Self-improving classification performance through GAN distillation

Matteo Pennisi Simone Palazzo Concetto Spampinato
PeRCeiVe Lab
Department of Electrical, Electronic and Computer Engineering – University of Catania, Italy
matteo.pennisi@studium.unict.it, {simone.palazzo,concetto.spampinato}@unict.it

Abstract

The availability of a large dataset can be a key factor in achieving good generalization capabilities when training deep learning models. Unfortunately, dataset collection is an expensive and time-consuming task, especially in specific application domains (e.g., medicine). In this paper, we present an approach for overcoming dataset size limitations by combining a classifier with a generative adversarial network (GAN) trained to synthesize "hard" samples through a triplet loss, to encourage the model to learn class features which may be under-represented or ambiguous in a small dataset. We evaluate the proposed approach on subsets of CIFAR-10 in order to simulate a low data availability, and compare the results achieved by our method with those obtained when training in a standard supervised setting over the same reduced set of data. Performance analysis shows a significant improvement in accuracy when training the model on GAN-generated hard samples: our GAN distillation approach improves accuracy in the reduced dataset scenario by about 5 percent points, compared to standard supervised training. Ablation studies and feature visualization confirm that our generative approach is able to consistently produce synthetic images that allow the model to improve its performance even with low data availability.

1. Introduction

The availability of large datasets can be a key factor when training deep learning model, whose learning capabilities easily lead the model to overfit the data, thus causing generalization issues. Large datasets help to better model the variability of the process under analysis and prevent the network from employing spurious information (e.g., noise and artifacts) as a shortcut to solve the task at hand. However, building datasets of a size that is large enough to successfully train a deep model may be a complex task. Data collection is usually costly and time-consuming; moreover, in certain domains — e.g., medical imaging — collecting additional samples may become impractical for several reasons, such as privacy concerns and machinery availability. Hence, when it is not possible to effectively increase a dataset's size, countermeasures to overfitting must be taken on the methodological side.

A common technique employed to increase the apparent size of the dataset — especially in vision tasks — is data augmentation, i.e., creating multiple versions of a sample by applying random modifications (e.g., cropping, flipping, color jittering). Of course, the main information content of the edited sample is the same, so the impact of data augmentation, though significant, is limited. Another approach that helps to reduce overfitting on small datasets is to use pre-trained model (for instance, on ImageNet [11]). In this case, training starts from a stable configuration in the parameter space and from a meaningful set of features, leading to a solution to the current task that is less likely to focus on non-relevant signal components in the attempt to bypass general feature learning. This behavior can also be strengthened by freezing some portions of the model, reducing its learning capability but at the same time preventing it from moving too far from the starting solution. However, even when using pre-trained models, their complexity leads nonetheless to overfitting, if the size of the dataset is not large enough.

Recently, some solutions have been explored that employ generative adversarial networks (GANs) [3] to synthesize data samples in order to artificially increase the dataset used to train a model [1, 9, 10, 13], but the gain in performance obtained with current techniques is limited (about 0.2%); for this reason, this type of methods have not gained momentum.

We here resume the idea of leveraging the capability of generative models to synthesize high-quality and diverse images in order to enhance classification performance without requiring new, manually-annotated, samples. Indeed, dataset diversity is the key for successful real-world deployment: regardless from the size of the dataset, modelling or generating long tails of the data distribution may allow classifiers to boost their performance. Thus, in the attempt to answer the question of whether GANs can be effectively employed to accomplish this task, we here propose

an approach that enforces synthetic images represent *hard samples* for the classifier according to hard mining learning strategy. The key intuition behind this work is that hard sample synthesis makes it possible to provide more informative training samples for increasing robustness and better accuracy.

Indeed, the contribution of augmenting a dataset with images belonging to the mode of the data distribution is relatively limited and has more of a regularization effect (akin to standard data augmentation). Hard sample generation, on the contrary, may actually help the model to learn the most confounding features and input patterns and that are closer to the boundary with other classes, thus providing more informative samples than "easy" data points. We refer to this process as *GAN distillation*, since the trained generator, rather than simply approximating the data distribution, has to learn to *distill* the information related to input patterns that the model is not able to classify, and synthesize original samples from that specific data distribution.

We therefore propose a framework where multiple models — classifier, generator, discriminator, feature extractor — interact in a multi-objective optimization task, where the generator is trained to synthesize realistic images (with the discriminator providing the corresponding error signal) that follow a feature distribution (estimated by the feature extractor) that is similar to samples which the classifier is not able to classify correctly. By combining a standard cross-entropy classification loss, an adversarial loss for image generation and a triplet loss for feature alignment, we are able to synthesize samples that provide information content that is either under-represented in the original dataset or that encodes features at the boundary between classes.

We test our approach on subsets of CIFAR-10, in order to simulate low data availability, using multiple state-of-the-art model architectures. Our results show that our method improves by a significant margin performance in settings where a small fraction of data is available. Moreover, it is also able to yield an accuracy gain on the full dataset, demonstrating that our hard sample generation approach is well-founded and may boost, in a self-training fashion, performance also in large dataset scenarios.

2. Related work

Our approach mainly pertains the synthetic generation of image samples to enhance the performance of visual classifiers. Training over generated data and analyzing classification performance has been investigated in recent studies [9, 10, 13]. However, these works have revealed how popular GANs, when employed as a data augmentation technique, significantly under-perform when compared to classifiers trained on real images [13]. Even when training classifiers with highly realistic images, such as those generated by BigGAN [2], performance is still far from being satisfactory [9, 10], while a minor gain (about +0.2%) is observed when mixing real and synthetic images in specific settings. Thus, generated samples seem to act as regularizers during training, but do not add any additional knowledge to models.

In this work we leverage hard mining, a popular metric learning method based on the identification of hard samples among the set of training data. A related strategy has been recently applied in [1], where GANs are employed for generating hard samples, following a similar motivation as the one underlying our work. The key difference between [1] and our approach stands in the way hard samples are generated. In [1], hard mining is done through a latent code optimization, while the GAN model remains unaltered during training, and only the latent space is updated. This results in navigating the GAN input latent space to seek the samples corresponding to the most complex images. We, instead, force the GAN model (thus it is updated during training) to generate hard samples through a triplet loss by ensuring that their features are closer to those of the most confounding samples for the classifier. Thus, while in [1] only a subset of the input latent space allows for the generation of hard samples, in our approach all input latent values ideally lead to hard samples, ensuring variability in the generated data. Furthermore, [1] uses their approach for training classifiers using only synthetic data, while our method is specifically thought to increase the performance of an already-trained classifier without using any additional annotated sample.

3. Method

The objective of this work is to generate synthetic images purposefully crafted to enhance the performance of the classifier in a limited data setting, by encouraging the generator model to synthesize data points representing *hard examples* for the generator, whose distribution in the original dataset may be under-represented or close to the boundary between classes.

Formally, our architecture consists of a *classifier* $\mathbf{C}$, a *feature extractor* $\mathbf{F}$, a *generator* $\mathbf{G}$ and a *discriminator* $\mathbf{D}$, as shown in Fig. 1: $\mathbf{C}$ is trained to classify input samples, while $\mathbf{G}$ and $\mathbf{D}$ are trained in a generative adversarial framework [3]. The input dataset $\mathcal{D}$ is annotated with labels from c, i.e., $\mathcal{D} = \{(x_i, y_i)\}_{i=1...N}$, with N being the dataset size.

The whole training procedure is divided into three stages, described in the following.

3.1. Classifier pre-training and hard/easy sample labeling

In the first training stage of the proposed framework, we initialize the classifier $\mathbf{C}$ by training it on dataset $\mathcal{D}$ in a

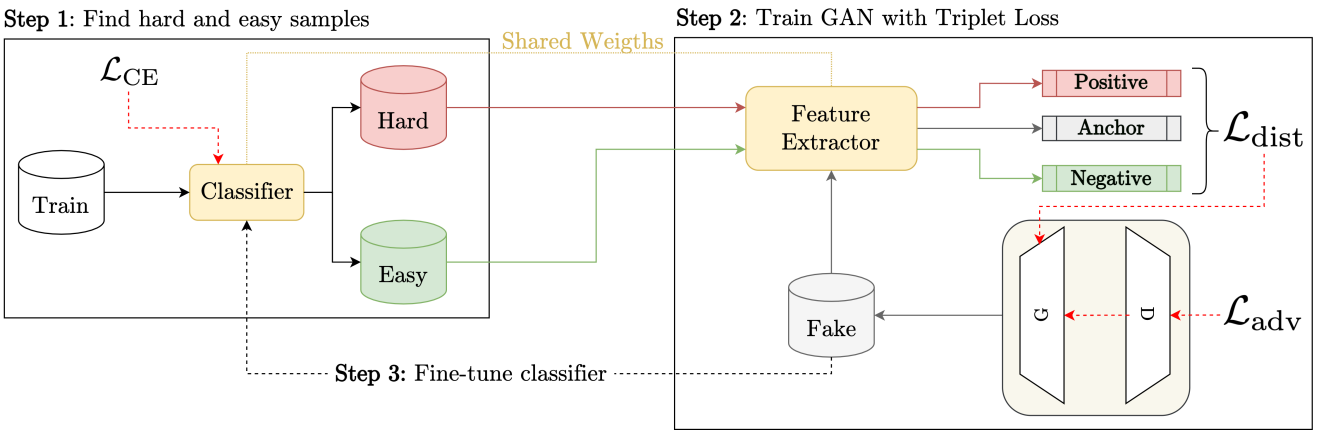


Figure 1: Architecture of the proposed framework. In the first stage, we pre-train the classifier on the dataset, and label training data between easy and hard samples. We then pre-train the GAN using a triplet loss that encourages the model to generate realistic samples that match the feature distribution of hard samples. Finally, we train both models simultaneously, fine-tuning the GAN to approximate the changing hard sample feature distribution.

standard supervised setting with cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}\left[\log \mathbf{C}\left(x_i\right)_{y_i}\right] \tag{1}$$

where $\mathbf{C}\left(\cdot\right)$ is the softmax output computed by the classifier, and y_i addresses the softmax component of the correct class.

During training, we also monitor the model's performance on a validation set: model selection for use in the following training phases is carried out by choosing the model at the lowest validation loss. Then, we go through the training set and divide samples between a *hard sample set* $\mathcal{D}_h$ and an *easy sample set* $\mathcal{D}_e$, such that $\mathcal{D}_h \cap \mathcal{D}_e = \varnothing$ and $\mathcal{D}_h \cup \mathcal{D}_e = \mathcal{D}$. A sample $x_i \in \mathcal{D}$ is *hard* if the predicted label is wrong or if the predicted label is correct but the confidence of the prediction (estimated with softmax) is lower than a certain threshold p_h; otherwise, x_i is labeled as an *easy* sample.

The objective of this pre-training stage is to identify the sets of easy and hard samples, and isolate the distribution of samples that the model struggles with, so that we can train the generator to match such distribution. An alternative to pre-training would be to jointly train the classifier and the GAN from scratch and adapt the distribution estimated by the generator to match a simultaneously-learned distribution of hard samples. However, in our preliminary experiments the models were not able to converge to a solution. Most likely, the data distribution synthesized by the generator in the initial training phases does not actually reflect the distribution of "hard samples" (i.e., encoding features that are either under-represented or located proximally to class boundaries), rather the initial "confusion" of the classifer. As a result, generated samples do not help improve the performance of the classifer, but negatively affect it by introducing noisy associations between synthetic data and labels.

3.2. GAN pre-training

We then separately initialize the GAN generator and discriminator, by training the former to synthesize realistic images and the latter to distinguish between real and generated samples. We therefore follow the standard GAN training procedure, training the discriminator $\mathbf{D}$ on real images from the entire training set $\mathcal{D}$ and on fake images generated by $\mathbf{G}$, with a binary cross-entropy loss function that aims at teaching it to distinguish between real and generated images:

$$\begin{aligned}\mathcal{L}_{\mathbf{D}} = &-\mathbb{E}_{x\sim\mathcal{D}}\left[\log\left(\mathbf{D}\left(x\right)\right)\right] - \\ &-\mathbb{E}_{z\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\log\left(1-\mathbf{D}\left(\mathbf{G}\left(z\right)\right)\right)\right],\end{aligned} \tag{2}$$

where $\mathbf{D}(\cdot)$ and $\mathbf{G}(\cdot)$ compute the output of the generator and discriminator networks, and z is a Gaussian noise vector used as input to the generator.

The generator $\mathbf{G}$ is trained adversarially to fool $\mathbf{D}$ into thinking that synthetic images are real:

$$\mathcal{L}_{\mathbf{G}} = -\mathbb{E}_{z\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\log\left(\mathbf{D}\left(\mathbf{G}\left(z\right)\right)\right)\right]. \tag{3}$$

The overall adversarial loss $\mathcal{L}_{\text{adv}}$ is simply defined as:

$$\mathcal{L}_{\text{adv}} = \mathcal{L}_{\mathbf{D}} + \mathcal{L}_{\mathbf{G}}. \tag{4}$$

In our preliminary experiments, we attempted to train the generator and discriminator from scratch while carrying out the GAN distillation step (described in the following). However, similarly to what explained above concerning the joint training of classifier $\mathbf{C}$, GAN training suffered

its random initialization, generating samples which lacked the desired realism requirements and further degrading the performance of the classifier through the presentation of unrealistic examples.

3.3. GAN distillation

The set of hard samples $\mathcal{D}_h$ represents a portion of the data distribution that the model is not able to learn correctly; hence, we would like our generator $\mathbf{G}$ to be able to synthesize samples from that distribution and help classifier $\mathbf{C}$ to learn features from hard samples that make them more distinguishable from other classes. While the adversarial loss $\mathcal{L}_{\text{adv}}$ encourages the generator $\mathbf{G}$ to synthesize realistic samples, we also need to enforce that generated samples match the hard sample distribution.

To this aim, we employ an additional *distillation loss* $\mathcal{L}_{\text{dist}}$, designed to push features of generated images to be similar of those of hard samples from $\mathcal{D}_h$. Our feature extractor $\mathbf{F}$ shares the same weights as classifier $\mathbf{C}$ — in practice, it is obtained by extracting output features at one of $\mathbf{C}$'s intermediate layers. The definition of $\mathcal{L}_{\text{dist}}$ is based on a triplet loss [12]: given the *anchor* generated samples $x_g = \mathbf{G}(z)$ (for a random z), *positive* hard sample $x_h \in \mathcal{D}_h$ and *negative* easy sample $x_e \in \mathcal{D}_e$, we want to make x_g's features closer to x_h than to x_e. The resulting constraint is implemented by the following loss function:

$$\begin{aligned}\mathcal{L}_{\text{dist}} = &\mathbb{E}_{z,x_h,x_e}\big[\max(\|\mathbf{F}\left(\mathbf{G}(z)\right) - \mathbf{F}\left(x_p\right)\|_2 - \\ &- \|\mathbf{F}\left(\mathbf{G}(z)\right) - \mathbf{F}\left(x_e\right)\|_2 + m, 0)\big],\end{aligned} \tag{5}$$

where $\mathbf{F}$ extracts unit-normalized features from a given input (real or generated) sample and m is a margin value. Intuitively, the aim of the triplet loss is to encourage the generated *anchor* sample to have features whose distance to *positive* (hard) samples is lower by a certain margin (set to 1) than to *negative* (easy) samples, therefore mapping the generator's latent space to $\mathbf{F}$'s feature space where the separation between classes is complex or not well represented in the dataset.

At training time, we jointly fine-tune the whole framework to let the classifier improve by training on the synthetic hard samples, while in turn encouraging the generator $\mathbf{G}$ to produce "harder" samples in order to adapt to $\mathbf{C}$'s feature changes. At this stage, we therefore train $\mathbf{C}$ and $\mathbf{G}$ alternately, one epoch each. GAN training is carried out by simultaneously optimizing $\mathcal{L}_{\text{adv}}$ and $\mathcal{L}_{\text{dist}}$; when training the classifier to minimize $\mathcal{L}_{\text{CE}}$, we prepare input batches that include a fraction p_r of real samples and a fraction $p_g = 1 - p_r$ of generated samples, to prevent the classifier from focusing too much on hard samples and forgetting easy ones.

The pseudo-code of our GAN-distillation approach is shown in Algorithm 1.

Algorithm 1: Gan Distillation Approach

Input : $\mathbf{C}$, $\mathbf{G}$, $\mathbf{D}$ pretrained on $\mathcal{D}$; p_h

```
// Compute hard and easy samples
```
$\mathcal{D}_h = \varnothing, \mathcal{D}_e = \varnothing$
foreach $x_i, y_i \in \mathcal{D}$ **do**
  $\hat{y_i}, prob = \mathbf{C}(x_i)$
  if $\hat{y_i} \neq y_i$ ***or*** $prob \leq p_h$ **then**
    $\mathcal{D}_h \leftarrow x_i$
  else
    $\mathcal{D}_e \leftarrow x_i$
  end
end
```
// GAN distillation
```
$\mathbf{F}$ = $\mathbf{C}$ without last fully-connected layer
while *Training* **do**
  ```
  // GAN Training Epoch
  ```
  foreach $x_i, y_i \in \mathcal{D}$ **do**
    $x_g = \mathbf{G}(z; y_i)$
    Compute $\mathcal{L}_{\mathbf{D}}(\mathbf{D}(x_i, x_g))$
    Backpropagate $\mathcal{L}_{\mathbf{D}}$ and update $\mathbf{D}$
    Sample $x_e \in \mathcal{D}_e$ with $y_e = y_i$
    Sample $x_h \in \mathcal{D}_h$ with $y_h = y_i$
    $f_g = \mathbf{F}(x_g)$
    $f_e = \mathbf{F}(x_e)$
    $f_h = \mathbf{F}(x_h)$
    Compute $\mathcal{L}_{\mathbf{G}}(\mathbf{D}(x_g))$
    Compute $\mathcal{L}_{\text{dist}}(f_g, f_h, f_e)$
    $\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{dist}} + \mathcal{L}_{\mathbf{G}}$
    Backpropagate $\mathcal{L}_{\text{tot}}$ and update $\mathbf{G}$
  end
  ```
  // Classifier Training Epoch
  ```
  foreach $x_i, y_i \in \mathcal{D}$ **do**
    Sample $x_i, y_i \in \mathcal{D}$
    $x_g = \mathbf{G}(z; y_i)$
    $x = [x_g, x_i]$
    $y = [y_i, y_i]$
    Compute $\mathcal{L}_{\text{CE}}(x, y)$
    Backpropagate $\mathcal{L}_{\text{CE}}$ and update $\mathbf{C}$.
  end
end

4. Experiments

4.1. Dataset

We carry out our experiments on the CIFAR-10 dataset, including 50,000 training images and 10,000 test images, at 32×32 pixel resolution. In order to simulate a limited data scenario, we employ only 20% of the training set for actual training, and 10% as a validation set. Generalization performance is evaluated on the entire test set. In the following, we will refer to "training set" as the 20% subset of the orig-

inal training set. We also include experiments showing how the performance of our approach varies when using different fractions of the original training set.

4.2. Architecture and training details

As classifier $\mathbf{C}$, we employ state-of-the-art deep convolutional architectures, namely, AlexNet [8], ResNet-50 [5] and DenseNet-121 [6]. The choice of these models is meant to test the effectiveness of the proposed learning strategy w.r.t. model capacity. Feature extractor $\mathbf{F}$ is obtained by reading the output of the classification model before the last fully connected layer (which projects to class scores). Since we are replicating a scenario of limited availability of data, we adopt Data-Efficient GAN [15] as our generative model, as it is purposely designed to deal with limited amount of training data.

Classifier $\mathbf{C}$ is pre-trained using a batch size of 512 and a learning rate of 10^{-4}, Adam [7] is used as optimizer with default hyperparameters. Training is carried out for 200 epochs; as already mentioned, we select the model's parameters at the lowest validation loss in order to initialize the sets of hard and easy samples. Hyperparameter p_h (threshold of prediction confidence estimated with softmax, below which we identify a sample as hard) is set to 0.6.

GAN pre-training is carried out using a batch size of 64 and the Adam optimizer, with a learning rate of $2 \cdot 10^{-4}$. We train for 600 epochs, alternating discriminator and generator mini-batch updates. Using our default dataset configuration (20% fraction of training samples from the original CIFAR-10 training set), we achieve a Fréchet Inception Distance (FID) of 20.05.

In the final stage, we train the full framework, including the GAN distillation loss, by alternating, at each epoch, between training the classifier and the GAN. Hyperparameter p_r (the fraction of real samples used to build a mini-batch) is set to 0.5. In this stage we train each model for 250 epochs.

4.3. Results

The first battery of tests assesses the contribution that our GAN distillation approach provides to classification accuracy when using only a fraction of the entire training set (20% of the original CIFAR-10 training set). Table 1 reports the results of these tests with different state-of-the-art classification backbones, namely AlexNet, ResNet-50 and DenseNet-121. In all cases, there is a gain in accuracy; however, while with lower capacity models the increase is higher (about 5 percent points), with DenseNet-121 we observe a gain of about 2 percent points. This can be expected, as the baseline itself yields a higher classification accuracy with more complex model, reducing the room for improvement that can be provided by our method.

We then evaluate the performance of our GAN distillation method with respect to the training set size. Indeed, increasing the size of the available training set helps the model learn better features: it is therefore interesting to measure the impact of GAN distillation when data availability becomes less critical. Hence, we estimate the performance of AlexNet using different fractions of the training set, gradually increasing from 20% to 90% (note that 10% is always reserved for validation). Results are given in Table 2, and compare our GAN distillation approach with standard supervised training on real samples only (*Baseline* in Table 2). It can be noticed that GAN distillation improves performance even at larger fractions of the available training data. Moreover, the gain in accuracy has an interesting trend: when up to 40% of the training set is used, GAN distillation provides a higher and higher improvement in accuracy; from 50% on, GAN distillation yields decreasing returns. This is not surprising, since a larger data availability leads to a better sampling of the data distribution, thus helping the baseline training to capture "hard" features from input samples. Nevertheless, even in this case GAN distillation proves useful in identifying complex cases and achieving higher classification accuracy.

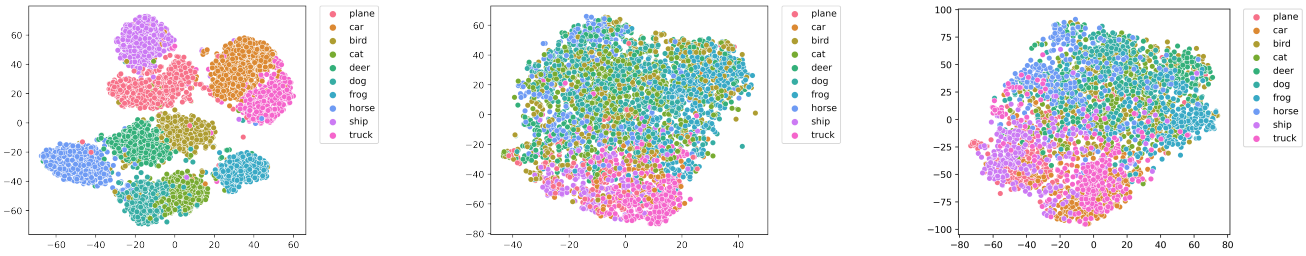
The presented results show that the augmenting a dataset

Table 1: GAN distillation performance with different classification backbones using only 20% of the original CIFAR-10 training set. Results are computed on the test dataset, at the training epoch yielding the highest validation accuracy. The baseline corresponds to standard supervised training on real samples.

	AlexNet	**ResNet-50**	**DenseNet-121**
Baseline	69.63	71.85	79.41
GAN distil.	**74.56**	**77.48**	**81.50**
Gain	+ 4.93	+5.63	+2.09

Table 2: Performance of GAN distillation, compared to standard supervised training on real samples only, while increasing the training set size. AlexNet is used for all the evaluations. Results are taken at the highest validation accuracy. Baseline refers to standard supervised training using only real samples.

	% of Training Set				
	30%	*40%*	*50%*	*60%*	*90%*
Baseline	74.92	77.12	79.38	81.37	83.60
GAN distil.	**80.04**	**82.70**	**84.25**	**85.21**	**86.46**
Gain	+5.12	+5.58	+4.87	+3.84	+2.86



(a) Easy samples before GAN distillation (b) Hard samples before GAN distillation (c) Hard samples generated by GAN

Figure 2: t-SNE executed on features extracted on: (a) easy samples, (b) hard samples before GAN distillation and (c) hard samples after GAN distillation.

.

with GAN-generated hard samples helps improve performance. However, one can wonder whether the benefits are provided by our specific hard mining formulation of the GAN distillation approach, or by the simple inclusion of synthetic samples, acting as an augmentation and regularization technique. We therefore evaluate the contribution of our GAN distillation technique, compared to using a GAN simply as a data augmentation tool, on the setup with AlexNet as a backbone when using 20% of the original training set. As it can be seen from Table 3, data augmentation using standard GAN yields an increase in accuracy of about 2.5 percent points, whereas adding the distillation loss leads to an accuracy gain of about 5 percent points. Hence, this shows that the proposed framework is able not only to generate realistic images that augment the apparent training data size, but also to distill images that are more informative for classification than those generated with a standard generative adversarial objective.

We also evaluate the impact of hyperparameter p_r, i.e., how the ratio between real/fake samples in a batch affects classification performance in GAN distillation. Intuitively, a high value of p_r is a more conservative choice, as more real samples are included in a batch: as a result, the training process focuses more on learning the distribution of available data, limiting the exploration of synthetic hard samples. On the contrary, a low value of p_r forces the classifier to mostly use generated samples, which can be more useful for identifying difficult class boundaries, but may "distract" it from the main data distribution. The results of this experiment are reported in Table 4. The results justify our choice of $p_r = 0.5$ for our default configuration in our experiments, although other settings around that point yield similar performance. As expected, it is still necessary to include real samples to make the classifier work, highlighting that we are not yet at the point to train a classifier with fake (GAN-synthesized) samples only.

We continue our evaluation by observing whether the distribution of samples generated through GAN distillation matches that of hard samples. In this experiment, carried out when using 20% of the original training set, we employ t-SNE [14] to visualize features of easy and hard samples after the initial classifier pre-training stage, and compare such features with those extracted by synthetic hard samples provided by the generator after the GAN distillation stage. Fig. 2 shows the t-SNE visualization of feature vectors extracted by $\mathbf{F}$, revealing how, initially, easy sample are well-clustered (Fig. 2a), while hard samples are not

Table 3: Ablation study to assess the contribution of our GAN distillation loss, compared to the baseline approach (standard supervised training) and to the use of a GAN simply as a data augmentation tool. Results are reported when training on 20% of the original training set, using AlexNet as classification backbone.

	Accuracy	**Accuracy Gain**
Baseline	69.63	-
GAN augmentation	72.17	+2.54
GAN distillation	**74.56**	**+4.93**

Table 4: Classification performance of our GAN distillation method w.r.t. the ratio between true and fake samples in batches of 512 samples. Results are reported when training on 20% of the original training set, using AlexNet as classification backbone.

Real/fake (%)	**Accuracy**
100/0	69.63
75/25	74.15
50/50	74.56
25/75	74.45
0/100	58.55

(Fig. 2b). After GAN distillation, the set of synthetic samples, showed in Fig. 2b, appears to be distributed similarly to real hard samples, as is the objective of the approach. It is interesting to note that, besides enhancing classification accuracy, our GAN distillation approaches is also able to preserve data semantics. Indeed, by closely inspecting Fig. 2c, it can be observed that all inanimate objects (e.g., truck, car, ship, plane) lie in a region different (bottom-left in Fig. 2c) from the one (top-right in Fig. 2c) of the animate objects (e.g., bird, cat, deer, dog). This demonstrates the capability of the GAN distillation loss to group features meaningfully, besides pushing the generator to synthesize hard samples.

It is also interesting to evaluate the progress of GAN and classifier training during the distillation phase; in particular, how the quality of generated hard samples affects classification accuracy. To do so, we evaluate the Frèchet Inception Distance (FID) [4], measuring the realism of generated images, and test classification accuracy at corresponding epochs. In Fig. 3 we report the two quantities at each epoch of the GAN distillation stage, when training on 20% of the original training set. As it can be observed, the GAN

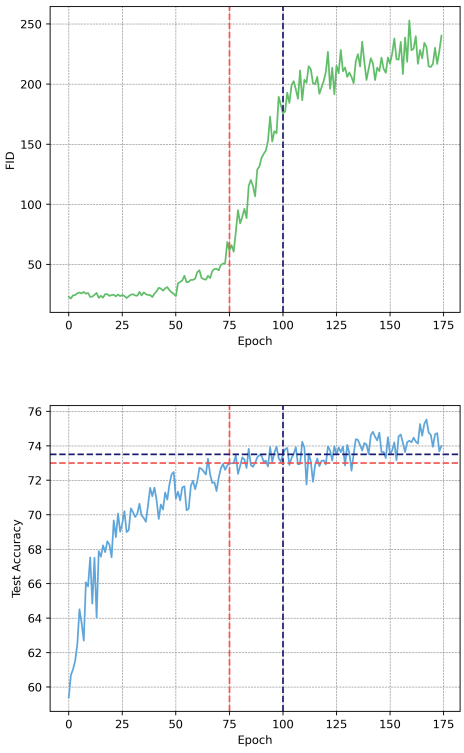


Figure 3: (Top image) FID of the GAN model and (bottom image) test classification accuracy during models' training. Despite mode collapse of the GAN (around epoch 75), the classifier yields good generalization performance, even better than when the GAN does not collapse.

initially achieves a low FID, i.e., it is able to generate realistic diverse images. However, at around epoch 75 in Fig. 3) it starts to exhibit signs of mode collapse — generating the same images for different latent codes — and, accordingly, its FID increases significantly. Nevertheless, it is interesting to note how the classifier performance does not degrade, as expected; rather, it continues to increase, despite marginally. This seems to suggest that the GAN model collapses to a subset of synthesized images that somehow still summarizes hard sample features.

In order to investigate this occurrence, we plot the t-SNE feature visualization of synthetic hard samples generated after mode collapse over the distribution of easy samples. Interestingly, as shown in Fig. 4, it is possible to observe that collapsed examples lie at the borders of each class cluster, confirming our previous interpretation and explaining how they still provide meaningful information for training the classifier. To further verify this behavior, we also compare

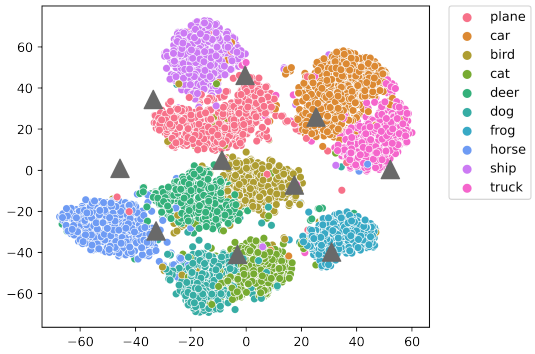


Figure 4: t-SNE visualization of collapsed images. They lies in the border of each cluster, showing their properties of summarizing hard sample features.

classification performance when using a) using *collapsed images* and, b) synthesized images (in the following, *non-collapsed images*) with the GAN model performing well (i.e., before epoch 75 in Fig. 3). In particular, given that

Table 5: Classification accuracy comparison when using collapsed and non-collapsed images in our GAN distillation approach.

	Accuracy
Collapsed images	74.56
Non-collapsed images	70.37

with GAN collapse there is no variability in the generated images per class, we feed the classifier with only one *non-collapsed image* per class, modified through a small Gaus-

sian noise, to replicate the same tiny variability observed in the collapsed ones. Results in Table 5 clearly indicate that the collapsed images retain more information useful to help the classifier than the non-collapsed ones. Indeed, when using a random non-collapsed image, there is almost no improvement in performance.

Fig. 5 shows some examples of images generated before the generator's mode collapse (top) and after mode collapse (bottom).

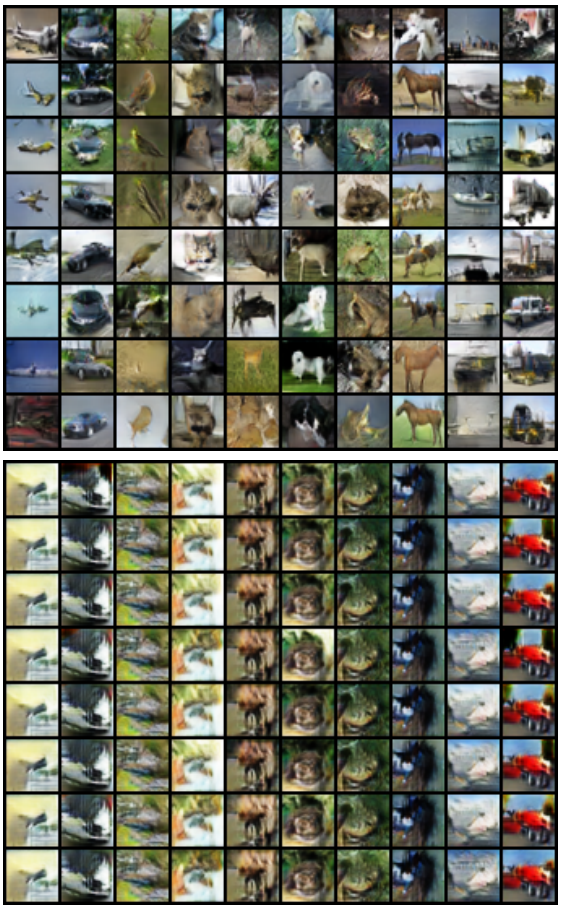


Figure 5: Synthesized images when the GAN does not collapse (top image) and when it does (bottom image).

5. Conclusion

We presented an approach for improving classification performance in low data-availability scenarios, by training a generative adversarial network to synthesize data points belonging to the hard sample distribution for a classifier. We formulated the distribution constraint on the GAN by means of a triplet loss that encourages the generator to synthesize samples whose features (extracted from the classification network) are closer to hard samples than easy samples.

The resulting model achieves better classification performance than standard supervised training, both in low data-availability settings and when the entire training dataset is used, demonstrating that the proposed approach is able to improve generalization of models, regardless of the available amount of data. Moreover, we showed that our GAN distillation approach successfully generates images that match the target distribution of hard samples and that significantly contribute to performance improvements, even after mode collapse of the generator.

These results represent a promising starting point for methods that attempt to use generative models to improve classification accuracy. In the future, we aim at extending our approach to deal with high-resolution images (e.g., ImageNet) in even more complex scenarios with low data-availability.

References

[1] Victor Besnier, Himalaya Jain, Andrei Bursuc, Matthieu Cord, and Patrick Pérez. This dataset does not exist: training models from generated images. In *ICASSP*, 2020. 1, 2

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 2

[3] Ian J. Goodfellow et al. Generative adversarial nets. In *NeurIPS*, 2014. 1, 2

[4] Martin Heusel et al. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017. 7

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 5

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 5

[9] Suman Ravuri and Oriol Vinyals. Classification accuracy score for conditional generative models. In *NeurIPS*, 2019. 1, 2

[10] S. Ravuri and O. Vinyals. Seeing is not necessarily believing: Limitations of biggans for data augmentation. In *ICLR Workshop*, 2019. 1, 2

[11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. 1

[12] Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *NeurIPS*, 2004. 4

[13] K. Shmelkov, C. Schmid, and K. Alahari. How good is my GAN? In *ECCV*, 2018. 1, 2

[14] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *JMLR*, 2014. 6

[15] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient GAN training. In *NeurIPS*, 2020. 5