthermore, we have several options for the ratio of feature embeddings to keep from the previous tasks. We evaluated the performance of the final classifier trained with [20%, 50%, 100%] feature embeddings from all the classes. Instead of picking random feature representation from previous classes, we rank the feature embeddings by the distance from the mean for each class and sample the desired proportion of feature vectors from each task. By doing so, we produce a representative subsample of data for each class. This method brings further reduction in memory requirements with minimal performance loss as well as serves to discard any outlier in the generated feature vectors.

Considering the vast number of hyperparameters to play with, different possible NN architectural choices with various configurations, and more importantly limited computational resources of Google Collaboratory, it was not feasible to finetune a model designed from scratch to an additional, relatively challenging dataset.

6.2 Finetuning

As a starting point, we try to follow general patterns on network design of StyleGAN and BigBiGAN however we also had to deviate from the optimal parameters there. For learning rate, we started with 2e-4 for all sub-networks, the value advised in [21] however we later found that the discriminator with 1e-4 learning rate produces slightly better results. For the augmentation strength and transformation layers, we stick to the strategy proposed in [19] since transformation layers and augmentation strength were already tested there. For the update frequency of sub-network weights, we follow the rule advised in [21] and update the discriminator twice while updating the generator and encoder once. For the dimensionality of the output feature vector, we start with the value of 512 as proposed in [18], however, we later figured out that for the CIFAR datasets we can pick a lower dimension since a higher value did not always bring good performance. We also have some modifications in the encoder sub-network. Initially, we start with ResNet-101 network design but for the low-dimensional images of CIFAR datasets, we later switch to a more compact ResNet model.


Noise
Generator
Database
Discriminator
Encoder


Noise
Generator
Database
Discriminator
Encoder

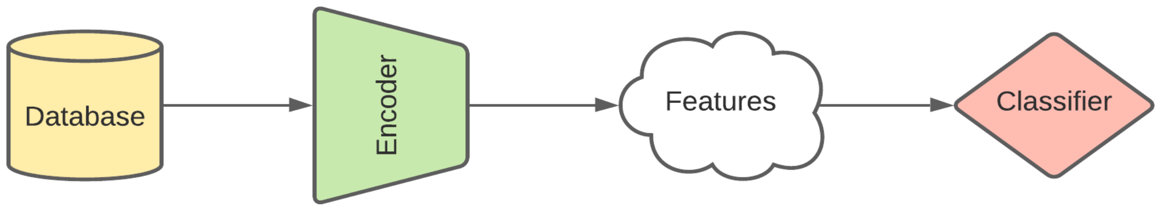
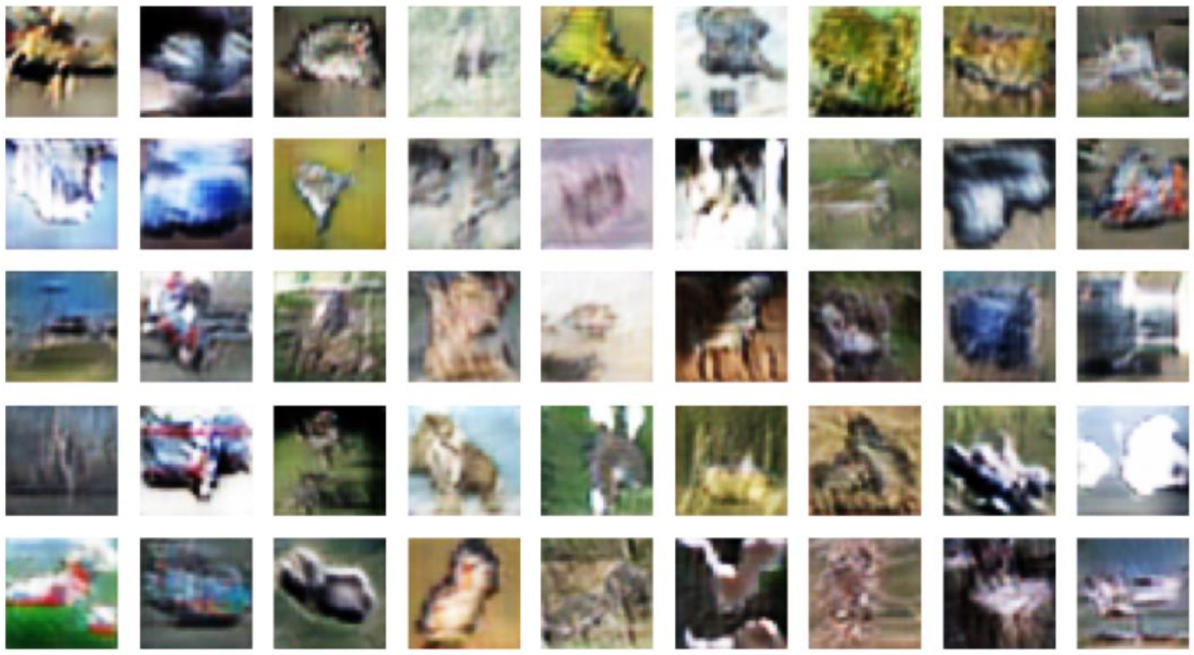

Database
Encoder
Features
Classifier

FIGURE 6.1 OUR TRAINING PROCEDURE

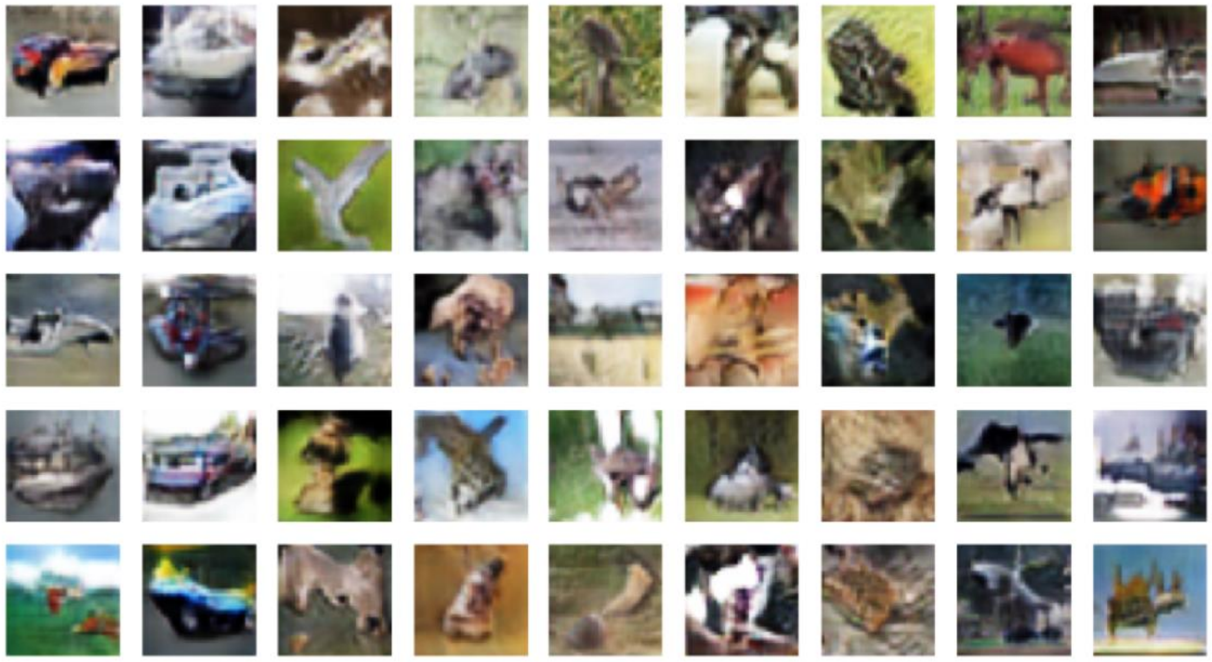
For each task, we start by training our proposed model to build a solid feature extractor. At the end of the training, we detach the feature extractor from our model, feed the input data to it, generate low-dimensional feature embeddings that can summarize high-dimensional input data. The last step is to train a shallow network to classify the data based on generated feature embeddings.

6.3 BiStyleGAN outputs

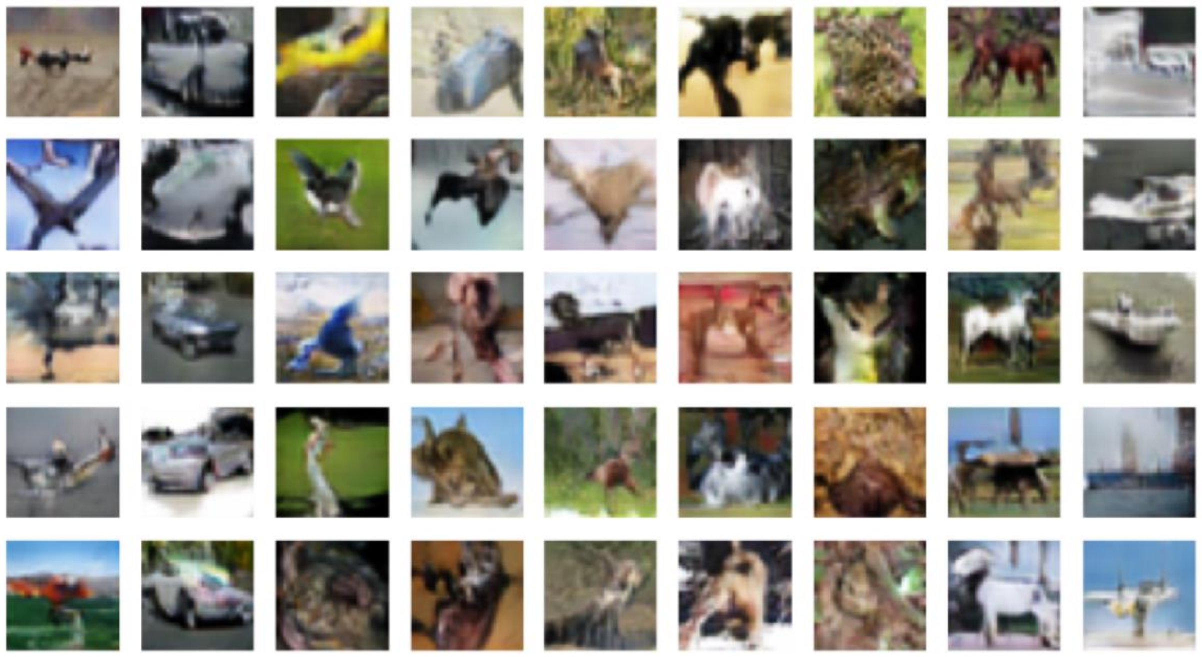
As mentioned earlier in section 5.6.4, keeping track of loss metrics is not enough to evaluate the performance of a GAN model. And considering the fact that the main goal of this research deviates from the objective of a GAN architecture, we did not use inception score and Fréchet inception distance which are selected as main metrics for evaluation of the model in StyleGAN [18]. Instead during the training process, we constantly checked the output samples generated by the generator network of our approach based on the hypothesis of Donahue et al. mentioned above. One fact should be highlighted that we cannot expect excellent output images from our model since we optimize two loss terms instead of one which is the case in other GAN applications. Our model tries to both generate good images and at the same time, representative feature embeddings. Again, by setting relative weights to unary loss terms we can prioritize image generation over-representation learning. Intuitively, we selected equal weights for these tasks since our main task is representation learning but at the same generated images are more representative than generated feature vectors. Figure 6.2 depicts images produced by the generator model in different epochs from the CIFAR-10 dataset. Generated images somehow capture the semantics behind the real samples. The model especially produces good images from "car" and "horse" classes. We trained our proposed model in different settings. We fed the whole training dataset of CIFAR-10 to our model to produce a feature extractor. Later we fed the test dataset to the trained encoder sub-network of our model. We trained a classifier with feature vectors of the training dataset and test it with feature vectors of the test dataset. We have also made experiments in CIFAR-100 dataset however due to factors mentioned in section 6.1, we could not produce satisfactory results.



Epoch 10



Epoch 25



Epoch 50

FIGURE 6.2 OUTPUTS IMAGES OF BISTYLEGAN AFTER 10, 25, 50 EPOCHS

6.4 Classification with generated features

After feeding all the tasks to our model, we trained a shallow network to classify feature embeddings. We started with a simple model, the nearest mean classifier. The nearest mean classifier basically labels the data based on the distance to the mean of the class. Furthermore, for each class, we checked the performance of the classifier with a subset of feature embeddings selected by the distance to the class mean.

Experiments with the nearest mean classifier revealed that our feature embeddings are not centered around one point. Accuracy on the test dataset was not satisfactory. Therefore, we refer to a more sophisticated classifier, support vector machine for our classification task. With the support vector machine, we achieved considerably better results than the nearest mean classifier. We also fed the feature embeddings to the multilayer perceptron with one hidden layer, but the support vector machine classifier was superior. Figure 6.3 depicts the classification accuracy of feature vectors generated by the ResNet-50 model. As it is seen from the figure, the support vector machine classifier plainly overpowers other classifiers. The nearest-mean classifier was the worst-performing classifier because generated feature embeddings are not centered around the class mean. Multilayer perceptron never managed to surpass support vector machine in any hyperparameter settings.

Acquired results show that the optimal size for the noise vector is 200 which we have found independently for our specific network configurations. Interestingly, training 50% of feature embeddings with 200x1 vector size produces higher accuracy than training all feature embeddings with 100x1 vector size. Furthermore, with a 512x1 noise vector size, all the classifiers achieve higher accuracy when 50% of the feature embeddings are used for the training.

Figure 6.4 depicts the performance of two encoder architectures deployed for our training process. Accuracy results are obtained after feeding generated features to the SVM classifier. In low feature dimension, ResNet-50 surpasses ResNet-101 model however with 512x1 noise vector, ResNet-101 clearly brings better performance.

Since CIFAR-10 only has 10 classes, it is not quite useful for the incremental learning task. However, since we use GAN for representation learning which requires a huge number of samples for each class, we could not produce equal results with the CIFAR-100 dataset. Direct comparison with the iCaRL approach [**10**] would not be quite objective since it uses CIFAR-100 as the benchmark dataset. However, one fact should be highlighted that while iCaRL uses real data samples for the rehearsal, we propose to use low-dimensional feature embeddings.

We have also tested the ability of our model to generate samples in a conditional setting. As we highlighted before conditional generation allows us to produce an output from desired class, tackles the stochastic nature of GAN networks. We were able to generate samples for any given class and this property essentially alleviates the class imbalance problem which is a very common issue in real-world scenarios.

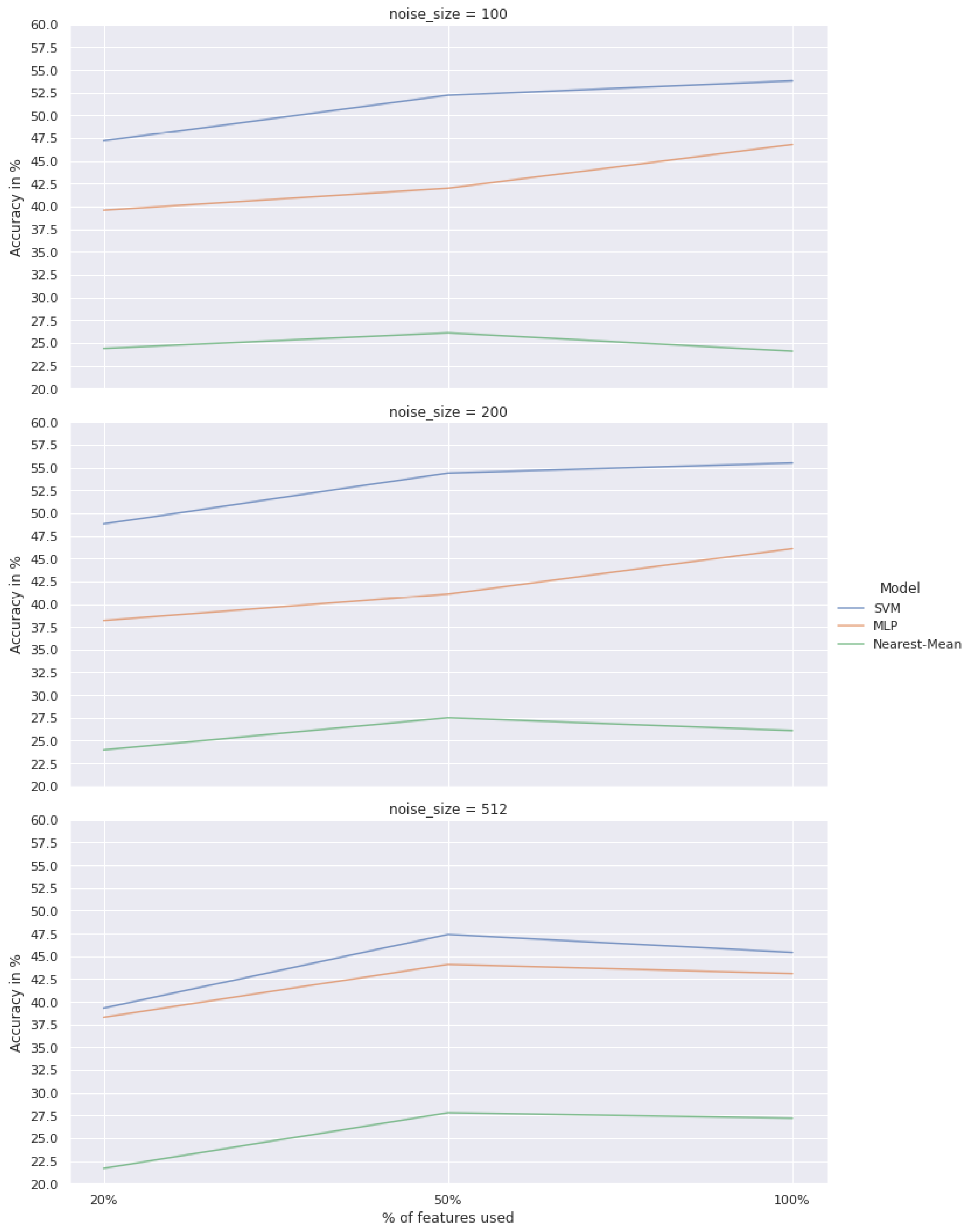


FIGURE 6.3 TRAINING FEATURE EMBEDDINGS IN VARIOUS SETTINGS

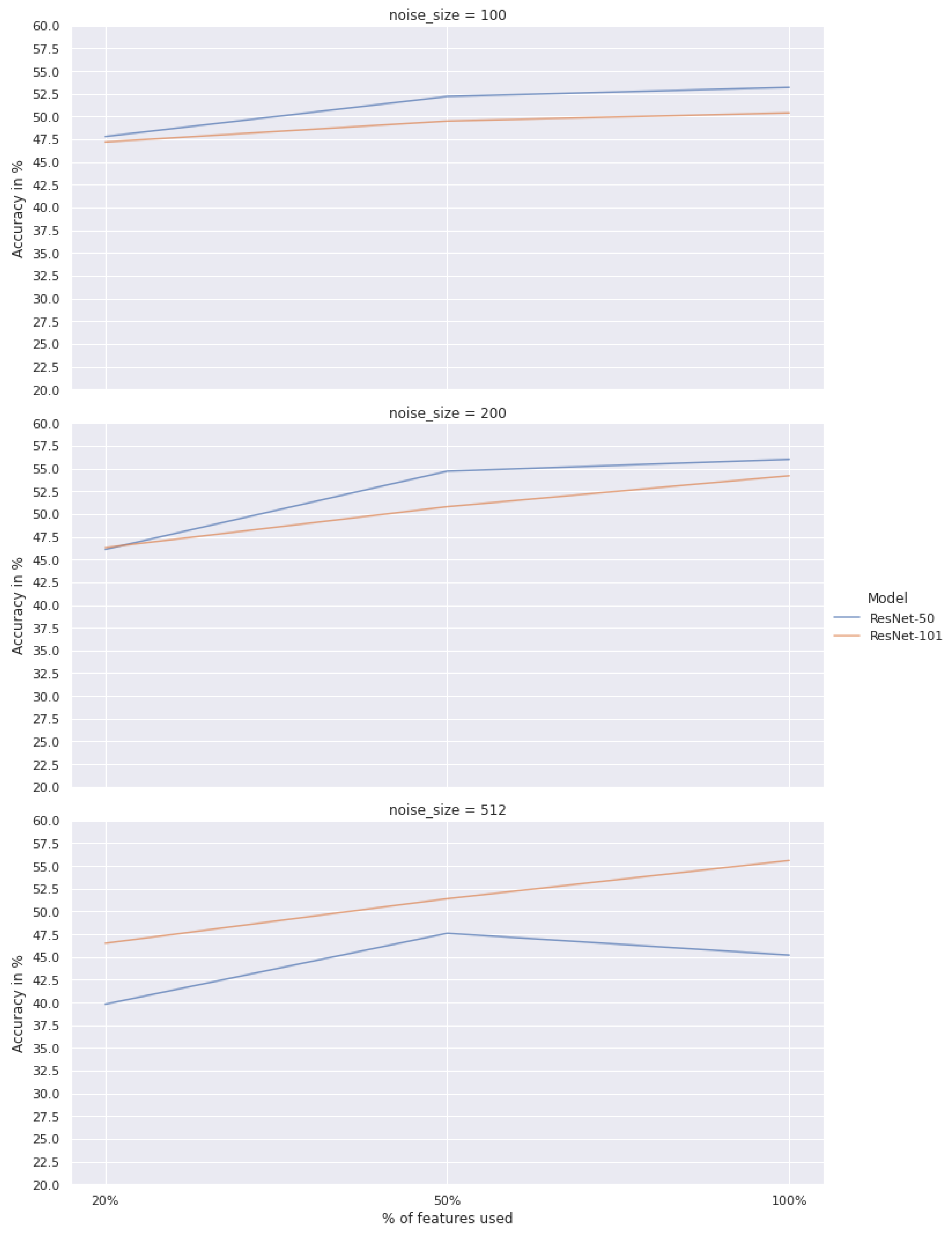


FIGURE 6.4 RESNET-50 AND RESNET-101 FEATURE EMBEDDINGS

6.5 Transfer learning

We have also tested the transfer learning capabilities of bidirectional GAN architecture. For that, we picked up the already trained encoder network of BigBiGAN on another dataset and used it to extract features from images the model has not seen before. Later we fed the features embeddings to the generator network of the same model to evaluate the reconstruction performance of the architecture (see Figure 5.11). We noticed that input samples after this process are not translated to pixel-accurate images and the model discards fine details on those samples. Nevertheless, generated images tend to capture higher-level semantic information about the input samples, preserve coarser details such as color, general shape. As shown Figure 6.5, we preserve semantic information in input images (except in the last image). We believe that if the proposed architecture is trained on a higher scale, with a huge dataset, the generated encoder model can effectively learn representative feature embeddings for input images.

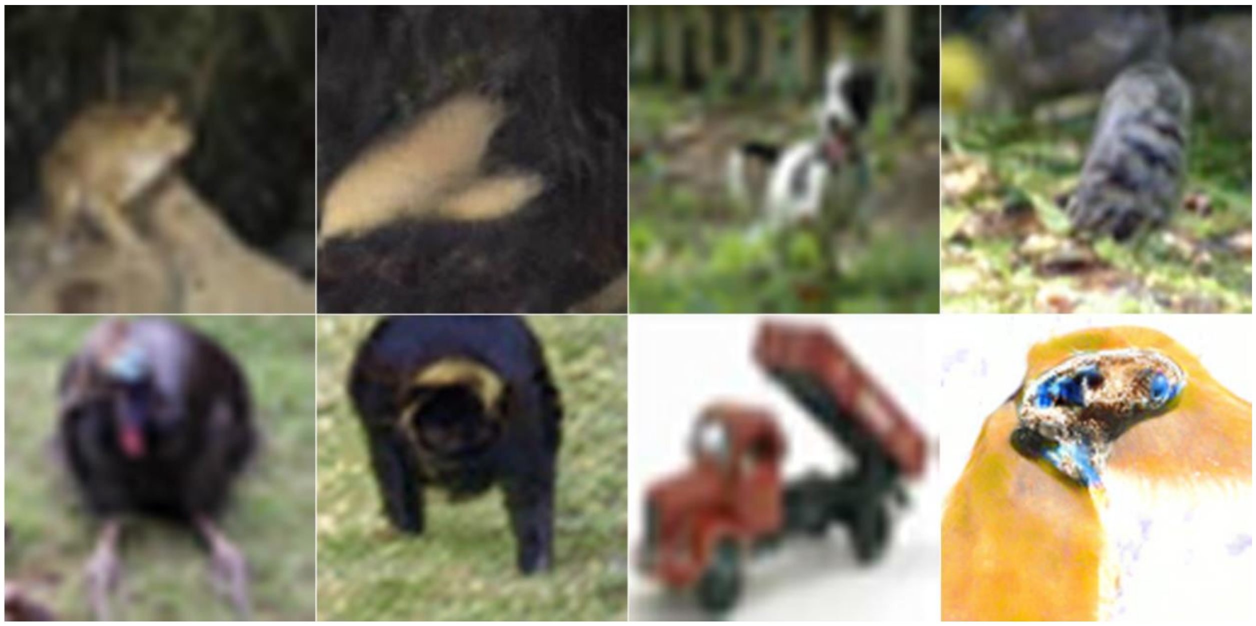


FIGURE 6.5 FEEDING CIFAR-10 IMAGES TO THE PRE-TRAINED BIGBIGAN NETWORK.

7. Conclusion

The objective of this thesis was to evaluate GAN architecture in an incremental learning scenario. We designed a GAN model by combining two successful, fundamentally different GAN applications. We produced low-dimensional inputs that are not subject to privacy concerns, class imbalance problem, and memory footprint issues by picking the most efficient incremental learning strategy. The proposed model is especially suitable for datasets suffering from the lack of image samples. Our model has the ability to generate the desired number of images and feature embeddings for classes that are low in samples. This is the main advantage of our approach over classic feature extractor models which are limited to operate only on the available data. Furthermore, the property of generating image samples during the training process facilitates the evaluation of generated feature embeddings.

Despite computational limitations, we did extensive hyperparameter optimization, tried various network configurations to find the best-suited set of parameters for our specific use case. We followed the general patterns proposed in [19] and [21] to reach the optimal performance quickly; however, we also overcame inconsistencies that arose when merging these two different architectures.

The experiments with our proposed model revealed that our approach can learn semantic information about the input images and even can produce good output images despite learning two tasks simultaneously. However, it does not explicitly learn low-level separable features of the classes as effectively as classical features extractors. The mentioned trait is natural to generative adversarial networks since, by its nature, GAN models belong to generative family of models. They do not learn discriminant function among input samples belonging to the different classes which is the case for discriminative models. We had the idea of adding additional classification terms into the loss equation to promote the learning of distinctive feature embeddings, however having already separate loss terms for representation learning and image generation, we would end up failing to learn any task if we impose an additional task to our model.

We also evaluate the possibility of transferring knowledge from pre-trained bidirectional GAN models. Experiments with BigBiGAN architecture revealed the possibility of training the proposed architecture on a higher scale and learning low-dimensional feature vectors for any other dataset that the model has not seen before.

8. References

[1] I. Goodfellow et al., "Generative adversarial networks", Communications of the ACM, vol. 63, no. 11, pp. 139-144, 2020. Available: 10.1145/3422622.

[2] “Deep Blue versus Garry Kasparov,” Wikipedia, 11-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov. [Accessed: 11-Oct-2021].

[3] B. Liu, "Learning on the Job: Online Lifelong and Continual Learning", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 09, pp. 13544-13549, 2020. Available: 10.1609/aaai.v34i09.7079.

[4] A. Brock, J. Donahue, en K. Simonyan, “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, CoRR, vol abs/1809.11096, 2018.

[5] M. Biesialska, K. Biesialska, en M. R. Costa-jussà, “Continual Lifelong Learning in Natural Language Processing: A Survey”, CoRR, vol abs/2012.09823, 2020.

[6] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks", Proceedings of the National Academy of Sciences, vol. 114, no. 13, pp. 3521-3526, 2017. Available: 10.1073/pnas.1611835114.

[7] Z. Li en D. Hoiem, “Learning without Forgetting”, CoRR, vol abs/1606.09282, 2016.

[8] A. Chaudhry, P. K. Dokania, T. Ajanthan, en P. H. S. Torr, “Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence”, CoRR, vol abs/1801.10112, 2018.

[9] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, en J. van de Weijer, “Class-incremental learning: survey and performance evaluation”, CoRR, vol abs/2010.15277, 2020.

[10] S.-A. Rebuffi, A. Kolesnikov, en C. H. Lampert, “iCaRL: Incremental Classifier and Representation Learning”, CoRR, vol abs/1611.07725, 2016.

[11] R. Aljundi et al., “Online Continual Learning with Maximally Interfered Retrieval”, CoRR, vol abs/1908.04742, 2019.

[12] Y. Xian, T. Lorenz, B. Schiele, en Z. Akata, “Feature Generating Networks for Zero-Shot Learning”, CoRR, vol abs/1712.00981, 2017.

[13] A. Makhzani, J. Shlens, N. Jaitly, en I. J. Goodfellow, “Adversarial Autoencoders”, CoRR, vol abs/1511.05644, 2015.

[14] X. Mao, Z. Su, P. S. Tan, J. K. Chow, en Y.-H. Wang, “Is Discriminator a Good Feature Extractor?”, CoRR, vol abs/1912.00789, 2019.

[15] A. Iscen, J. Zhang, S. Lazebnik, en C. Schmid, “Memory-Efficient Incremental Learning Through Feature Adaptation”, CoRR, vol abs/2004.00713, 2020.

[16] J. Donahue, P. Krähenbühl, en T. Darrell, “Adversarial Feature Learning”, CoRR, vol abs/1605.09782, 2016.

[17] T. Karras, S. Laine, en T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks”, CoRR, vol abs/1812.04948, 2018.

[18] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, en T. Aila, “Analyzing and Improving the Image Quality of StyleGAN”, CoRR, vol abs/1912.04958, 2019.

[19] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, en T. Aila, “Training Generative Adversarial Networks with Limited Data”, CoRR, vol abs/2006.06676, 2020.

[20] M. Mirza en S. Osindero, “Conditional Generative Adversarial Nets”, CoRR, vol abs/1411.1784, 2014.

[21] J. Donahue en K. Simonyan, “Large Scale Adversarial Representation Learning”, CoRR, vol abs/1907.02544, 2019.