The Journal of Supercomputing (2021) 77:13358–13384
https://doi.org/10.1007/s11227-021-03808-2

Scalable balanced training of conditional generative adversarial neural networks on image data

Massimiliano Lupo Pasini[1] · Vittorio Gabbi[2] · Junqi Yin[3] · Simona Perotto[4] · Nouamane Laanait[5]

Accepted: 9 April 2021 / Published online: 26 April 2021

Abstract

We propose a distributed approach to train deep convolutional generative adversarial neural network (DC-CGANs) models. Our method reduces the imbalance between generator and discriminator by partitioning the training data according to data labels, and enhances scalability by performing a parallel training where multiple generators are concurrently trained, each one of them focusing on a single data label. Performance is assessed in terms of inception score, Fréchet inception distance, and image quality on MNIST, CIFAR10, CIFAR100, and ImageNet1k datasets, showing a significant improvement in comparison to state-of-the-art techniques to training DC-CGANs. Weak scaling is attained on all the four datasets using up to 1000 processes and 2000 NVIDIA V100 GPUs on the OLCF supercomputer Summit.

Keywords Generative adversarial neural networks · Deep learning · Supercomputing · Computer vision

✉ Massimiliano Lupo Pasini
lupopasinim@ornl.gov

[1] Oak Ridge National Laboratory, 1 Bethel Valley Road, Computational Sciences and Engineering Division, Bldg. 5700, Rm F119, Mail Stop 6085, P.O. Box 2008, Oak Ridge, TN 37831, USA

[2] Department of Automation and Control Engineering, Milan, MI 20133, Italy

[3] Oak Ridge National Laboratory, 1 Bethel Valley Road, National Center for Computational Sciences, Oak Ridge, TN 37831, USA

[4] Department of Mathematics, Milan, MI 20133, Italy

[5] Anthem, Inc., Atlanta, GA 30326, USA

1 Introduction

Generative adversarial neural networks (GANs) [1–4] are deep learning (DL) models whereby a dataset is used by an agent, called the generator, to sample white noise from a latent space and simulate a data distribution to create new (fake) data that resemble the original data it has been trained on. Another agent, called the discriminator, has to correctly discern between the original data (provided by the external environment for training) and the fake data (produced by the generator). The generator prevails over the discriminator if the latter does not succeed in distinguishing anymore the original from the fake. The discriminator prevails over the generator if the fake data created by the generator is categorized as fake, and the original data is still categorized as original. An illustration that describes a GANs model is shown in Fig. 1. Originally, GANs have been used on image data to improve the generalizability of DL models for object recognition. In particular, the goal was to use GANs for data augmentation to generate new data (similar to the original data), and use the augmented dataset to improve the accuracy of the object classifier. Since GANs was originally introduced, the scope of applications that rely on GANs to overcome computational limitations has broadened. For instance, recent applications of GANs are related to video synthesis [5] to improve the resolution of videos generated through videocameras, and face image alteration [6] to help national security agencies identify fugitive criminals that undergo mild facial surgeries.

The training of GANs is driven by the values of the cost functions associated with the agents [7]. The cost functions used to evaluate the performance of the discriminator and the generator are related to the number of false positives (original images identified by the discriminator as fake) and false negatives (fake images identified by the discriminator as original). The task of the discriminator is relatively simple in that it only has to assign a Boolean value to an image, according to whether the image is predicted as original or fake. On the contrary, the generator needs to map white noise sampled from the latent space into newly

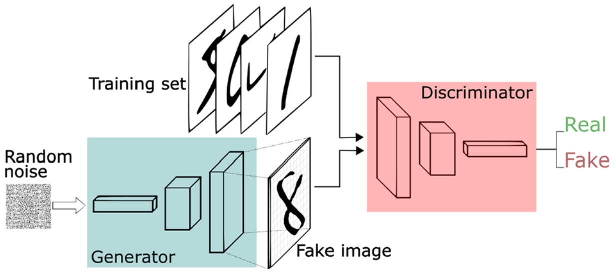


Fig. 1 The GAN framework pits two adversaries against each other in a game. Each player is represented by a differentiable function controlled by a set of parameters. Typically these functions are implemented as deep neural networks. Training examples are randomly sampled from the training set and used as input for the first player, the discriminator. The goal of the discriminator is to output the probability that its input is real rather than fake, under the assumption that half of the inputs it is ever shown are real and half are fake. Image from https://sthalles.github.io/intro-to-gans/

created images, and the images created must reproduce relevant features that belong to each data category represented in the training data.

This imbalance between the difficulty of the computational tasks of discriminator and generator is natural in GANs and cost functions currently used to measure the performance of the generator do not retain information about the disparity in computational tasks between discriminator and generator. As a result, the precision attained by the discriminator in performing its tasks (saying if an image is fake or original) is always higher than the precision with which the generator performs its own (create a whole set of fake images from white noise). Recent game theoretic results show that the unbalanced training of GANs can cause the generator to cycle [8] or converge to a (potentially bad) local optimum [9], which causes the generator to get stuck reproducing only one specific data point (this phenomenon is known in the DL literature as mode collapse). It is thus important to balance the training of GANs models in order to improve the performance of the generator and obtain fake images with similar features to the ones contained in the original data, but this task is challenging [10–14].

Some recent approaches have tackled the imbalance of the two agents by changing the numerical optimization used to train the GANs model. [15] Other recent approaches have tackled the challenge of imbalance between discriminator and generator by improving the complexity of the GANs model [16–24]. However, datasets with a large number of categories still pose a non-trivial challenge that prevents the generator from attaining a good performance in creating new fake images still due to a large variability between classes.

In addition, all the existing approaches to train GANs are characterized by a limited parallelizability, in that existing parallel techniques for GANs are based on data parallelization that distributes large-scale data to multiple replicas of the same model via ensemble learning, and do not further enhance the scalability of GANs by attempting any model parallelization [11]. Therefore, state-of-the-art GANs approaches do not fully leverage high-performance computing (HPC) facilities to attain a better performance.

We propose a novel distributed approach to train CGANs through a nonzero-sum game formulation that uses data categories to address the performance disparity between discriminator and generator and improve the scalability of the CGANs training via model parallelization. We use the labels to split the data and process each class independently, using a generator for each class. Our distributed approach relies on a factorization of the data distribution where each factor is associated with a single data category. The factorization of the probability distribution makes our approach differ from standard CGANs, where the joint data probability is never decomposed into simpler factors and a single generator is still assigned with the task of creating new images that span all the data categories. The data splitting performed according to the labels removes the variability between classes and thus corrects the imbalance of standard GANs training. Because of the independence of each generator from the others, the generators can be trained concurrently and this enhances scalability.

2 Related work

Some approaches presented in the literature address the imbalance of the two agents by changing the numerical optimization used to train the GANs model. An example is the Competitive Gradient Descent method (CGD) [15], which recasts the GANs training as a zero-sum game, whereby the discriminator and the generator compete against each other, and the goal is to identify an equilibrium between the agents. However, the zero-sum formulation does not reflect well the interaction between generator and discriminator during the GANs training, since the loss of one agent does not directly translate into the gain of the other agent, as it is indeed assumed in a zero-sum game. Other recent approaches have tackled the challenge of imbalance between discriminator and generator by improving the complexity of the GANs model [16–24]. Among these approaches, Conditional GANs (CGANs) [25–30] proceed by expanding the latent space used as input for the generator by adding information about the data categories. The role of CGAN models is to reconstruct a joint data distribution defined on the expanded latent space that combines the image data with the corresponding labels. The inclusion of data labels as an additional latent space variable facilitates the generator in discerning relevant features that make an image more likely to belong to a data category than to another. However, datasets with a large number of categories still pose a non-trivial challenge that prevents the generator from attaining a good performance in creating new fake images still due to a large variability between classes.

Other approaches presented in the literature aim at improving the scalability of the training, such as ensemble learning [11]. Ensemble learning methods combine several machine learning models into one predictive model to decrease variance, bias, or improve predictions. In the context of GANs, one main advantage provided by ensemble learning is data parallelization, which accelerates the processing of large data by distributing it across different model replicas. Different model replicas exchange the portions of data in a round-robin fashion throughout consecutive iterations to ensure that the entire dataset is visited by each model replica. Moreover, the updates of the model parameters computed locally are exchanged between the model replicas at a tunable frequency to guarantee consistency between different replicas of the model.

3 Background on conditional generative adversarial neural networks

We first define the following input and output spaces, each with an associated probability distribution:

- Z is a noise space used to seed the generative model. $Z = \mathbb{R}^{d_Z}$, where d_Z is a hyperparameter. Values $\mathbf{z} \in Z$ are sampled from a noise distribution $p_{\mathbf{z}}(\mathbf{z})$. In our experiments $p_{\mathbf{z}}$ is a white noise model.

- Y is an embedding space used to condition the generative model on additional external information, drawn from the training data. $Y = \mathbb{R}^{d_Y}$, where d_Y is a hyper-parameter. Using conditional information provided in the training data, we define a density model $p_{\mathbf{y}}(\mathbf{y})$.
- X is the data space which represents an image output from the generator or input to the discriminator. Values are normalized pixel values: $X = [0,1]^W \times C$, where W represents the resolution of the input images, and C is the set of distinct color channels in the input images. Using the images in the training data and their associated conditional data, we can define a density model $p_{\text{data}}(\mathbf{x})$ of face images. This is exactly the density model we wish to replicate with the overall model in this paper.

We now define two functions:

- $G : Z \times Y \rightarrow X$ is the conditional generative model (or generator), which accepts noise data $\mathbf{z} \in Z$ and produces an image $\mathbf{x} \in X$ conditional to the external information $\mathbf{y} \in Y$.
- $D : X \rightarrow [0,1]$ is the discriminative model (or discriminator), which accepts an image $\mathbf{x}$ and condition $\mathbf{y}$ and predicts the probability under condition $\mathbf{y}$ that $\mathbf{x}$ came from the empirical data distribution rather than from the generative model.

The goal of CGANs is to provide a model that estimates the probability distribution $p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y})$, parameterized by parameters $\boldsymbol{\theta}$ that describes the DL model. We then refer to the likelihood as the probability that the model assigns to the training data: $\Pi_{i=1}^{m} p_{\text{model}}(\mathbf{x}_i, \boldsymbol{\theta}, \mathbf{y})$, for a dataset containing m training samples $\mathbf{x}_i$. Among the different types of generative models, GANs is a type of model that works via the principle of *maximum likelihood*. The principle of maximum likelihood aims at choosing the parameters $\boldsymbol{\theta}$ for the DL model that maximize the likelihood of the training data

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}(p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y})) \tag{1}$$

Using condition information provided in the training data, we define a density model $p_{\mathbf{y}}(\mathbf{y})$. CGANs use the Bayes theorem to combine the conditional probability $p_{\text{model}}(\mathbf{x}|\mathbf{y})$ and the density model $p_{\mathbf{y}}(\mathbf{y})$ to yield the joint model probability $p_{\text{model}}(\mathbf{x}, \mathbf{y})$:

$$p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}) = p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y}) p_{\mathbf{y}}(\mathbf{y}). \tag{2}$$

While (2) cannot be expressed in closed analytical form, CGANs can be trained without needing to explicitly define a density function, because this type of generative model offers a way to train the model while interacting only indirectly with $p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y})$, usually by sampling from it.

The two players in the game are represented by two functions. The discriminator is defined by a function D that takes $\mathbf{x}$ as input and uses $\boldsymbol{\theta}^{(D)}$ as parameters. The generator is defined by a function G that takes $\mathbf{z}$ as input and uses $\boldsymbol{\theta}^{(G)}$ as parameters. Both players have cost functions that are defined in terms of both players' parameters. A consensus has been reached in the literature about which cost functions fully describe the performance of the discriminator due to the simplicity of the discriminator's task [1, 31], whereas the complexity of the computational task of the generator still keeps different options open as to the cost function that better describes the generator's performance. In the zero-sum game formulation, the discriminator minimizes a cross-entropy and the generator maximizes the same cross-entropy. The generator's gradient tends to vanish when the discriminator successfully rejects the generator's samples with high confidence, but vanishing gradients reduce the effectiveness of the updates computed during the training. To avoid vanishing gradients, an approach widely used in the literature is to transform the GANs training into a nonzero-sum game. In the context of nonzero-sum games, the generator maximizes the log-probability of the discriminator being mistaken instead of having the generator minimize the log-probability of the discriminator being correct. Following the nonzero sum game formulation, the cost used for the discriminator is

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))) \tag{3}$$

The cost function we choose for the generator is

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z})). \tag{4}$$

This cost function $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ quantitatively describes the ability of the generator in tricking the discriminator so that the discriminator confuses fake images as real. This choice of $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ better reflects the goal of the generator as an individual agent, with respect to other alternatives that force a strong dependence of $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ on $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$, such as, for instance, in the zero-sum game formulation where $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$. The definition in (4) ignores the false positive, because the original images are not a product of the generator (only fake images are).

The discriminator wishes to minimize $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ and must do so while controlling only $\boldsymbol{\theta}^{(D)}$. The generator wishes to maximize $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ and must do so while controlling only $\boldsymbol{\theta}^{(G)}$. The solution of this mini-max game is a Nash equilibrium. Here, we use the terminology of local differential Nash equilibria [32]. In this context, a Nash equilibrium is a tuple $(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ that is a local minimum of $J^{(D)}$ with respect to $\boldsymbol{\theta}^{(D)}$ and a local maximum of $J^{(G)}$ with respect to $\boldsymbol{\theta}^{(G)}$.

The training process consists of a numerical optimization scheme that iteratively updates $\boldsymbol{\theta}^{(D)}$ and $\boldsymbol{\theta}^{(G)}$ [11]. On each step, two minibatches are sampled: a minibatch of $\mathbf{x}$ values from the dataset and a minibatch of $\mathbf{z}$ values drawn from the model's prior over latent variables. The standard choice of a numerical optimization algorithm used to update $\boldsymbol{\theta}^{(D)}$ and $\boldsymbol{\theta}^{(G)}$ is a gradient-based optimization algorithm called Adam [33].

4 Distributed conditional generative adversarial neural networks

Our novel approach aims at implementing a distributed version of CGANs as a nonzero-sum game. Figure 2 describes how our approach relates to the other generative models presented in the literature. Our distributed approach to train CGANs relies on the equality

$$p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta}) = \sum_{k=1}^{K} p_{\text{model}}(\mathbf{x}, \boldsymbol{\theta} | \mathbf{y}_k) p_{\mathbf{y}}(\mathbf{y}_k) \tag{5}$$

to distribute the computation of each term $p_{\text{model}}(\mathbf{x}, \mathbf{y}_k)$ by training K distributed CGANs, each one per class, and then we combine the results at the end of each training to yield $p_{\text{model}}(\mathbf{x})$. The numerical examples presented in this paper are characterized by a one-to-one mapping between $\mathbf{y}_k$ and the labels in the image dataset. The splitting of the total probability performed in (5) is possible only under the assumption that the input data contains labels, which is used to perform the splitting. The advantage of our approach consists in the fact that all the K distributed CGANs can be trained concurrently and independently of each other, thus exposing the model to a higher level of parallelism. If the complexity representation of the objects in each category is comparable, the training time for each distributed CGANs model is approximately the same, which in turn translates into promising performance in terms of weak scalability (the time-to-solution is constant for an increasing number of processors used to solve problems of increasing size so that the computational workload per processor is unchanged). An illustration that describes the distribution of CGANs is provided in Fig. 3.

The parameters $\boldsymbol{\theta}$ for each distributed CGANs are independent. Therefore, they are updated independently using Adam on each separate generator–discriminator pair. When the trained model is deployed for production, a random number generator provides the white noise and the label of the object whose image has to be generated. The randomly selected label determines which GANs pair to call, and

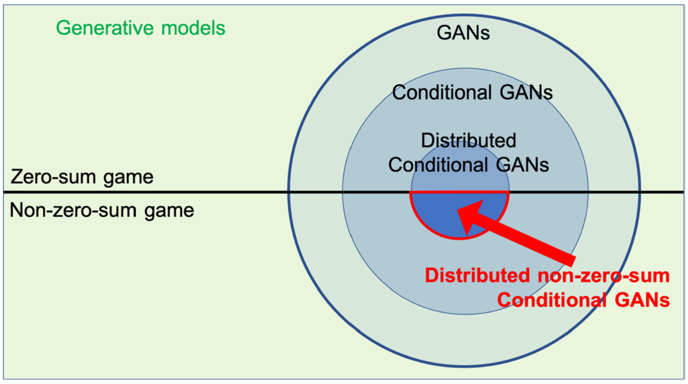


Fig. 2 Scheme representing the relationship between the generative models currently developed in the deep learning community

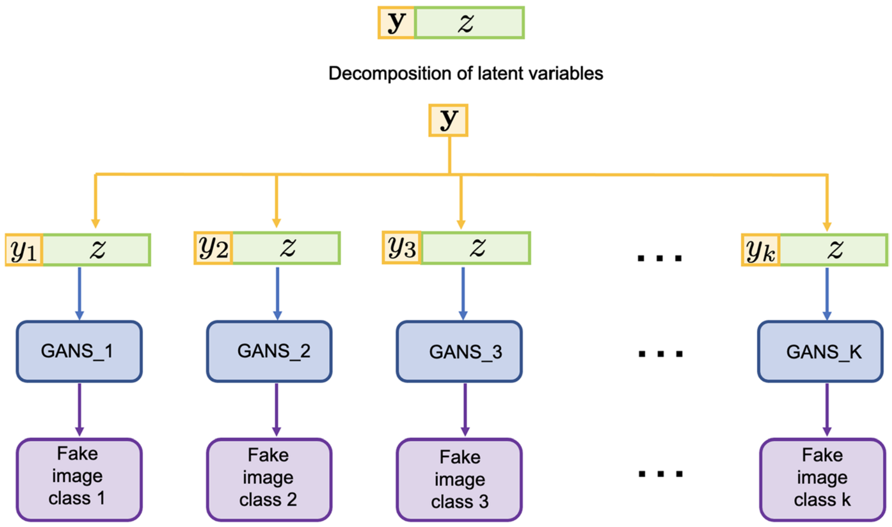


Fig. 3 Illustration of distributed CGANs

the white noise is passed to the selected GANs pair to generate a new fake image for the specific object category associated with the label. The fact that our approach never enforces an exchange of updates across different agent pairs makes it differ from ensemble learning, where the different model replicas continuously exchange local updates of $\boldsymbol{\theta}^{(D)}$ and $\boldsymbol{\theta}^{(G)}$ between each other to guarantee global consistency of the parameters. Ensemble learning mainly resorts to parallelization as a means to stabilize the training of GANs, and to accelerate the processing of large datasets. However, this stable training and faster data processing do not necessarily result in faster convergence. In fact, ensemble learning requires each model replica to span the entire dataset in a round-robin fashion, and this means that each model replica is required to reconstruct the data distribution associated with the entire dataset, meaning that the difficulty of the modeling task has not been addressed. Our distributed approach confines each model replica to be trained on data associated with a single label, thereby resulting into an accelerated training, because the number of data batches processed by each GANs pair is significantly reduced. The partition of the data according to the classes facilitates our approach to scale, as confirmed by the weak scaling tests presented at the end of numerical section.

Our approach can be changed so that one generator handles multiple classes altogether, and create fake images associated with multiple data classes. This variation of our approach would allow the code to run on small clusters (where there are not enough resources to instantiate several generators that can work independently). However, this would defeat the statistical motivation behind the way we split the data. In fact, we remind the reader that our distributed approach aims at reducing the variability in the portion of data that is handled by each generator separately. In situations like the ones described in this paper, the variability between classes is much larger than the variability between data of the same class. When multiple data classes are simultaneously handled by the same generator, the variability of the data portion still remains large, and this hinders the generator from thoroughly exploring the data space. Because of the lack of benefit in grouping multiple classes under the same generator, our distributed approach (with one generator per data class) is designed to work on large scale computers, because it inherently assumes that the number of processors available must be at least equal to the total number of data classes.

In terms of neural network architectures to model the agents, our distributed approach aims at leveraging the parallelization of the training to enhance the predictive performance of simple (and thus faster to train) neural networks as opposed to state-of-the-art GANs that focus on building more complex (and thus more expensive to train) neural networks. To this goal, the numerical results presented in Sect. 5 focus on using our distributed training to improve the performance of DC-CGANs, a relatively small and simple neural network architecture with respect to larger and more complex ones that have been recently proposed to improve the accuracy [19, 20, 22, 34].

5 Numerical results

In this section, we present numerical tests where we compare the performance of standard deep convolutional GANs [2] (DC-GANs) with deep convolutional conditional GANs (DC-CGANs) [35] and our distributed approach to train DC-CGANs on image data. Image data are represented as pixels in a Cartesian structure. For each pixel, a set of values called channels are assigned to describe the local graphic properties. The channels per pixel are only one for black and white images, and colored images have three channels (red, green and blue). In general, the graphic variability between classes is more pronounced than the graphic variability within a specific class, because objects of the same type generally resemble more than objects of different nature. The benchmark datasets we consider are characterized by labels that clearly separate images according to their category, and the category is related to the type of object represented in the image.

The comparison between standard DC-GANs, standard DC-CGANs and our novel method for distributed DC-CGANs is performed on a quantitative level by measuring the Inception Score (IS) [36] and the Fréchet Inception Distance (FID) [37]. The IS takes a list of images and returns a single floating point number, the score. The score is a measure of how realistic a GAN's output is. IS is an automatic alternative to having humans grade the quality of images. The score measures two things simultaneously: the image variety (e.g., each image is a different breed of dog), and whether each image distinctly looks like a real object. If both things are true, the score will be high. If either or both are false, the score will be low. A higher score describes better performance for GANs, as it means that the GAN model can generate many different distinct and realistic images. The lowest score possible is zero. Mathematically the highest possible score is infinity, although in practice there will probably emerge a finite ceiling. FID is another metric used to assess the quality of images created by the generator of a generative adversarial network (GAN).

Unlike IS, which evaluates only the distribution of generated images, the FID compares the distribution of generated images with the distribution of real images that were used to train the generator. Lower values of FID correspond to the distribution of generated images approaching the distribution of real images, and this is interpreted as an improvement of the generator in creating more realistic images .

In DC-GANs, the input of the generator has size 100 (size of the white noise) and the output of the generator has one channel for black–white images and three channels for colored images. The specifics of the architectures for generator and discriminator used to build DC-GANs models are described in Tables 1 and 2. DC-CGANs differ from DC-GANs because they use additional information about the labels to improve the training of the generative model. In DC-CGANs, the input of the generator has size $100 + K$ (100 is the size of the white noise and K is the number of data classes) and the output of the generator has two channels (one channel for the color and one for label) for black–white images and four channels (three channel for the color and one for the label) for colored images. The specifics of the architectures for generator and discriminator used to build DC-CGANs models are described in Tables 3 and 4. The architecture of generator and discriminator for our approach that implements distributed DC-CGANs are the same as for DC-GANs, because each generator–discriminator pair focuses only on one data class, so the conditional information about the data label is inherently retained in the selection of the data portion used for training. The training is performed using the optimizer Adam and a learning rate of 2e − 4, and a total number of 1,000 epochs for all the types of GANs we consider on each dataset.

In order to support our claim that the variability of the feature between image classes is much larger than the variability of the features between images of the same class, we run the Analysis of Variance (ANOVA) test [38] on each dataset. ANOVA is a procedure to measure the "variation" among and between groups, and

Table 1 Architecture of the generator in DC-GANs

Generator						
Layer	Input dim	Output dim	Activation	Kernel size	Stride	Padding
Input	100	8192	ReLU	/	/	/
Resizing						
Batch normalization (epsilon = 1e-5, momentum = 0.1)						
Upsample (scalefactor = 2)						
Convolution1	128	128	/	3	1	1
Batch normalization (epsilon = 0.8, momentum = 0.1)						
leakyReLU (slope = 0.2, inplace=True)						
Upsample (scalefactor = 2)						
Convolution2	128	64	/	3	1	1
Batch normalization (epsilon = 0.8, momentum = 0.1)						
leakyReLU(slope = 0.2, inplace=True)						
Convolution3	64	1 or 3	/	3	1	1
Tanh						

Table 2 Architecture of the discriminator in DC-GANs

Discriminator						
Layer	Input dim	Output dim	Activation	Kernel size	Stride	Padding
Convolution1	1 or 3	16	/	3	2	1
leakyReLU(slope = 0.2, inplace=True)						
Dropout (0.25)						
Convolution2	16	32	/	3	2	1
leakyReLU(slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Convolution3	32	64	/	3	2	1
leakyReLU(slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Convolution4	64	128	/	3	2	1
leakyReLU(slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Output	2048	1	Sigmoid	/	/	/

Table 3 Architecture of the generator in DC-CGANs. The letter *K* indicates the number of classes in the dataset

Generator						
Layer	Input dim	Output dim	Activation	Kernel size	Stride	Padding
Input	100 + K	8192	ReLU	/	/	/
Resizing						
Batch normalization (epsilon = 1e-5, momentum = 0.1)						
Upsample (scalefactor = 2)						
Convolution1	128	128	/	3	1	1
Batch normalization (epsilon = 0.8, momentum = 0.1)						
leakyReLU (slope = 0.2, inplace=True)						
Upsample (scalefactor = 2)						
Convolution2	128	64	/	3	1	1
Batch normalization (epsilon = 0.8, momentum = 0.1)						
leakyReLU (slope = 0.2, inplace=True)						
Convolution3	64	1 or 3	/	3	1	1
Tanh						

it is used to analyze the differences among means. ANOVA is based on the law of total variance, where the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether two or more population means are

Table 4 Architecture of the discriminator in DC-CGANs

Discriminator						
Layer	Input dim	Output dim	Activation	Kernel size	Stride	Padding
Convolution1	2 or 4	16	/	3	2	1
leakyReLU (slope = 0.2, inplace=True)						
Dropout (0.25)						
Convolution2	16	32	/	3	2	1
leakyReLU (slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Convolution3	32	64	/	3	2	1
leakyReLU (slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Convolution4	64	128	/	3	2	1
leakyReLU (slope = 0.2, inplace=True)						
Dropout (0.25)						
Batch normalization (epsilon = 0.8, momentum = 0.1)						
Output	2048	1	Sigmoid	/	/	/

equal. The null hypothesis of the ANOVA test assumes that the means of all the groups of the population are equal, whereas the alternative hypothesis assumes that at least one group of the population has mean different from all the others.

The test statistic used in the hypothesis test is the F-statistic, which is distributed as a Fisher distribution. High values of the F-statistic correspond to small p values (i.e., p values lower than 0.05) of the ANOVA test, and this results into statistical evidence to reject the null hypothesis and accept the alternative hypothesis, meaning that the variability between groups is much larger than the variability within groups. Statistical evidence to reject the null hypothesis is used as supporting argument that different classes can be treated separately.

5.1 Hardware description

The numerical experiments are performed using Summit [39], a supercomputer at the Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory. Summit has a hybrid architecture, and each node contains two IBM POWER9 CPUs and six NVIDIA Volta GPUs all connected together with NVIDIA's high-speed NVLink. Each node has over half a terabyte of coherent memory (high bandwidth memory + DDR4) addressable by all CPUs and GPUs plus 1.6 TB of non-volatile memory (NVMe) storage that can be used as a burst buffer or as extended memory. To provide a high rate of communication and I/O throughput, the nodes are connected in a non-blocking fat-tree using a dual-rail Mellanox EDR InfiniBand interconnect.

Table 5 Wall-clock computational time measured in seconds for the training of DC-GANs, DC-CGANs, and Distributed DC-CGANs on MNIST

	Wall-clock Time (seconds)
DC-GANs	89,020
DC-CGANs	92,020
Distributed DC-CGANs	10,011

Table 6 Inception score associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on MNIST dataset. The inception score is calculated using 10 splits to collect statistics

	Mean	Standard deviation
DC-GANs	2.54	0.54
DC-CGANs	2.62	0.03
Distributed DC-CGANs	2.71	0.04

5.2 Software description

The numerical experiments are performed using `Python3.7` with `PyTorch v1.3.1` package [40] for autodifferentiation to train the DL models with the use of GPUs, and the `mpi4py v3.0.2` tool is used for distributed computing.

As for the DC-GANs and the DC-CGANs approach, generator and discriminator are mapped to the same MPI processes. As for the distributed DC-CGANs, there are multiple discriminator–generator pairs, each one associated with a specific data class, and every discriminator–generator pair is mapped to an MPI process. Each MPI process instantiated in the distributed DC-CGANs is linked to two GPUs, one dedicated to training the discriminator and one dedicated to training the generator. Therefore, the total number of GPUs used with distributed DC-CGANs amounts to twice the number of MPI processes instantiated.

5.3 MNIST [41]

The ANOVA test run on the MNIST dataset produces a value for the *F*-statistic equal to 38.78, which leads to a p value close to zero. This indicates that there is strong statistical evidence to claim that the variability between images of different classes is larger than the variability between images of the same class. The wall-clock computational time is reported in Table 5, and it shows our distributed approach outperforming the standard DC-GANs and DC-CGANs by an order of magnitude. This is in line with what expected because our distributed approach processes the 10 portions of data that are associated with the 10 classes concurrently, whereas the standard DC-GANs and the standard DC-CGANs process the total dataset sequentially. The IS obtained with all the three GANs models is shown in Table 6 and the FID score is shown in Table 7. The results show a similar performance for DC-GANs, DC-CGANs and distributed DC-CGANs, with a slight improvement using the latter over the other two. Samples of fake images generated by the generator of DC-GANs,

Table 7 Fréchet Inception Distance associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on MNIST dataset

	FID
DC-GANs	75.11
DC-CGANs	69.72
Distributed DC-CGANs	65.12

DC-CGANs and distributed DC-CGANs trained on MNIST are shown in Tables 8, 9 and 10, respectively. The performance of DC-CGANs is very similar to the one of DC-GANs, because some of the digits are still written in a wobbly manner, preventing a clear understanding of what is the actual digit represented. This happens

Table 8 Fake images generated by DC-GANs trained on MNIST dataset

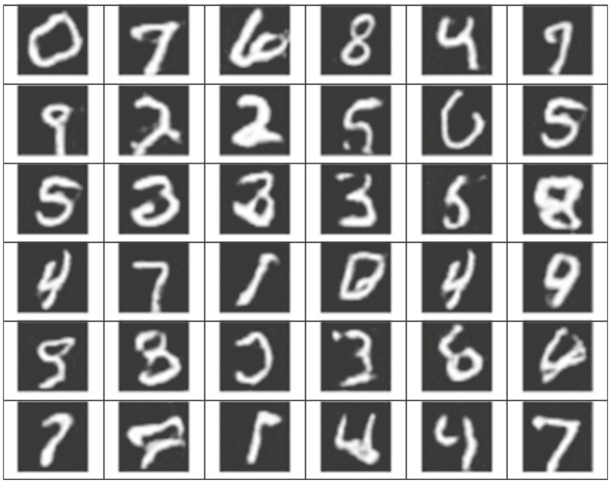


Table 9 Fake images generated by DC-CGANs trained on MNIST dataset

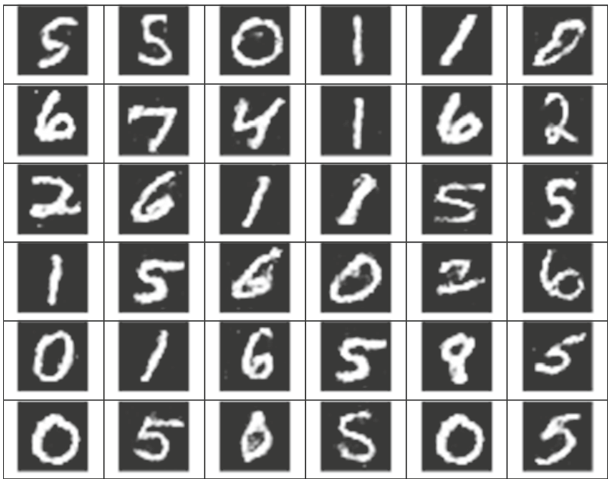
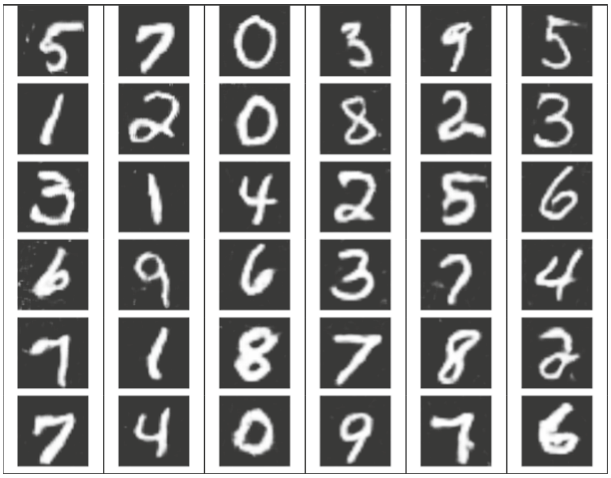


Table 10 Fake images generated by distributed DC-CGANs trained on MNIST dataset



in situations where multiple digits resemble very much (the shape of a 3 is very similar to the shape of an 8). The distributed DC-CGANs instead has a more consistent and clearer representation of all the digits.

5.4 CIFAR10 [42]

The ANOVA test run on the CIFAR10 dataset produces a value for the F-statistic equal to 22.12 for the red channel, 103.20 for the green channel, and 400.47 for the blue channel, which lead to p values close to zero. Also in this case, there is strong statistical evidence to claim that the variability between images of different classes is larger than the variability between images of the same class. The computational time measured in wall-clock time is reported in Table 11, and it shows our distributed approach outperforming the standard DC-GANs and DC-CGANs by an order of magnitude. The IS obtained with all the three GANs models is shown in Table 12 and the FID score is shown in Table 13. The distributed approach for DC-CGANs improves the performance of the generator, as it is also shown by a comparison between samples of fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs models in Tables 14, 15 and 16, respectively. It can be noticed that the images produced with distributed DC-CGANs are

Table 11 Wall-clock computational time measured in seconds for the training of DC-GANs, DC-CGANs, and Distributed DC-CGANs on CIFAR10

	Wall-clock Time (seconds)
DC-GANs	375,520
DC-CGANs	393,100
Distributed DC-CGANs	39,011

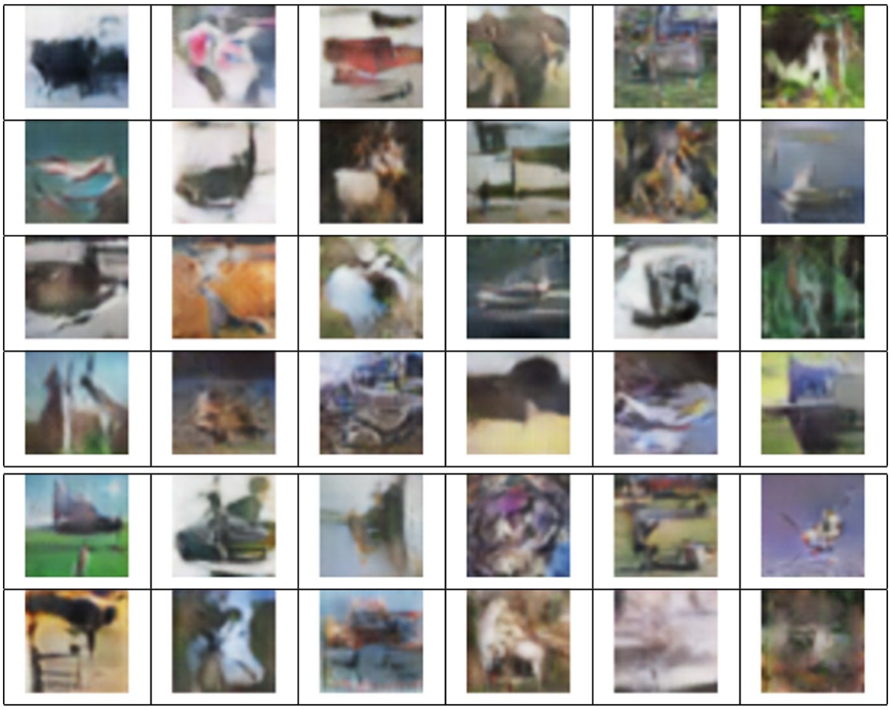
Table 12 Inception score associated with images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on CIFAR10 dataset. The inception score is calculated using 10 splits to collect statistics

	Mean	Standard deviation
DC-GANs	4.39	0.28
DC-CGANs	5.69	0.31
Distributed DC-CGANs	6.43	0.25

Table 13 Fréchet Inception Distance associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on CIFAR10 dataset

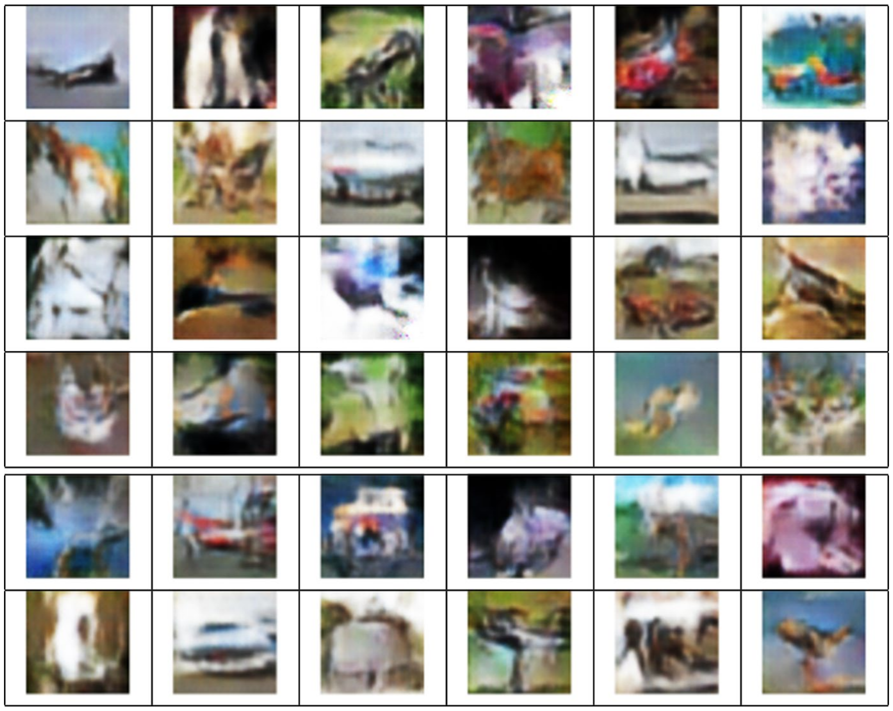
	FID
DC-GANs	14.13
DC-CGANs	11.12
Distributed DC-CGANs	9.41

Table 14 Fake images generated by DC-GANs trained on CIFAR10 dataset



more realistic than the ones produced by DC-GANs and standard DC-CGANs, as they represent objects that are easier to recognize as real. In particular, our distributed approach is the only one of the three methods to generate images that look like cars, trucks, and boats. These objects are more complex to reproduce because of the many components they are made of, and the standard GANs do not

Table 15 Fake images generated by DC-CGANs trained on CIFAR10 dataset



succeed in refining the training to properly capture important features, such as the wheels of a vehicle.

5.5 CIFAR100 [43]

The ANOVA test run on the CIFAR100 produces a value for the *F*-statistic equal to 37.43 for the red channel, 23.74 for the green channel, and 26.08 for the blue channel, which lead to p values close to zero. Also in this case, the variability between images of different classes is larger than the variability between images of the same class. The computational time measured in wall-clock time is reported in Table 17. Our distributed approach still outperforms the standard DC-GANs and DC-CGANs by almost two orders of magnitude. The IS obtained with all the three GANs models is shown in Table 18 and the FID score is shown in Table 19. The results confirm an improved performance using DC-CGANs instead of DC-GANs, which is further improved by our distributed approach to train DC-CGANs. Samples of fake images generated by the generator of DC-GANs, DC-CGANs and distributed DC-CGANs trained on CIFAR100 are shown in Tables 20, 21 and 22, respectively, and the results still show distributed DC-CGANs generating more realistic pictures that can better be recognized as real objects. Our distributed approach is the only approach able to clearly captures relevant feature that characterize objects, such as

Table 16 Fake images generated by distributed DC-CGANs trained on CIFAR10 dataset

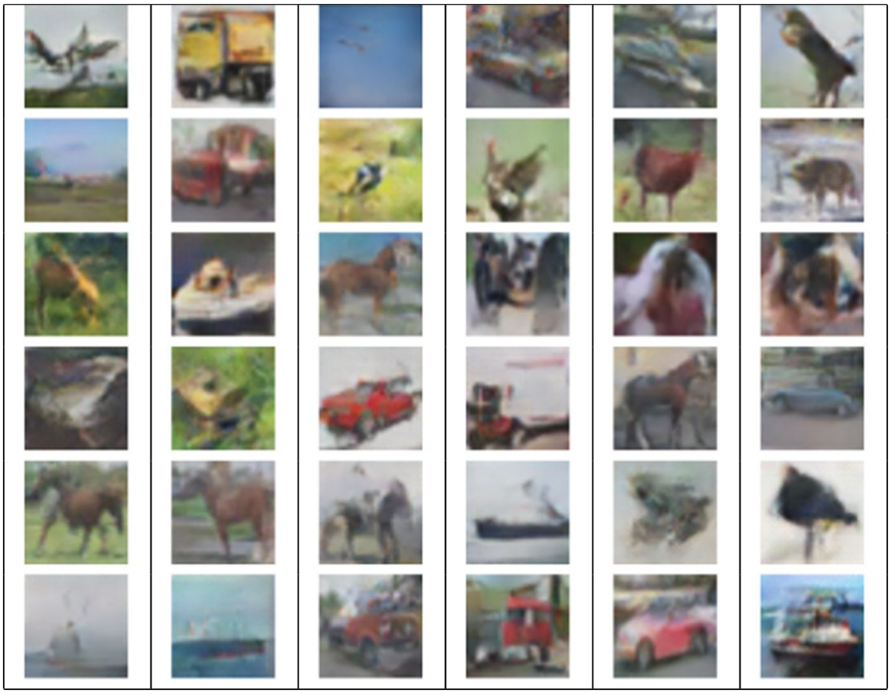


Table 17 Wall-clock computational time measured in seconds for the training of DC-GANs, DC-CGANs, and Distributed DC-CGANs on CIFAR100

	Wall-clock Time (seconds)
DC-GANs	475,520
DC-CGANs	493,100
Distributed DC-CGANs	4,201

Table 18 Inception score associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on CIFAR100 dataset. The inception score is calculated using 10 splits to collect statistics

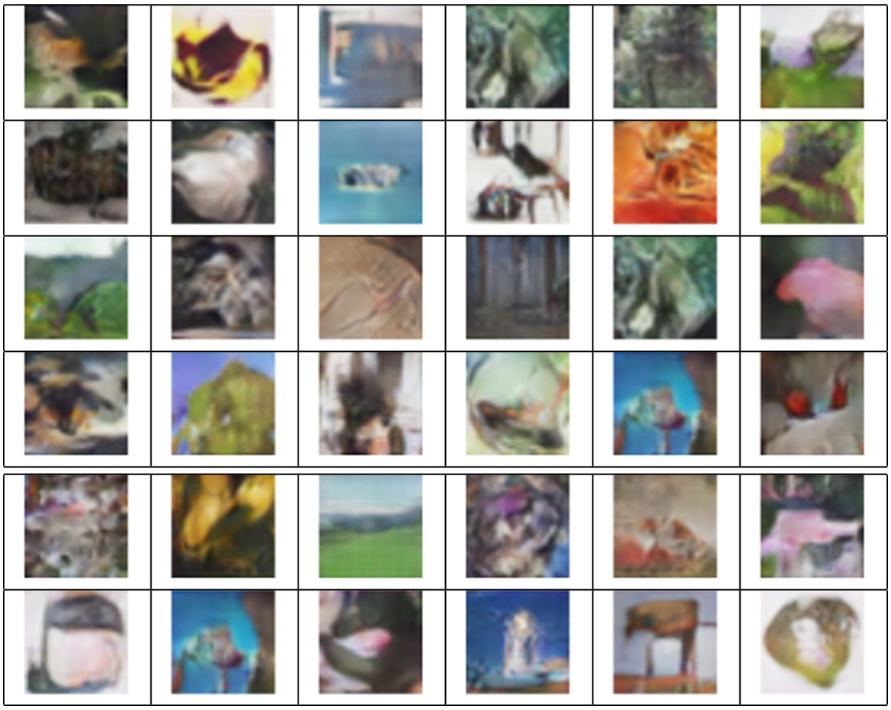
	Mean	Standard deviation
DC-GANs	4.59	0.28
DC-CGANs	5.62	0.23
Distributed DC-CGANs	6.61	0.21

cans, bottles, clocks, sunflowers, fish, trees, faces of human beings, chairs and landscapes. In particular, our distributed approach still exhibits the same quality already shown in CIFAR10 to accurately capture complex features like wheels that characterize vehicles.

Table 19 Fréchet Inception Distance associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on CIFAR100 dataset

	FID
DC-GANs	9.49
DC-CGANs	9.28
Distributed DC-CGANs	9.23

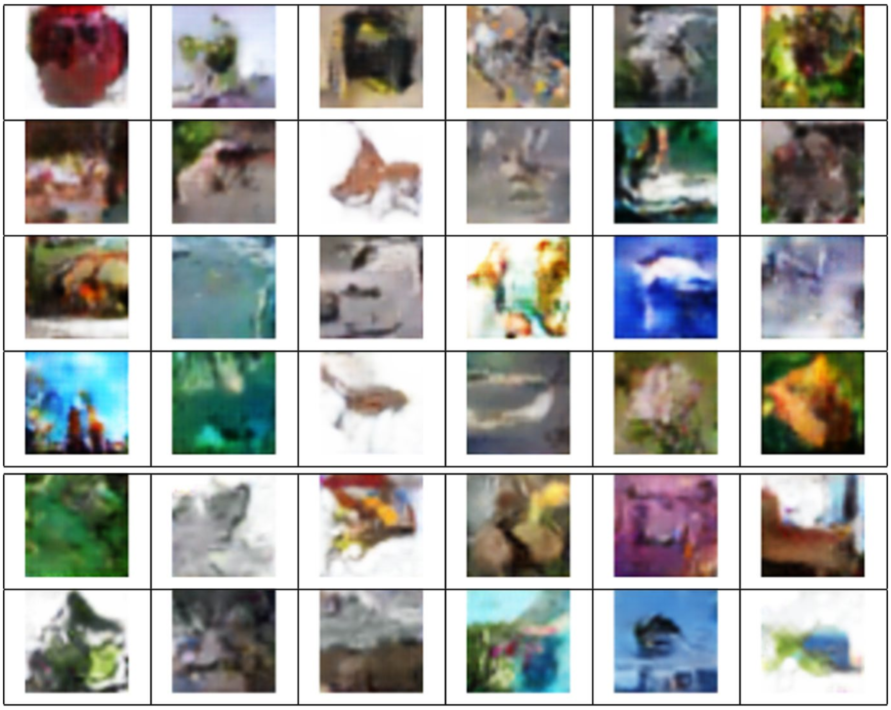
Table 20 Fake images generated by DC-GANs trained on CIFAR100



5.6 ImageNet1k [44]

The ANOVA test run on the ImageNet1k produces a value for the F-statistic equal to 57.43 for the red channel, 43.74 for the green channel, and 56.08 for the blue channel, which lead to p values close to zero. The variability between images of different classes is larger than the variability between images of the same class, and this justifies a distributed training for GANs as we propose. The computational time measured in wall-clock time is reported in Table 23. Our distributed approach outperforms the standard DC-GANs and DC-CGANs by completing the task almost 25 times faster. The IS obtained with all the three GANs models is shown in Table 24 and the FID score is shown in Table 25. The results confirm an improved performance using DC-CGANs instead of DC-GANs, which is further improved by our distributed approach to train DC-CGANs. Samples of fake images generated by the

Table 21 Fake images generated by DC-CGANs trained on CIFAR100



generator of DC-GANs, DC-CGANs and distributed DC-CGANs trained on ImageNet1k are shown in Tables 26, 27 and 28, respectively. Our distributed approach is the only one that effectively captures relevant features to reproduce landscapes, keyboards, houses, birds, vehicles, boats, and geisers, thus showing a better precision obtained by our approach.

5.7 Scaling performance of distributed DC-CGANs

We tested the scalability of our distributed approach to train DC-CGANs by running experiments on OLCF supercomputer Summit. We measured the wall-clock time needed by the distributed DC-CGANs to complete the training as a function of the number of data classes (and thus MPI processes). The parameters for the numerical optimization are the same as discussed before. The training of the model has been performed by distributing the computation through a one-to-one mapping between the `MPI` process and the data classes. Each MPI process was mapped to two NVIDIA V100 GPUs, so that the neural networks for discriminator and generator for each data class would be trained on separate GPUs. For the MNIST and CIFAR10 datasets, the number of MPI processes spans the range from 1 to 10, whereas the number of MPI processes, for the CIFAR100 dataset was set to 10, 20, 40, 80 and 100, and for the ImageNet1k dataset was set to 100, 200, 400, 800 and

Table 22 Fake images generated by distributed DC-CGANs trained on CIFAR100

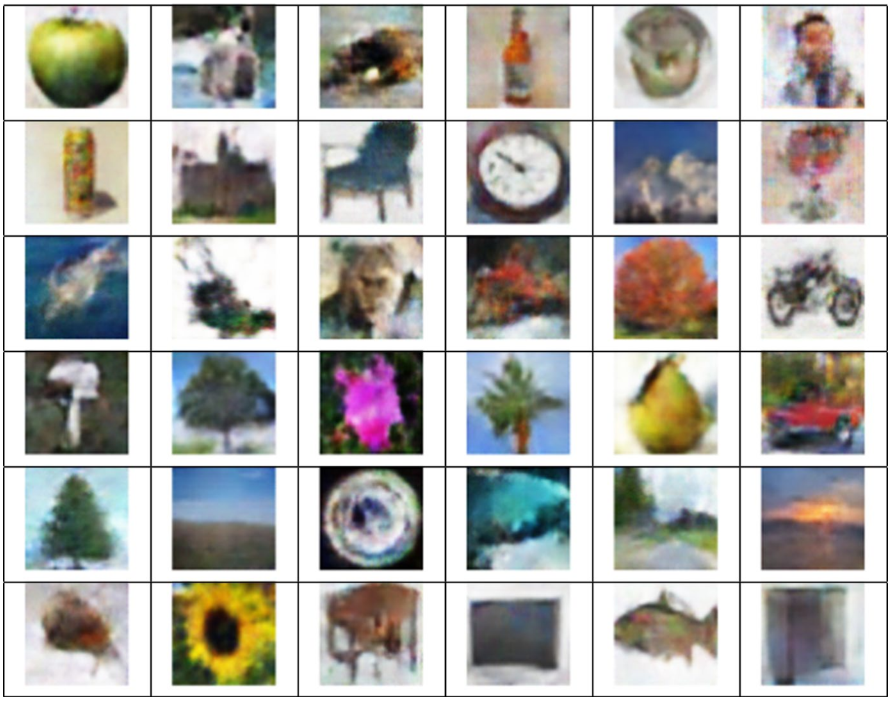


Table 23 Wall-clock computational time measured in seconds for the training of DC-GANs, DC-CGANs, and Distributed DC-CGANs on ImageNet1k

	Wall-clock Time (seconds)
DC-GANs	1,075,520
DC-CGANs	1,293,100
Distributed DC-CGANs	42,011

Table 24 Inception score associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on ImageNet1k dataset. The inception score is calculated using 10 splits to collect statistics

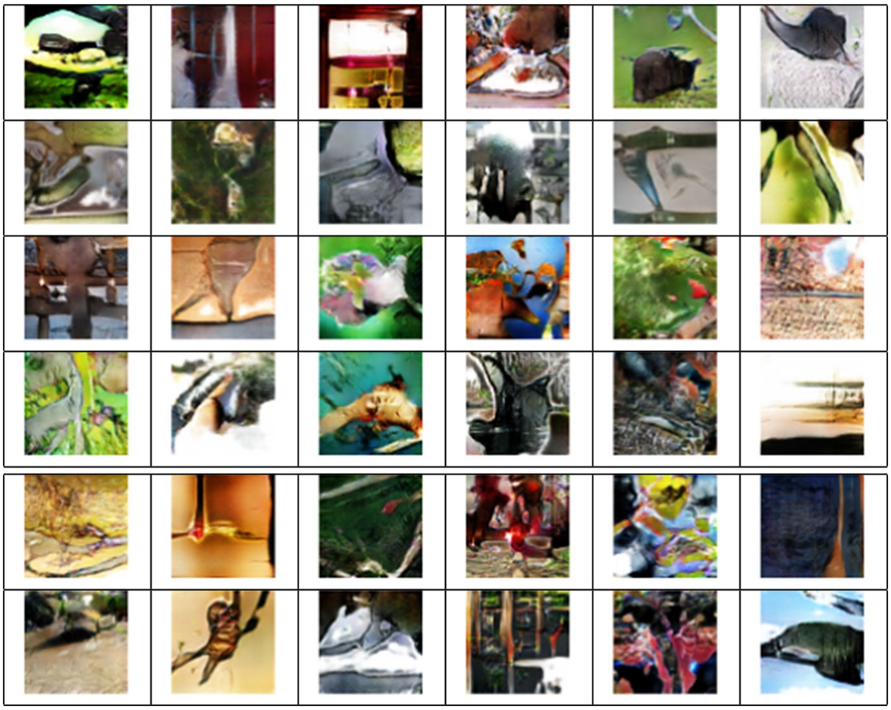
	Mean	Standard deviation
DC-GANs	7.24	0.78
DC-CGANs	8.91	0.85
Distributed DC-CGANs	11.61	1.12

1000. The results for the scalability tests are shown in Fig. 4 by reporting the runtime of the slowest processor. The trend of the wall-clock time shows that the computational time to complete the training is not affected by the increasing number of data classes, as long as the computational workload for each MPI process stays

Table 25 Fréchet Inception Distance associated with fake images generated by DC-GANs, DC-CGANs and distributed DC-CGANs trained on ImageNet1k dataset

	FID
DC-GANs	6.23
DC-CGANs	6.04
Distributed DC-CGANs	5.91

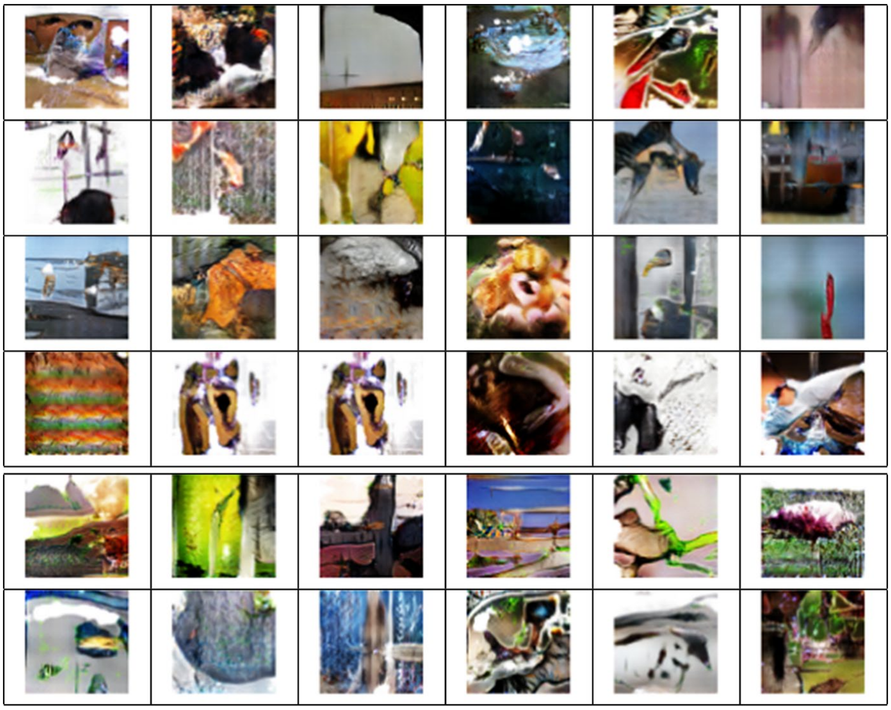
Table 26 Fake images generated by DC-GANs trained on ImageNet1k



fixed, thus showing that weak scaling is obtained by the training of distributed DC-CGANs on all four datasets.

The average GPU usage computed across all the GPUs engaged in the computation is 70% and the GPU memory occupation is around 31% for every numerical test performed.

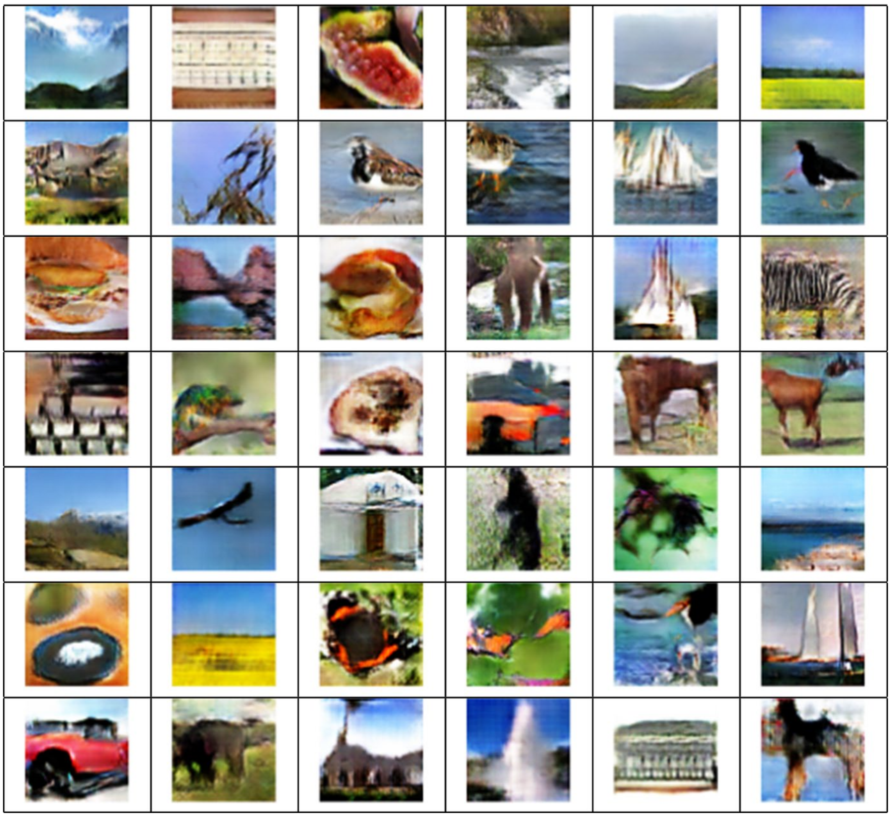
Table 27 Fake images generated by DC-CGANs trained on ImageNet1k



6 Conclusions and future developments

We presented a distributed approach to train DC-CGANs models which reduces the imbalance between the computational tasks of the discriminator and the generator. The distribution of the task is based on partitioning the data in classes and it is justified by the variability of features between data classes that prevails over the variability of features within data points of the same class. The reduced imbalance results in better images generated by the trained DC-CGANs compared to state-of-the-art approaches. In particular, our numerical results show that our distributed approach can significantly improve the performance of simple neural network architectures, such as DC-GANs and DC-CGANs, without necessarily using larger, more complex (and thus more expensive to train)

Table 28 Fake images generated by distributed DC-CGANs trained on ImageNet1k



models as other state-of-the-art require. The results have also shown an almost ideal weak scaling.

Our future work aims at extending the study performed in this work to more complex but also more accurate neural network architectures, such as auxiliary classifier GANs (AC-GANs) [19], residual neural networks (ResNet) [20], self-attention generative adversarial neural networks (SAGANs) [22], and Wasserstein generative adversarial networks (WGANs) [34].

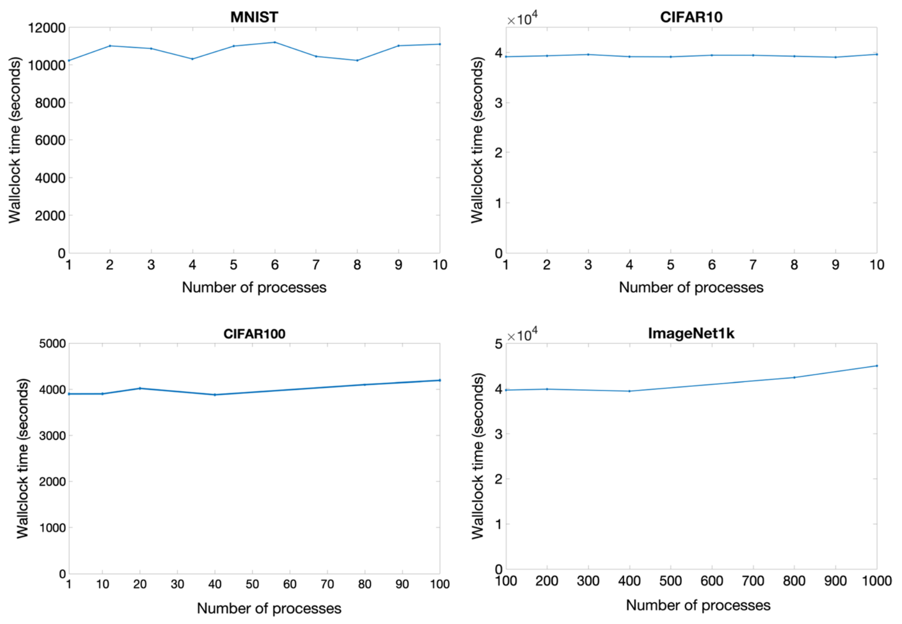


Fig. 4 Results of weak scaling tests for MNIST, CIFAR10, CIFAR100, and ImageNet1k by measurement of the wall-clock time

Acknowledgements Massimiliano Lupo Pasini thanks Dr. Vladimir Protopopescu for his valuable feedback in the preparation of this manuscript.

Research completed through the Artificial Intelligence Initiative sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy.

This work used resources of the Oak Ridge Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Goodfellow Ian J, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative Adversarial Networks. arXiv:1406.2661 [cs,stat]
2. Radford A, Metz L, Chintala S (2016) Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs]
3. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X (2016) Improved Techniques for Training GANs. arXiv:1606.03498 [cs]
4. Bertsekas D (2019) Multiagent Rollout Algorithms and Reinforcement Learning. arXiv:1910.00120 [cs]. version: 1
5. Liu M-Y, Huang X, Yu J, Wang T-C, Mallya A (2020) Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications. arxiv.org/abs/2008.02793 [cs.CV]

6. He Z, Zuo W, Kan M, Shan S, Chen X (2017) AttGAN: Facial Attribute Editing by Only Changing What You Want. arXiv:1711.10678 [cs.CV]
7. Goodfellow Ian J (April 2017) NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv:1701.00160 [cs]
8. Mertikopoulos P, Papadimitriou C, Piliouras G (2017) Cycles in Adversarial Regularized Learning. *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms*
9. Elad Hazan, Karan Singh, Cyril Zhang (2017) Learning linear dynamical systems via spectral filtering. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems. Curran Associates Inc, USA
10. Dauphin Yann N, de Vries Harm, Bengio Y (2015) Equilibrated Adaptive Learning Rates for Non-convex Optimization. arXiv:1502.04390 [cs]. version: 1
11. Ruder S (2017) An Overview of Gradient Descent Optimization Algorithms. arXiv:1609.04747 [cs]
12. Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. page 39
13. Ward R, Wu X, Bottou L (2018) AdaGrad stepsizes: Sharp Convergence over Nonconvex Landscapes, from any Initialization. arXiv:1806.01811 [cs, stat]. arXiv: 1806.01811 version: 1
14. Zhao Y, Li C, Yu P, Gao J, Chen C (2020) Feature Quantization Improves GAN Training. In Hal Daumé III and Aarti S, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11376–11386. PMLR, 13–18
15. Schäfer F, Anandkumar A (2020) Competitive Gradient Descent. arXiv:1905.12103 [cs, math]
16. Pascanu R, Gulcehre C, Cho K, Bengio Y (December 2013) How to Construct Deep Recurrent Neural Networks
17. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014). Going Deeper with Convolutions. arXiv:1409.4842 [cs]
18. Sreekumaran H, Hota A R, Liu Andrew L, Uhan Nelson A, Sundaram S (July 2015) Multi-Agent Decentralized Network Interdiction Games. arXiv:1503.01100 [math]
19. Odena A, Olah C, Shlens J (Aug 2017) Conditional Image Synthesis with Auxiliary Classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia, 06–11. PMLR
20. Wang M, Li H, Li Fang (2017) Generative Adversarial Network based on Resnet for Conditional Image Restoration. *ArXiv*, abs/1707.04881
21. Karras T, Aila T, Laine S, Lehtinen J (February 2018) Progressive Growing of GANs for Improved Quality, Stability, and Variation. arXiv:1710.10196 [cs, stat]
22. Zhang H, Goodfellow I, Metaxas D, Odena A (Jun 2019) Self-attention Generative Adversarial Networks. In Kamalika C and Ruslan S, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363. PMLR, 09–15
23. Andrew B, Jeff D, Karen S (2019) Large Scale GAN Training for High Fidelity Natural Image synthesis
24. Zhao S, Liu Z, Lin J, Jun-Yan Z, Han S (2020) Differentiable Augmentation for Data-Efficient GAN Training. arXiv: 2006.10738
25. Mirza M, Osindero S (November 2014) Conditional Generative Adversarial Nets. arXiv:1411.1784 [cs, stat]
26. Miyato T, Koyama M (2018) CGANs with Projection Discriminator
27. Kavalerov I, Czaja W, Chellappa R (2019) cGANs with Multi-Hinge Loss. *CoRR*, abs/1912.04216
28. Zhang He, Sindagi Vishwanath, Pate Vishal M (2020) Image de-raining using a conditional generative adversarial network. IEEE Trans Circuits Syst Video Technol. https://doi.org/10.1109/TCSVT.2019.2920407
29. Yang D, Hong S, Jang Y, Zhao T, Lee H (2019) Diversity-Sensitive Conditional Generative Adversarial Networks
30. Zhou P, Xie L, Zhang X, Ni B, Tian Q (2020) Searching towards Class-Aware Generators for Conditional Generative Adversarial Networks. arXiv:2006.14208 [cs.CV, cs.LG]
31. Lucic M, Kurach K, Bousquet O, Gelly S (2018) Are GANs Created Equal? A Large-Scale Study. arXiv:1711.10337 [stat.ML]

32. Ratliff LJ, Burden SA, Sastry SS (2013) Characterization and Computation of Local Nash equilibria in Continuous Games. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 917–924
33. Kingma Diederik P, Ba J(2017) Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs]
34. Arjovsky M, Chintala S, Bottou Léon (2017) Wasserstein Generative Adversarial Networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11. PMLR
35. Gauthier J (2015) Conditional Generative Adversarial Nets for Convolutional Face Generation - Technical report, Stanford University
36. Barratt S, Sharma R (2018) A Note on the Inception Score. arXiv:1801.01973 [cs, stat]
37. Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S (2017) GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv:1706.08500 [cs.CV, cs.LG]
38. Casella G, Berger R (2001) *Statistical Inference*. Duxbury Resource Center
39. Summit - Oak Ridge National Laboratory's 200 petaflop supercomputer https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
40. ...Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, Alché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems. Curran Associates Inc, USA
41. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/l
42. The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html
43. The CIFAR-100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html
44. ImageNet. http://image-net.org/