FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

GASTeN: Generative Adversarial Stress Test Networks

Luís Pedro Pereira Lopes Mascarenhas Cunha

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Carlos Soares

Co-supervisor: Prof. André Restivo

Co-supervisor: Prof. Luís F. Teixeira

October, 2022

GASTeN: Generative Adversarial Stress Test Networks

Luís Pedro Pereira Lopes Mascarenhas Cunha

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. João Moreira
Referee: Prof. Paulo Cortez
Supervisor: Prof. Carlos Soares

October 26, 2022

Abstract

Machine learning (ML) and deep learning (DL) are now ubiquitous in our society, and techniques that enable responsible usage are fundamental to safeguard people from being negatively affected. One particular example of DL's success is image classification. However, DL techniques function as black-box models whose knowledge representation is difficult to comprehend, and understanding the conditions under which they behave correctly is hard. Another example of a DL application is data generation, for which an algorithm known as GANs, mainly used for data augmentation, has achieved remarkable success. In GANs, two networks – the generator and the discriminator – are simultaneously trained. The generator learns to produce realistic data by trying to fool the discriminator, which is trained to distinguish between real and fake samples.

This dissertation proposes a GAN-based approach for synthesizing new data to help understand DL image classifiers. We aim to generate examples that are hard for a given classifier that we could, ultimately, systematically analyze to get information about cases where the model's performance degrades. For that, we opt to generate data classified with low confidence by a classifier. Our approach, dubbed GASTeN, consists of modifying the loss function of the generator to include a new objective, dubbed confusion distance, which reflects how far the generated images are from having the desired output by the targetted classifier, *i.e.*, the one we wish to evaluate. It introduces two hyperparameters, a weight to factor the new loss term, and the duration of pre-training of the GAN without any modifications.

We empirically evaluate our proposal by instantiating it with a DCGAN architecture and a confusion distance suitable for binary classification. In our experiments, we target classifiers of binary subsets of the MNIST and Fashion MNIST datasets. We explore several hyperparameter configurations and target classifiers with different performances, analyzing the algorithm's behavior by collecting quantitative metrics for the two optimization objectives – FID for image realness and the average value of the confusion distance for the goal of confusing the classifier. In our experiments, we show scenarios in which we can obtain a generator with the desired properties of generating data with high realisticness that is mainly classified with low confidence by the target classifiers, along with scenarios where our goal is not attained. We conclude that, while challenging to optimize for both objectives simultaneously, it is possible to achieve images with the desired properties, albeit at the cost of carefully chosen hyperparameters.

Keywords: machine learning, deep learning, image classification, generative adversarial networks, responsible artificial intelligence

Resumo

A aprendizagem computacional (AC) e a aprendizagem profunda (AP) são ubíquas na nossa sociedade, e técnicas que promovem a sua utilização responsável são fundamentais para salvaguardar as pessoas de serem afetadas negativamente pelas suas decisões. Um exemplo particular de sucesso da AP é a classificação de imagens. Apesar disso, as técnicas de AP funcionam como modelos caixa-preta cuja representação do conhecimento é ininteligível, e perceber as condições em que os modelos funcionam corretamente é difícil. Outro exemplo de aplicação da AP é a geração de dados, na qual um algoritmo conhecido por GANs, principalmente usado para aumento de dados, atingiu considerável sucesso. Nas GANs, duas redes – um gerador e um discriminador – são treinadas simultaneamente. O gerador aprende a sintetizar dados realistas tentando enganar o discriminador, que é treinado para distinguir entre dados reais e falsos.

Nesta dissertação, propomos uma abordagem baseada em GANs para sintetizar novos dados que possam ajudar a compreender melhor classificadores de imagem baseados em aprendizagem profunda. O nosso objetivo é gerar exemplos que sejam difíceis para um dado classificador, que podem, em última instância, ser analisados sistematicamente para obter informação sobre cenários em que o functionamento do modelo degrada. Para isso, optamos por gerar dados que são classificads com baixa confiança por um classificador. A nossa abordagem, GASTeN, consiste em modificar a função de perda do gerador de forma a incluir um novo objetivo, *distância de confusão*, que reflete quão longe as imagens geradas estão de serem classificadas como desejado pelo classificador avaliado. O algoritmo introduz dois hiper-parâmetros, um peso que é multiplicado pelo novo termo da função de perda e a duração do pré-treino feito à GAN sem qualquer modificação da função de perda.

Avaliamos empiricamente a nossa abordagem instanciando-a com uma arquitetura DCGAN e uma distância de confusão para classificação binária, utilizando classificadores binários para subconjuntos dos conjuntos de dados MNIST e Fashion MNIST. Exploramos diversas configurações de hiperparâmetros e classificadores com diferentes capacidades, analisando o comportamento do algoritmo através de métricas quantitativas para cada um dos objetivos de otimização – FID para o realismo das imagens e a média da distância de confusão para o objetivo de confundir o classificador. Também analisamos os exemplos gerados. Concluimos que é difícil otimizar os dois objetivos simultaneamente. No entanto, mostramos cenários em que conseguimos obter geradores com as propriedades desejadas, coseguindo gerar imagens realistas que são classificadas com baixa confiança pelos classificadores utilizados.

Keywords: aprendizagem computacional, aprendizagem profunda, classificação de imagens, redes adversariais generativas, inteligência artifical responsável

Acknowledgements

First, I would like to thank my supervisors, André Restivo, Carlos Soares, and Luís Teixeira, for the helpful discussions and guidance.

A huge thanks to all my family, specially my parents and sister, for their support during my entire life.

To my girlfriend, Filipa, for helping me navigate the ups and downs with optimism.

Last but not least, I want to thank my friends, particularly those I made during this journey. Sharing this stage of my life with you has been a privilege.

Luís Cunha

Contents

List of Figures

List of Tables

Abbreviations

AC-GAN	Auxiliary Classifier Generative Adversarial Network
ACD	Average Confusion Distance
AMT	Amazon Mechanical Turk
ATN	Adversarial Transformation Networks
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CV	Computer Vision
DCGAN	Deep Convolutional Generative Adversarial Network
DL	Deep Learning
EMD	Earth Mover's Distance
FGSM	Fast Gradient Sign Method
FID	Fréchet Inception Distance
GAN	Generative Adversarial Network
GASTeN	Generative Adversarial Stress Test Network
InfoGAN	Information Maximization Generative Adversarial Network
IS	Inception Score
KL	Kullback-Leibler
L-BFGS	Limited Memory Broyden–Fletcher–Goldfarb–Shanno
ML	Machine Learning
NLP	Natural Language Processing
PGD	Projected Gradient Descent
ReLU	Rectified Linear Unit
RNG	Random Number Generator
SVM	Support Vector Machine
VAE	Variational Autoencoder
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Network with Gradient Penalty

Chapter 1

Introduction

In recent years, machine learning (ML) and deep learning (DL) [19] have become increasingly popular and are widely used. Reasons for popularity are that, as our society becomes more digitalized, the amount of generated and stored data that can potentially be used for machine learning increases. Also, the increase in computing capabilities has enabled the successful usage of DL, which benefits from specialized hardware, such as GPUs, to train models effectively on large amounts of data.

However, relying on autonomous decision-making algorithms to affect people's lives requires ethical considerations, especially as ML makes its way into more critical applications. This aspect led to the necessity of methodologies and techniques that enable responsible usage of ML and AI in general – responsible AI [3]. One of those techniques consists of accompanying models with Model Cards [41] – a report documenting an ML model with information relevant to the user, such as the intended use case, evaluation metrics across different conditions, and recommendations.

An example of ML popularity is image classification, for which state-of-the-art approaches are DL-based. DL models consist of neural networks with one or several hidden layers and many parameters to optimize, hence deep neural networks (DNN). As the functions learned by these models are a complex composition of several linear and non-linear operations, they do not provide any justification for the given predictions, and the decision process is opaque for humans. As such, these models are black-box and are deemed uninterpretable. Interpretability is, however, a desired property for ML models, primarily due to ethical and legal reasons when applying them in sensitive scenarios. As such, a great deal of work addresses the interpretability of DL models, a field known as explainable AI [3]. The difficulty in understanding the knowledge represented by a DL model also poses the challenge of understanding the model's behavior under different conditions.

Another example of DL application is image generation, for which an algorithm known as Generative Adversarial Networks (GANs) [20], referred to as one of the most exciting subjects in DL, has achieved remarkable success. GANs are mainly used for data augmentation and can generate realistic high-resolution images. In GANs, two neural networks (a generator and a discriminator) compete so that the generator learns to synthesize images by trying to fool the discriminator,

which aims to distinguish real from fake images.

1.1 Motivation

As stated, there is a pressing need for methodologies that enable responsible usage of ML and DL to enable a fair and responsible usage of these algorithms, which can affect people's lives. Model cards [41] can contribute to responsible AI by increasing a model's transparency regarding its performance, use case, and scenarios where it may not work as expected.

One aspect where responsible usage of classification models can be improved, particularly with uninterpretable DL models, is comprehending the scenarios where the model's performance degrades. By identifying and analyzing examples where the model achieves worse performance than expected (*e.g.*, , when compared to the used test dataset), we can better understand its limitations and the conditions that affect its performance negatively. This relevant information could, for instance, be included in model cards so that the model's users are better informed about the model's capabilities.

Ultimately, we think that by understanding and reporting the conditions for which the models become less reliable, ML can be used more responsibly and generalized to more contexts, including sensitive domains.

1.2 Goals

This dissertation aims at solving the problem of generating realistic data that is hard for a binary classifier. As a criterion for difficult data, we choose data that the classifier predicts with low confidence. In other words, we aim to generate data that sits on the border between the two considered classes for a binary classifier. For this data, the model's output is as close as possible to the decision threshold separating both classes. Additionally, our goal is to generate realistic data using the GAN framework, modifying it so that the learned data distribution has a higher frequency of data in the border between the two classes without disregarding the fidelity of the generated data. As the most prominent success cases of GANs are in computer vision (CV), we limit the scope of this work to image classifiers and, in particular, DL-based classifiers.

Adversarial attacks [53, 22] are a similar line of research that aims at finding images that are misleading to the classifier, such that they look like belonging to a class but are classified differently. Our work diverges from it, however, not only on our stated goal but also on the type of data we want to generate. While adversarial attacks find misclassified images to exploit classifiers maliciously, we focus on images in a frontier where the classifier cannot clearly classify them in an attempt to document the classifier's behavior and better understand it.

Our main goal is to adapt GANs to address the described problem, *i.e.*, generate images in the border between two classes for a given classifier. We hope that, by being hard to classify, the data can be used to gather insights and better understand the target classifier. To achieve our goal, we focus on binary image classification and propose a GAN-based methodology. The GAN

modification proposed introduces the target classifier in the training loop, using its output on the generated images as part of a new loss term for the generator.

More generally, our solution involves minimally modifying the original GAN framework by adding a new term to the loss that specifies a desired property of the generated data. As such, a side goal arises from our proposed solution. We intend to contribute to GAN research, particularly how they can be modified to generate images with desired properties and an empirical study of our proposed variant.

1.3 Outline

The remaining of this document contains the following chapters:

- Chapter 2 (p. 4), **Background and Related Work** introduces core concepts relevant to fully understanding this work and related relevant work found in the literature, particularly regarding adversarial attacks.
- Chapter 3 (p. 19), **GANs for Stress Testing** further describes the problem at hand and details our proposal to address it.
- Chapter 4 (p. 25), **Experimental Setup** focuses on the approach taken to analyze our proposal, describing the experiments conducted in detail as well as details related to the implementation.
- Chapter 5 (p. 36), **Results and Discussion** we report and discuss the results of the conducted experiments.
- Chapter 6 (p. 51), **Conclusions** draws final remarks on the work carried out in this dissertation and presents ideas to guide future work.

Chapter 2

Background and Related Work

This chapter introduces concepts that are essential for understanding this work and relevant related work. Section 2.1 introduces the task of image classification and fundamental Machine Learning concepts. In Section 2.2 we introduce Generative Adversarial Networks, some of its variants (*cf.* Section 2.2.1), and evaluation methods found in the literature (*cf.* Section 2.2.2). In Section 2.3 we explore works that focus on adversarial attacks of image classifiers, focusing on those that leverage image generation approaches, particularly GANs.

2.1 Image Classification

Classification is a common task in Machine Learning (ML) and Computer Vision (CV). Classification can be binary if the data belongs to one of two classes, multi-class if the data is classified as one of more than two classes [47], or multi-label if the data can belong to multiple classes simultaneously [58]. More formally, the task can be defined as, given a set of labeled training data consisting of N pairs ($S_{train} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where x_i is a data example and y_i the ground truth label), estimating a function $y = f(x)$ that approximates the real mapping between data and labels. The goal is that the model can generalize to previously unseen data and predict the label correctly for new examples. As such, a test dataset containing data not used during the training process is commonly used to test the generalization capabilities of a classification model.

In CV, classification occurs on visual data. Popular datasets used to benchmark classifiers include MNIST (small grayscale images of digits) [34], FashionMNIST (drop-in replacement for MNIST with garments instead of digits) [65], CIFAR-10 (small color images, each belonging to one of ten classes) [30], CelebA (celebrity portraits labeled with attributes) [35], and ImageNet (images of objects belonging to 1000 classes) [14].

Classical approaches for image classification loosely consist of two steps: feature extraction and classification [54]. For instance, a visual vocabulary of features could be constructed by clustering descriptors extracted from a dataset of images using an algorithm such as SIFT [36]. This vocabulary can be used to build a vector that encodes an image according to the frequency

of the visual words it contains [57]. In this case, the vector that encodes the image is an input example x for ML classification algorithms, like Support Vector Machines (SVM) [12].

Due to advancements in hardware capability and availability, approaches based on deep learning (DL), a subset of ML, have become feasible and now dominate the landscape in CV applications, including classification. These approaches consist of neural network architectures with multiple layers and many learnable parameters that require large amounts of training data. Additionally, CV has contributed significantly to pushing DL research.

Convolutional neural networks (CNN) are the core building block of neural networks for vision-related tasks. These networks contain layers that perform convolution operations, where a kernel K is applied to an image I, outputting O, often called a feature map, as in Equation 2.1 [19].

$$O(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.1}$$

The convolution operation is defined by the kernel's height and width (m,n) (the depth is the same as in the input image), the number of kernels (which defines the depth of the output feature maps), and the stride (which controls the step between windows where the convolution is applied). Padding, *i.e.*, 0-valued columns and rows of pixels added to the image borders, can also be applied to the input. A convolution operation is illustrated in Figure 2.1.

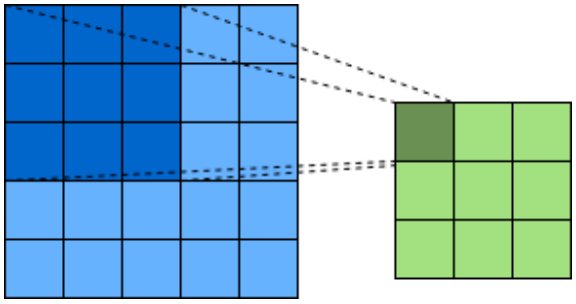


Figure 2.1: Illustration of a convolution operation with a 3×3 kernel on a 5×5 input. With stride 0 and no padding, the resulting feature map has a size of 3×3.

In a CNN layer, a convolution is commonly followed by a non-linear function so that the network can approximate complex non-linear functions, and pooling, for dimensionality reduction. The pooling operation has similar hyperparameters to the convolution. However, it is a fixed function, *i.e.*, does not have trainable weights, that reduces neighboring elements to a single element. Usual pooling functions are *max*, *average*, and *sum*, among others. Pooling operations are commonly strided so that the kernel shifts without overlapping with the previous position. Pooling is exemplified in Figure 2.2.

Unlike classical approaches, CNNs are end-to-end solutions to CV tasks and do not require a carefully engineered feature extraction step. The convolutional blocks learn to extract features and can be followed by a fully-connected layer that outputs a class distribution for classification.

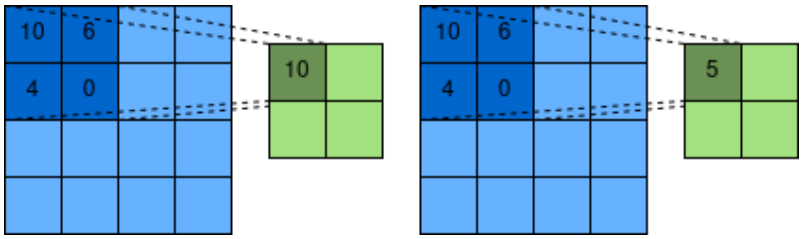


Figure 2.2: Examples of max (left) and average (right) pooling, with kernel size 2×2 and stride 2.

Over the years, several CNN-based architectures have been proposed, such as VGG [50], ResNet [24], and Inception [52]. Figure 2.3 depicts the architecture of the third version of the Inception network.

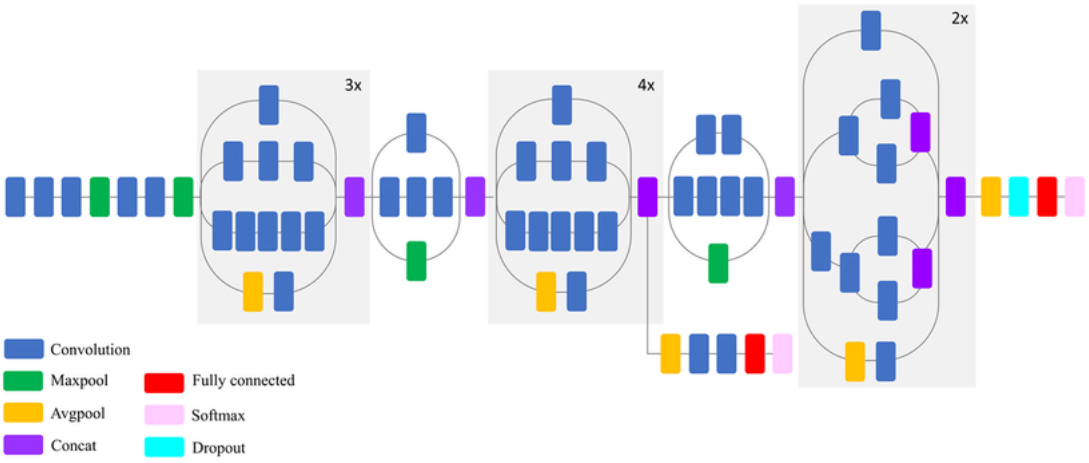


Figure 2.3: Architectural overview of the Inception v3 network, taken from [39]. The number on the upper right corner of the grey blocks represents the times that block is stacked sequentially.

2.2 Generative Adversarial Networks

Generative adversarial networks (GAN) are a class of deep generative models that have been applied successfully to image generation tasks. The GAN framework is a two-player game between two neural networks, a generator (G) and a discriminator (D). The generator is a transformation that maps samples from a noise distribution $z \sim p_z$ to images that match the original data distribution p_d. The discriminator's role is to distinguish between real ($x \sim p_d$) and fake ($\hat{x} = G(z)$) samples, while the generator tries to fool it into classifying fake samples as real. The original GAN formulation [20] is a *minimax* game described by the following equation:

$$\min_G \max_D [\mathbb{E}_{x\sim p_d}[log(D(x_{real}))] + \mathbb{E}_{z\sim p_z}[log(1-D(G(z)))]] \tag{2.2}$$

D aims to maximize the probability of correctly classifying samples, and G to minimize that same probability. Intuitively, if the discriminator has good performance and cannot distinguish synthesized images, it must mean that generated images are realistic. Figure 2.4 depicts an overview of the components present in the classic GAN formulation.

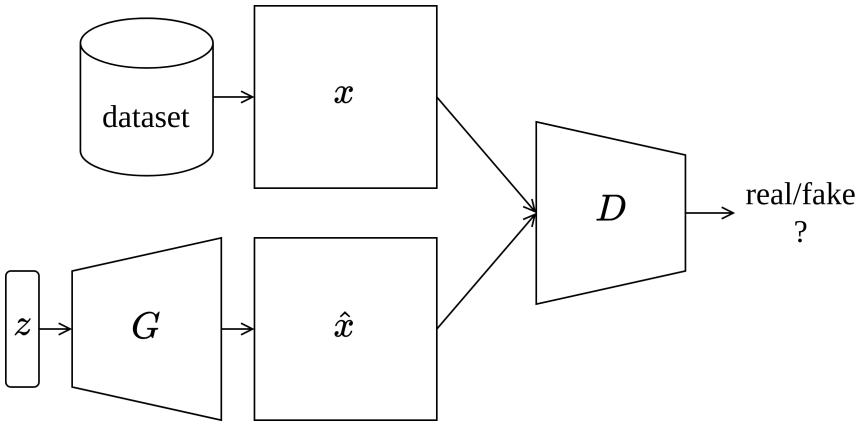


Figure 2.4: Diagram with an overview of the components of a GAN.

During training, the discriminator is fed real samples along with synthesized images. As the source of the images is known, it can be trained as a binary classifier by minimizing the following loss:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_d}[log(D(x))] - \mathbb{E}_{z\sim p_z}[log(1-D(G(z)))] \tag{2.3}$$

Instead of maximizing the discriminator's loss, *i.e.*, minimizing the probability of $G(z)$ being fake, $log(1-D(G(z))$, the authors [20] propose maximizing the probability of $G(z)$ being real. This loss function change improves gradients, avoiding saturation of the loss function earlier in training, where the discriminator confidently classifies the generated images as fake. In this scenario, $log(1-D(G(z))$ is close to 0 (note that if the image is classified as fake then $D(G(z)) = 0$), and the loss function does not provide a useful signal for updating the generator. The generator is then trained to minimize:

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z}[log(D(G(z)))] \tag{2.4}$$

Training GANs effectively presents some obstacles. Firstly, as two neural networks are being updated simultaneously, the players may fail to reach equilibrium, and training may not converge. Another common scenario of GANs train failing is commonly known as mode collapse [21], which consists of the generator not being able to generate diverse images and instead always mapping different noise samples to the same output image.

2.2.1 Variants

As previously mentioned, despite their potential for image generation, training GANs effectively presents several challenges. Several variants and training techniques have been proposed to tackle these difficulties and improve the original GAN. The proposals include changing the network's architecture and modifying loss functions [61, 37]. Other proposals also introduce new training objectives and additional information to the latent space from which the generator samples. In this subsection, we present GAN variants relevant to this work.

The **Deep Convolutional GAN (DCGAN)** [46] architecture introduces techniques and architectural guidelines that improve GAN training. These include eliminating pooling layers and replacing them with transposed convolutions for upsampling in the generator network and strided convolutions in the discriminator. The *tanh* activation function is used in the output layer of the generator, and ReLU activation is used in the remaining layers. Leaky ReLU is used in all discriminator layers. Batch Normalization is used in all network layers, except for the generator's output and discriminator's input layers. In the experiments, the authors use the Adam optimizer [29]. A DCGAN generator is illustrated in Figure 2.5.

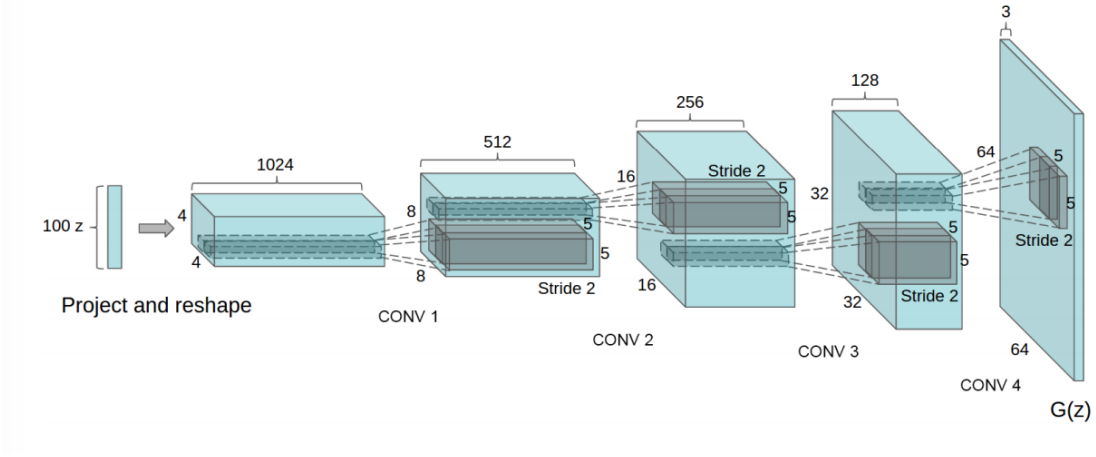


Figure 2.5: DCGAN generator architecture used for 64×64 color image generation, taken from [46]. Transposed convolutions with stride 2 are used for upsampling.

Some GAN variants move away from the purely unsupervised scenario and use additional information. The **Conditional GAN (CGAN)** [40] encodes additional the class of the example y that is passed to G and D along with the inputs of the original setting. As such, G now can create samples ($\hat{x} = G(z|y)$) conditioned on the desired class. Similarly, the discriminator classifies the image as real or fake given y ($C(\hat{x}|y)$). Besides using y to condition data generation, other constraints can be used.

The **InfoGAN** [10] and the **AC-GAN** [42] build upon the **CGAN** by using classification as an auxiliary task.

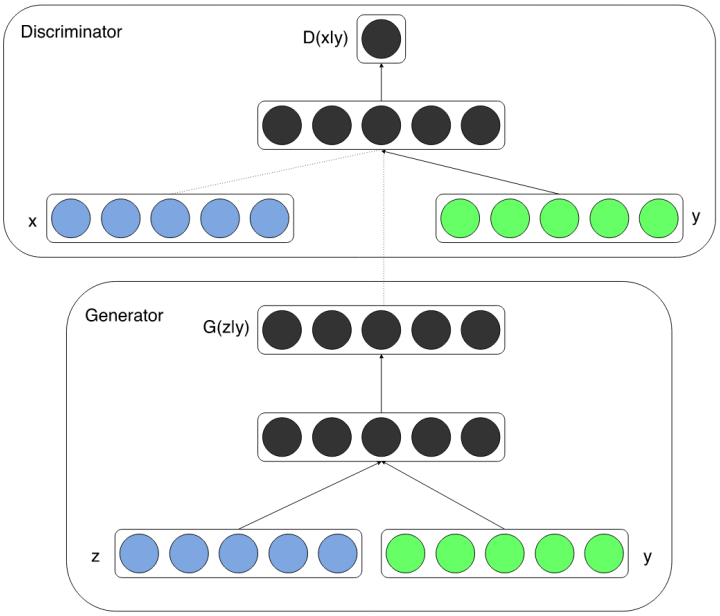


Figure 2.6: CGAN architecture, where additional information y is encoded and fed to both the generator and the discriminator. Taken from [40].

The **Information Maximizing GAN (InfoGAN)** [10] aims at learning meaningful representations c in an unsupervised fashion. To do so, the input of the generator is decomposed in z, which is a noise vector as in the original GAN, and c, a set of variables with meaningful information concerning the image's attributes. A classifier C is tasked with reconstructing c from the generated data. During training, an objective is introduced to minimize information loss between c and $G(z,c)$, thus maximizing mutual information between the meaningful latent representation and the generated data. The *minimax* game formulation results as follows:

$$\min_{G,C} \max_{D} V(D,G) - \lambda L_1(G,C) \ , \quad \lambda > 0, \tag{2.5}$$

where V is the same objetive function as in the original GAN formulation and L_1 a lower bound of the mutual information. For performance reasons, **InfoGAN** can have C sharing all but the last fully-connected layers with D. Experiments show that **InfoGAN** can learn relevant c, such as stroke width for the MNIST dataset and wide or narrow faces on the CelebA dataset [10].

On the other hand, the **Auxiliary Classifier GAN (AC-GAN)** [42] tasks D with classifying the generated image, introducing a new objective that both G and D need to maximize. The new objective consists of the cross-entropy loss to ensure the correct classification of the samples according to the ground truth label for the authentic images and c for the fake images. As such, D, in addition to classifying the realness of an image, it also outputs the class distribution of the image. **AC-GAN** differs from **InfoGAN** in the stated goal, as the latter aims at learning representations, and in the fact that **AC-GAN** only uses the class label as c. **AC-GAN** improves sample quality and diversity over **CGAN**. As opposed to **CGAN**, the additional information c is not fed to the discriminator in the **InfoGAN** and **AC-GAN**. Figure 2.7 contains an overview of the AC-GAN

architecture.

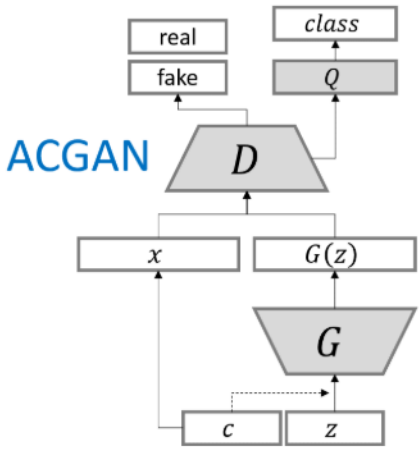


Figure 2.7: Diagram of the AC-GAN architecture, where D functions both as a discriminator and a classifier, from [59].

Training GANs present some difficulties. Notoriously, it is not guaranteed that convergence is achieved during training, particularly if one of the components gets comparably better at its goal than the other. Also, since both components are trained separately using loss functions that do not directly reflect image quality, the loss does not provide adequate information to understand if training is progressing correctly. In order to address these limitations, Arjovsky *et al.* [2] propose a GAN variant with a new loss function – **Wasserstein GAN (WGAN)**. The authors note that the Earth mover (EM) distance (also called Wasserstein-1 distance) between two distributions is continuous and, as such, can be used as an optimization objective for GAN training. As computing the distance is intractable, the authors prove that the discriminator, which they dub *critic*, can be used to estimate the EM distance. The discriminator then aims to minimize Equation 2.6, an estimation of the EM distance. In turn, the generator minimizes Equation 2.7 (the symmetric of the second term of the discriminator's loss).

$$\mathcal{L}_D^{WGAN} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{z\sim p_z}[D(G(z))] \tag{2.6}$$

$$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{z\sim p_z}[D(G(z)))] \tag{2.7}$$

Besides modifying the loss function, removing the sigmoid activation from the last layer of the discriminator (or *critic*) is required since it now performs a regression task instead of classification. By optimizing D, the distance estimation improves, and the gradients provided for optimizing the generator also improve. While the same would happen with the original GAN loss function, the generator's loss saturates, and its gradients vanish in that scenario.

The **WGAN** improves training stability by providing smooth gradients to the generator independently of the discriminator's performance, eliminating the scenario where the discriminator

becomes too good, not providing useful gradients for the generator to train. Additionally, by training the discriminator for more iterations than the generator, the EM estimation provided by the loss function improves and has valuable properties for monitoring training. The authors note, however, that training can become unstable when using momentum-based optimizers such as Adam and instead use the RMSProp optimizer.

A constraint posed in WGAN is that the *critic* needs to be a 1-Lipschitz function. To enforce this constraint, the training procedure of WGAN clips the weights of the *critic*, which leads to it learning simple functions underutilizing its capacity. [23] overcome this by enforcing the constraint differently, by introducing a penalty to the *critic*'s gradient **(WGAN-GP)**. WGAN-GP enabled the successful training of GANs with deeper architectures. It also eliminates WGAN's stability issues when using momentum-based optimizers.

2.2.2 Evaluation

Another challenge of GAN research is how to evaluate generated samples quantitatively to allow fair comparison between different architectures while being sensitive to common problems, such as overfitting and mode collapse. Likelihood-based metrics computed on a test dataset, common across Machine Learning, are unsuitable since GANs do not directly compute the data distribution. Even when using estimation techniques, the average log-likelihood is flawed, as models with high likelihood can produce bad samples and vice-versa [56]. Several quantitative evaluation measures have been proposed [6, 7], notably the Inception Score (IS) [48] and the Fréchet Inception Distance (FID) [26].

Inception Score. The motivation behind the IS [48] is that a good image should be clearly classified by a state-of-the-art classifier and that the generated images should be equally distributed across all classes. As such, an Inception classifier [52] pre-trained on the ImageNet dataset [15] is used.

The classifier is used to obtain the conditional class distribution of the samples ($p(y|\hat{x})$), where low entropy, *i.e.*, a single class with a high value, indicates that the image has a high discriminability. The class distribution of all samples ($p(y)$) is also computed (*cf.* Equation 2.8). It should have high entropy, *i.e.*, all classes should be present with a similar probability, indicating that the model generates images from all classes.

$$p(y) \approx \frac{1}{N}\sum_{i=1}^{N} p(y|\hat{x}_i) \tag{2.8}$$

Then, the Kullback-Leibler (KL) divergence between both distributions is computed and exponentiated, as in Equation 2.9. A higher IS score is better.

$$IS = exp(\mathbb{E}_{\hat{x}}(KL(p(y|\hat{x})||p(y)) \tag{2.9}$$

The authors [48] find that the IS correlates reasonably with image quality. However, IS has drawbacks. It fails to detect overfitting, as the score would be high if the generated images are copied from the dataset. Furthermore, a GAN that always generates the same realistic image for each class would also score highly.

Fréchet Inception Distance. FID [26] computes the distance between the distribution of the representation of real and fake samples in feature space. Intuitively, if the fake images are realistic, then the representations in feature space obtained by a good classifier for the fake data will be similar to the representations obtained for the real data. It also uses the Inception classifier to obtain an embedded image representation by extracting the outputs at a given layer of the Inception network. Ideally, this vector encodes relevant features of the image.

These embeddings are viewed as a multidimensional Gaussian distribution, for which the mean (μ) and covariance (C) are computed. These statistics are computed both for the original dataset (μ_r, C_r) and a large number of synthetic samples (μ_g, C_g). The distance between the two distributions is computed using the Fréchet distance [18], resulting in Equation 2.10. As it is a distance, lower FID values are better.

$$FID((\mu_r, C_r), (\mu_g, C_g)) = ||\mu_r - \mu_f||^2 + Tr(C_r + C_g - 2\sqrt{C_r C_g}) \tag{2.10}$$

FID has improvements over the IS. It is more sensitive to noise addition and, in contrast to the IS, worsens if samples within a class lack variety [26]. A drawback of FID is that the mean and covariance of a multivariate Gaussian may not be a suitable approximation of the distribution. Also, as it relies on an Inception network pre-trained on the ImageNet dataset, it may fail to capture relevant features of images from different datasets.

2.3 Adversarial Attacks

ML models are susceptible to malicious manipulation of input data [13], and image classifiers have also been found to be susceptible to adversarial attacks [53, 22], which consists of purposely constructing examples misclassified by a target classifier. The first proposed adversarial attacks consist of applying small perturbations to an image (x) to produce an adversarial example (x_{adv}) that is misclassified by a given classifier f. The attack can be targeted if the adversarial example is classified as a determined label y_{target} or non-targeted if the goal is just that the adversarial example is misclassified. The assumption that the adversarial example is a sample of the original class y stems from the small perturbation and the proximity between x and x_{adv}. Being L a distance metric and ε a small scalar, the perturbation-based adversarial attacks can be formulated as finding x_{adv} subject to the constraint in Equation 2.11 and either the constraint in Equation 2.12 or the one in Equation 2.13, depending on whether the attack is targeted or not, respectively.

$$L(x, x_{adv}) \leq \varepsilon \tag{2.11}$$

$$f(x_{adv}) = y_{target} \neq f(x) \tag{2.12}$$

$$f(x) \neq f(x_{adv}) \tag{2.13}$$

Several methods have been proposed to find a minimal perturbation r to an image that fools the target classifier. An example is the one proposed by Szegedy *et al.* [53], which became known as the L-BFGS attack due to using a numerical optimization algorithm known as Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS). More specifically, the problem is solved by optimizing Equation 2.14, where c is the smallest positive number such that r satisfies the condition in Equation 2.12, where $x_{adv} = x + r$. The smallest suitable c is found using a line search.

$$c|r| + \mathcal{L}(x+r, y_{target}) \tag{2.14}$$

Another example is the Fast Gradient Sign Method (FGSM) [22], which is less computationally expensive when compared to the L-BFGS attack. It works by computing the gradients of the loss of the target network regarding the original image and then updating the image in the direction that increases the loss. As such, r is computed according to Equation 2.15, where ε is a hyperparameter that controls the perturbation magnitude. The core idea behind this method is to perform gradient optimization regarding the input data in the direction that it increases the model's loss. This idea has influenced several attacks that have since been proposed [31, 32, 38].

$$r = \varepsilon \cdot sign(\nabla_x \mathcal{L}(x,y)) + \mathcal{L}(x+r, y_{target}) \tag{2.15}$$

Carlini & Wagner [9] proposed a method of running adversarial attacks, which, in contrast to the two previously seen, is effective against a form of defense known as defensive distillation [43]. In the C&W algorithm, the constraint in Equation 2.12 is replaced by $o(x+r) \leq 0$, with o being a function that evaluates to a non-positive value if and only if $f(x+r) = y_{target}$. Among the seven possibilities tried, the authors found o to be more effective in the formulation present in Equation 2.16, where $Z(x)$ represents the output of the target classifier f except the last softmax layer and j and j are the indices of y and y_{target}, respectively. By incorporating the constraint in the objective function, the optimization problem becomes minimizing Equation 2.17, where c is a suitable positive constant and D one of the L_0, L_2, and L_∞ norms. The optimization problem is solved using the Adam optimizer [29].

$$o(x) = \max(0, \max_{i \neq t}(Z(x)_i) - Z(x)_t) \tag{2.16}$$

$$D(x, x+r) + c \cdot of(x+r) \tag{2.17}$$

Unlike the previously described optimization-based approaches, Adversarial Transformation

Networks (ATN) [4] are neural networks trained to generate adversarial examples from an input example, with the advantage of low running times at inference time. Training involves minimizing a cost function with two terms, the first (L_X) accounting for minimal perturbation of the input image and the other for inducing a wrong prediction by the target classifier in either a targeted or untargeted fashion. The second term (L_Y) consists of an L_2 loss between the target classifier's (f) output and the desired output. In the presented work, the authors focus on the targeted attack scenario. The target distribution vector is computed by setting the target class to a specified value and renormalizing the remaining distribution so that other classes maintain the original relative order. The authors present two approaches for training the ATN.

Zhao *et al*. [67] use generative models to generate adversarial examples that look natural rather than noisy variants of existing data. The presented approach leverages GANs to sample images of the desired data distribution and induce perturbations in the latent space instead of the images. The intuition is that since G has been trained using the GAN procedure to generate realistic images from the noise vector z, the generator will map z to a natural-looking image independently of its value. The authors use the WGAN variant and train it alongside an Inverter (I), a neural network that learns the inverse mapping of G. I is used to find the latent code z' of an image x, to which perturbations are added. G is used to generate examples from the perturbed latent code, which can query f to find if the attack was successful (*cf.* Figure 2.8).

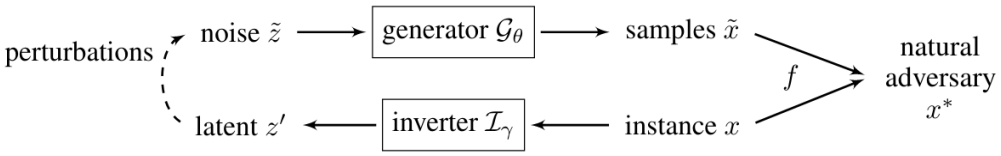


Figure 2.8: Workflow for generating adversarial examples with WGAN as described in [67] (reproduced from [67]).

The authors note that Δz, *i.e*., the average size of induced perturbations to obtain adversaries, shows a reasonable correlation with the classifier's accuracy, suggesting that it could be a measure of its robustness, as more robust classifiers require a more significant perturbation to be fooled. The authors also point out that the difference between the original and adversarial examples can provide insights into the classifier's behavior.

AdvGAN [64] uses GANs to introduce perturbations on images by having G learn a perturbation from an input image sample which is then added to the original instance to generate the adversarial example ($x + G(x)$). In addition, the target classifier f is introduced to the training loop, to obtain $f(x+G(x)$. An overview of the proposed framework is depicted in Figure 2.9. An additional loss term is added to the original GAN objective to force f to classify the generated image as y_t by using f's training loss (*e.g*., , cross-entropy) between f's output and the desired target. As in other constrained attacks, a loss term to minimize perturbation is added. In addition to using AdvGAN in the white-box setting, the authors show how it can be used for black-box attacks by distillation, which consists of optimizing a new classifier to have the same output as f to make it behave similarly. Then, the attack is applied to the distilled network. The authors

find that jointly optimizing the distilled model with the generator, also using perturbed samples to query f, yields better results than performing static distillation with the training data as a previous step. Both white and black-box attacks achieve high success rates against existing defenses. In addition, the authors perform a human study on Amazon Mechanical Turk (AMT) to validate the realism of generated adversarial examples.

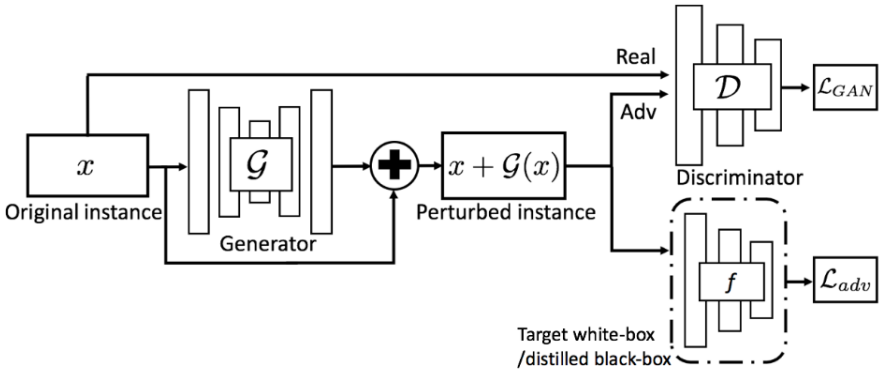


Figure 2.9: Diagram representation of AdvGAN, taken from [64].

Song *et al.* [51] propose an approach for creating unrestricted adversarial examples. In contrast to previously described approaches, the constraint forcing similarity between an actual image and the adversarial example is removed. An AC-GAN (with WGAN-GP loss) is used to generate images conditioned on the desired class. Adversarial examples are found by searching over the latent code z by optimizing a loss function that induces f to classify $G(z|c)$ as y_t (in the targeted case). While restricted adversarial approaches assume that the adversarial example is from the desired class as it results from a minimal perturbation to a real image, the nature of the presented work does not allow that assumption, and a human study on Amazon Mechanical Turk (AMT) [8] is performed.

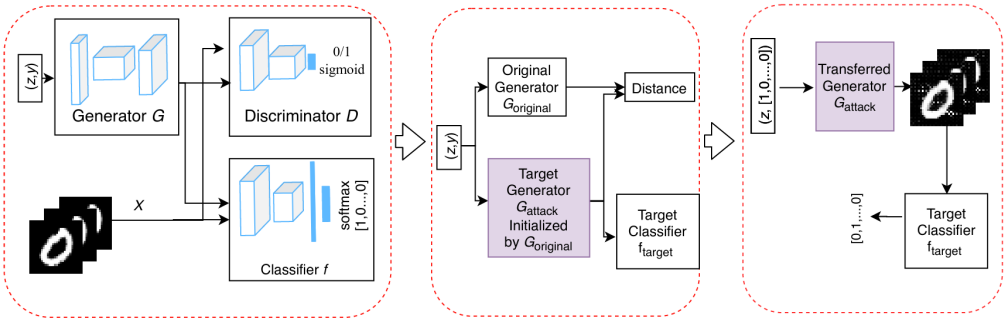


Figure 2.10: AT-GAN steps to produced adversarial examples. First, an AC-GAN is trained. The generator is then optimized to generate adversarial attacks, after which it can be used to generate adversarial samples from noise. Figure taken from [60].

.

Unlike the attacks described above, AT-GAN [60] aims to learn the distribution of adversarial

examples. It works in two steps. Firstly, a GAN is used to learn the distribution of realistic images, resulting in a generator that maps from z to x ($G_{original}$). An AC-GAN with WGAN-GP loss is used. Then, $G_{original}$'s weights are copied to a new generator G_{attack}. Then, G_{attack} is optimized such that, for the same z, it generates images that are similar to those of $G_{original}$ but are misclassified by f. Ideally, the resulting G_{attack} generates adversarial examples from any input noise. Experiments show that the proposed attack has higher success rates than [51]'s approach. Figure 2.10 depicts the workflow for generating adversarial attacks with AT-GAN.

Like AT-GAN [60], Dunn *et al.* [16] propose a method to generate unrestricted adversarial attacks that requires pre-training of a GAN model to generate realistic data. It differs in the second step of the pipeline, and it involves fine-tuning the GAN by adding a new term to the loss function that the generator minimizes. This new term leads the generator to create images that attack the target network, while the original loss function is kept to ensure the data's realism. As the authors note, the gradients of both terms almost always point in different directions, difficulting the optimization process. As such, they combine both terms as in Equation 2.18, instead of simply adding them. κ is a hyperparameter that represents the confidence of the generated, *i.e.*, the desired distance between the logit for the target class and the next largest logit. Additionally, the fine-tune loss function, *i.e.*, the one that includes both terms, is used only with a given probability to avoid the attack term from dominating.

$$\mathcal{L}_{finetune} = s(\mathcal{L}_G) \cdot s(\mathcal{L}_{attack} - \kappa), \text{ where } s(l) = \begin{cases} 1 + exp(l) & \text{if } l \leq 0 \\ 2 + l & \text{otherwise} \end{cases} \tag{2.18}$$

The authors use an AC-GAN for image generation, allowing them to generate images conditional on class and thus generate targeted attacks. The authors test their approach on the MNIST dataset [34], achieving comparable attack efficacy and image realism, by performing a study using AMT [8], compared to Song *et al.* [51]. However, they present the fact that the generator is trained with a focus on a particular network as an advantage since the generator can adapt to the target network, thus being effective against a network that is adversarially trained.

Tao *et al.* [55] also propose a method for generating unrestricted examples, referred to as EGM. As opposed to [60, 16], it decouples the realistic image generation step from manipulating the artificial image into an adversarial example, thus having two components G and T in the adversarial attack generation pipeline. G is the generator of an AC-GAN, capable of generating new images conditioned on class. T is responsible for transforming images into adversarial examples conditionally on the desired target class, similar to the ATN [4] method previously described, and is trained with two objectives. The first is a reconstruction loss to minimize the perturbations induced in the original example and thus allowing the assumption that the adversarial example belongs to the same class as the one generated by G, and a loss to lead the images to morph into examples that fool the target network. The GAN and T are trained simultaneously. At inference time, a single EGM can generate adversarial samples for all combinations of actual class and wrongly predicted class by the target network.

Joshi *et al*. [28] propose a framework for generating a different type of adversarial attacks, dubbed semantic attacks. In contrast to the previously presented attacks, an adversarial example is generated by modifying a variable attribute of the input image, i.e., one that is not discriminative of the image class. For instance, the presence of eyewear and hair length is a variable attribute of an image of a person's face. To do so, the authors use generative models that allow tuning attributes that control semantically significant features and empirically analyze the framework using Fader Networks [33] and Attribute GANs [24] in the presented work using the CelebA dataset [35].

Fader Networks [33] consist of an encoder-decoder architecture, where the encoder embeds the input data to a latent code z, and the decoder is trained to reconstruct the data given z and the variable attributes y. During inference, the user specifies y according to the desired characteristics of the generated image. A critical part of the approach is that the encoder should not include information related to the variable attributes in z. This is achieved using adversarial training by employing a third network, the discriminator. It is trained to predict y given z and an additional objective for the encoder, which should obtain z such that the discriminator cannot predict the attributes y. He *et al*. [25] argue that the method that enforces invariance of features in Fader Networks is too restrictive and propose Attribute GANs (AttGANs). Like Fader Networks, the image generator has two components: an encoder and a decoder. However, instead of employing adversarial learning to disentangle z and y, the feedback from an attribute classifier is used to enforce that the generated images have the desired attributes. In addition, a discriminator to enforce image realism is also used. The reconstruction loss from the decoder using the original attributes enforces that the latent code preserves the invariant features.

The proposed method of generating attacks consists of optimizing the attribute vector of the image, given a pre-trained parametric generative model as described above. The output of the target network on the modified image is used to compute an untargeted adversarial loss. The loss is then backpropagated through the classifier and the generator. The process is iterative until the attribute vector leads to a generated image capable of fooling the target network. The authors conduct experiments using single-attribute and multi-attribute Fader Networks, sequential single-attribute Fader Networks to achieve multi-attribute attacks, and multi-attribute AttGANs, using different attributes. The authors observe that increasing the number of attributes achieves more effective attacks. They show that the method achieves attacks that are more effective than FGSM [22], PGD [38], and the spatial attacks of [17], but not the Carlini & Wagner [9] L_∞-attack.

Qiu *et al*. [45] propose an alternative method to generate semantic adversarial attacks, which they refer to as SemanticAdv. Similarly, the authors leverage a parametric generative model, in this case, StarGAN [11]. The method aims to perform targeted attacks by modifying a single attribute. Firstly, the generative model generates two images, one with the attribute unmodified and another with the attribute with the desired value. A feature map from a specified generator layer is extracted for each image in the generation process. These feature maps are then linearly interpolated using a tensor of the same shape as the feature maps, *i.e*., using a different weight per channel, weight, and height of the feature map. The resulting feature map is fed to the remaining layers of the generator in order to obtain the adversarial example for the given attribute. The

optimization process to obtain the optimal interpolation parameter is achieved by minimizing a function with two objectives, one related to the effectiveness of the targeted attack and the other is a smoothness constraint. The authors analyze the method empirically using several face identity verification datasets, face landmark detection, and street view images.

2.4 Summary

In this chapter, we convey essential concepts for understanding this work. We start by describing core ML concepts and formalizing the task of image classification, including an overview of CNNs, which are the fundamental building blocks that enabled successful approaches for the task (*cf.* Section 2.1, p. 4).

Then, we present GANs, a two-player game where a generator learns a data distribution by generating samples that fool a discriminator (*cf.* Section 2.2, p. 6). GANs have idiosyncrasies that make their application difficult, such as failing to converge and mode collapse, and several variants have been proposed to improve the original formulation. Some modify the network architecture, others propose alternative loss functions, and others leverage extra information (*cf.* Section 2.2.1, p. 8). Another major challenge with GANs is the difficulty of evaluating them. We describe two common approaches found in the literature, the Inception Score and the Frechét Inception Distance (*cf.* Section 2.2.2, p. 11).

Recent work in adversarial attacks, which deal with maliciously finding examples that a model misclassifies, is also reviewed (*cf.* Section 2.3, p. 12). This area deals with a problem closely related to the problem we address in this dissertation, particularly those that aim to generate unrestricted adversarial attacks. In these attacks, the images are not constrained to be small perturbations of existing ones. Despite the similarities between both problems, ours differs from the adversarial attack formulation in the images we aim to generate. Instead of images that are misclassified by a classifier, we aim at generating images that the classifier cannot classify unambiguously as belonging to a class, hoping to achieve data that is useful for understanding the model.

Chapter 3

GANs for Stress Testing

This chapter describes the problem addressed and our proposed solution. Section 3.1 further elaborates on the problem description presented in Chapter 1. Section 3.2 details the GAN variant proposed in this dissertation, dubbed GASTeN.

3.1 Problem Statement

We propose a GAN-based approach to generate data that contains properties regarding a classifier's performance without sacrificing its realisticness. Specifically, in this work, we are interested in data that a classifier cannot confidently classify. In other words, considering the binary classification setting that serves as a case study for empirical evaluation in this dissertation, the desired generated data should be classified as close to the decision threshold, sitting in the frontier between both classes. Additionally, we define the following *desiderata* that our solution should fulfill:

D1: Realistic data. The generated data should be realistic to represent data that could be used in a real scenario.

D2: GAN variant with minimal modifications. As GANs are an active area of research with unsolved challenges, we hope that by introducing minimal modifications to the original GAN framework, our solution can leverage future improvements.

D3: Improve the probability of generating an image belonging to the desired distribution. The proposed variant is only helpful if it improves the chance of generating images with the desired properties.

D4: Modular criteria for target distribution. The solution should enable using different criteria to specify the properties of the goal images.

3.1.1 Scope

This dissertation proposes a GAN-based approach to generate hard data for a given classifier. While hard data could be defined in several ways, we choose to obtain data that a classifier does

not classify confidently. As such, we limit the scope to finding data that is classified with low confidence by a target classifier. Our motivation stems from the fact that this data could help us understand the conditions under which the classifier does not perform as expected, promoting a more considerate use of ML. However, in this dissertation, how the generated data can be used to understand the model is out of scope, and we focus on the data generation aspect of the problem.

Despite proposing an algorithm that could, at least in theory, be used for several types of data, because GANs were initially proposed for CV, we focus on image data for the empirical study conducted in the context of this dissertation. Additionally, because most relevant image classifiers leverage DL methods, particularly CNN architectures, we focus on analyzing those types of classifiers. However, despite limiting the scope for the empirical evaluation performed in this dissertation, the approach is general. It could be used for other classification tasks and classifiers as long as these are differentiable.

3.1.2 Main Hypothesis

Given the problem presented so far, we formulate a hypothesis to guide our work. The hypothesis is as follows:

> *"With minimal modifications, the GAN framework can generate images for which a given classifier exhibits behaves less reliably, classifying them with less confidence while preserving data realism and visual fidelity."*

As stated in the hypothesis, we want to modify GANs to generate data for which the classifier has varying performance levels. More specifically, in this work, we delve into generating images that a model classifies with less confidence, *i.e.*, the classifier's output is closer to the decision threshold. In the binary classification setting, the classifier cannot clearly distinguish between the two classes, and the image is in the border between both classes.

As such, the following objectives will allow testing the stated hypothesis:

O1 Adapt GANs to generate realistic data that is classified in the border by a given classifier.

O2 Study the method's sensibility regarding different hyperparameter values.

O3 Study the method's effectiveness when facing classifiers with different predictive performances.

3.2 Proposal

Our goal is to generate realistic images that are useful to understand classifiers and gain insights into their behavior, particularly in scenarios where the classifiers predict an image's class with less confidence. Admittedly, we became aware of the similarity between the research field of adversarial attacks and the problem we tackle at a late stage of the work developed in this dissertation. Most approaches for adversarial attacks, particularly those that use image generation techniques

such as GANs, were unfamiliar. Our proposed solution was already designed when we delved deeper into studying the field of adversarial attacks. Consequently, our approach does not consist of adapting existing techniques for adversarial attacks to our use case and, unfortunately, does not leverage insights found in said works.

In order to achieve our stated goal, we propose a GAN extension dubbed Generative Adversarial Stress Test Network (GASTeN). The approach is akin to that of Dunn *et al.* [16], which includes extending the generator's loss function with a new objective. Similarly, we leverage the GAN framework as a foundation to generate realistic images and extend it with the classifier to test (C). The classifier's output on the generated images is used as part of an optimization objective for the generator, guiding it to generate images in the desired performance spectrum.

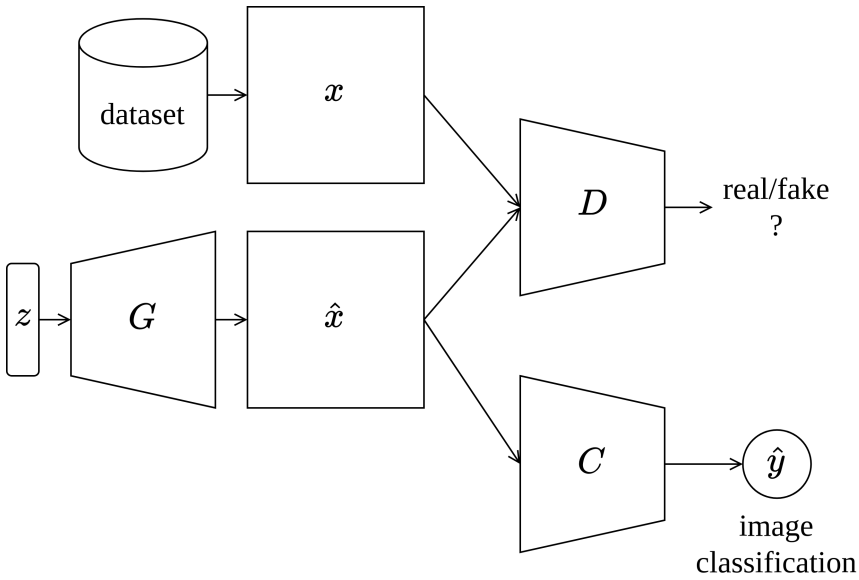


Figure 3.1: Diagram with an overview of the components of GASTeN. It differs from the usual GAN composition in including the target classifier C. C is not updated during training, and its outputs on the generated images are used in the loss function of G.

The modifications introduced by GASTeN, not present in the classical GAN formulation, are the following:

Classifier (C): the classifier whose performance we are testing is introduced in the training loop. As the goal is to analyze its behavior in the frontier, it is not optimized and is used only to classify the images synthesized by the generator $G(z)$. The outputs $C(G(z))$ are then used in the generator's loss function to guide its training. The usage of $C(G(z))$ to optimize G imposes the constraint that the classifier must be fully differentiable so that backpropagation can be applied. Thus, GASTeN is suitable for neural network-based classifiers, but not for some ML approaches that are non-differentiable. The components of GASTeN are depicted in Figure 3.1 (p. 21).

Generator loss function: In addition to training G to deceive D, we introduce a new objective in the loss function of the generator. In contrast to Dunn *et al.* [16], we introduce the new

objective by simply adding it to the original loss. This term measures how far the generated images are from being classified in the border between classes, and is a function of $C(G(z))$, which we dub *confusion distance* (*cd*). Furthermore, a hyperparameter α is used to weight the confusion distance term, resulting in Equation 3.1, effectively serving to control the trade-off between realistic image generation and fooling C.

$$\mathcal{L}_G^{GASTeN} = \mathcal{L}_G^{GAN} + \alpha \cdot cd(C(G(z))) \tag{3.1}$$

Since G's loss function is merely extended, GASTeN can, in theory, be used with any GAN variant as the base. As the role of D remains to distinguish between real and fake samples, no modification is introduced in its loss function.

Confusion distance: the *confusion distance* (*cd*) is a function of C's output, and its name stems from it representing how far the generated images are from being predicted with low confidence by the classifier. We focus on a binary classification setting in order to enable a better analysis of our algorithm's behavior. Thus, the output of C is a scalar between 0 and 1. Our aim is then to generate images that lay in the frontier between two classes, *i.e.*, the classifier cannot confidently distinguish the class of the image. A suitable confusion distance measures the distance from the classifier's prediction to the decision threshold. Since we are going for low-confidence predictions, the true label of an image is not required, and a non-conditional GAN, *i.e.*, the class of the generated image is not controllable, is used. Equation 3.2 shows the resulting expression for a threshold of 0.5.

$$cd(C(G(z)) = |C(G(z)) - 0.5| \tag{3.2}$$

Despite focusing on non-conditional image generation and binary classification in this work, the algorithm is not tightly coupled to the proposed *cd* function. Different functions can be used to tackle other scenarios and objectives. For instance, the confusion distance could be inverted to look for images where the classifier has total confidence. Ultimately, GASTeN can be seen as a broad framework to generate images with properties that can be specified as a differentiable function. In the current work, the property is that the image is predicted with low confidence by a specific classifier.

In order to assess if it is more effective to adapt a generator that is already capable of generating realistic images than to train a generator from scratch using the modified loss function, we introduce a pre-train phase for GASTeN. The training process employed is now a two-step process. The first step consists of pre-training the chosen GAN architecture using the default process to learn a generator ($G_{original}$) to create realistic images that belong to the used dataset distribution. In the second step, the classifier C is added to the process, and the generator $G_{original}$ and discriminator $D_{original}$ that result from the first step are further trained using the GASTeN loss

function. Note that C's weights remain frozen. The components present in each step are depicted in Figure 3.2.

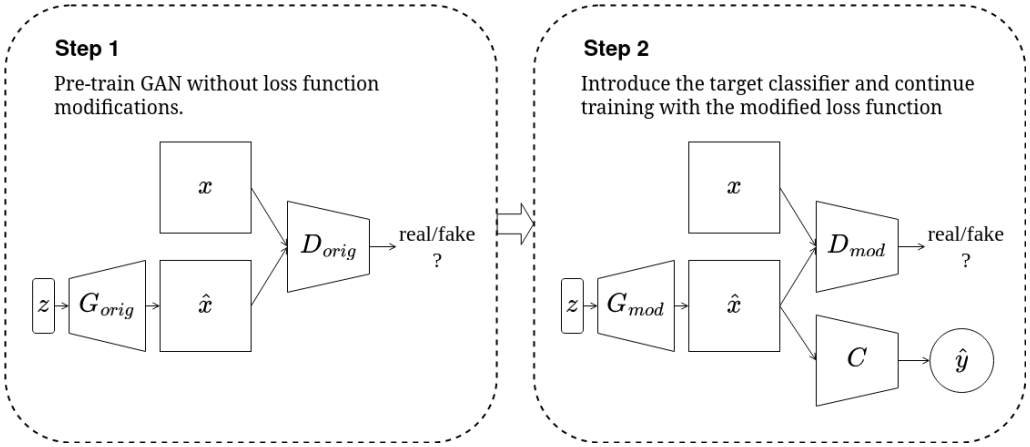


Figure 3.2: Two-step training of GASTeN.

3.2.1 Hyperparameters

Since GASTeN uses a GAN architecture as its basis for image generation, using GASTeN also requires carefully selecting the hyperparameters used when training the original GAN architecture selected. Additionally, the subtleties related to GAN training, such as difficulty to converge, are also present in GASTeN. Careful selection of a suitable GAN architecture, optimizer, and training hyperparameters is required.

GASTeN also introduces two more hyperparameters:

Weight α: factor multiplied to the confusion distance loss term of the generator, defining the weight given to it. It controls the trade-off between generating realistic images and manipulating the generator into generating images with a low *confusion distance*. It is a real-valued positive number.

Number of pre-train epochs: defines the number of epochs for which the pre-train phase is carried out. This pre-train phase merely consists of training the underlying GAN architecture without any modifications, *i.e.*, to achieve a generator capable of producing realistic images without regarding the second goal of confusing the classifier.

3.2.2 Validation Methodology

In order to validate our approach, we run our algorithm on standard benchmark classification datasets used in CV, MNIST [34] and Fashion MNIST [65]. As we want to test it for binary classification, we use a subset of multi-class classification datasets consisting of examples that belong to one of two classes. We instantiate GASTeN using an influential GAN architecture – DCGAN [46] – that achieves good results in terms of image generation in the chosen datasets.

To better comprehend the behavior of the proposed algorithm, we run it using classifiers with varying performance for each dataset. To obtain the classifiers, we train neural networks with architectures based on CNNs, the building blocks of popular neural network architectures for CV. For each classifier, we also repeat runs using different values for the previously described GASTeN hyperparameters, α, and the number of pre-train epochs. By doing so, we can study the behavior of our algorithm regarding three dimensions: quality of the target classifier, quality of the GAN modified in the second training phase, and weight given to the confusion distance loss.

During experiment execution, we gather quantitative metrics that reflect the performance of the algorithm in the two dimensions for which we optimize: the FID score, to measure image quality, and average confusion distance, to see if the GAN is generating images that confuse C. We also gather samples (and C's prediction on them) to inspect them visually, keeping in mind the FID score's limitations.

Chapter 4 (p. 25) further elaborates on the experiments conducted and presents details on neural network architectures, datasets, and hyperparameters used. It also contains relevant information regarding the implementation of the experiments.

3.3 Summary

This chapter explained the problem tackled in this dissertation (*cf.* Section 3.1, p. 19) and the solution we propose (*cf.* Section 3.2, p. 20). Succinctly, we address the problem of using GANs to generate images with desired properties, specifically being classified with low confidence by a given classifier. We propose a GAN variant – GASTeN – which differs from the original GAN framework by including the evaluated classifier, whose output on the generated images is used as the argument of a function that is added to the generator's loss function. Lastly, we introduce the methodology used for validation, which includes running experiments with different combinations of hyperparameters and classifiers to evaluate.

Chapter 4

Experimental Setup

This chapter details the experiments conducted to evaluate our approach. Section 4.1 describes the datasets used. Section 4.2 describes in detail the architecture of the GAN used as well as the architectures of the classifiers. Section 4.3 further details how the experiments are executed, hyperparameters tested, and the evaluation metrics collected. Section 4.4 provides additional details on the implementation of the experiments, including technologies and libraries used.

4.1 Datasets

In order to test our approach, we performed experiments with two commonly used datasets in the CV literature: the MNIST and the Fashion MNIST datasets. This section describes the datasets and how the present work uses them.

4.1.1 MNIST

The MNIST dataset [34] is a collection of labeled handwritten digits. The digits are 28×28 grayscale images, as exemplified in Figure 4.1. The dataset is divided into two sets: a training set with 60000 samples, and a test set, with 10000 samples. In Table 4.1, the class distribution of the MNIST dataset is presented. As can be seen, the dataset is approximately balanced, with each class having a similar number of examples, around 6000 in the training set and 1000 in the test set.

The MNIST dataset is simple and the images have small dimensions, making it a popular dataset for testing algorithms across CV literature. Due to its simplicity, it is easy to achieve low errors, and some authors point out that the dataset is too easy [65]. In spite of that, it is suitable for the problem at hand since the image simplicity makes qualitatively analyzing and comparing the generated images easier. Additionally, it makes obtaining classifiers with small errors feasible without resorting to large complex networks, which reduces the experiments running time and allows for a greater number of hyperparameters to be tested.

In this work, we are interested in studying the applicability of our approach for binary classification tasks to facilitate conclusions regarding our approach's behavior. It should be noted that our approach could be generalized for the multiclass scenario (*cf.* Section 3.2, p. 20). As such,

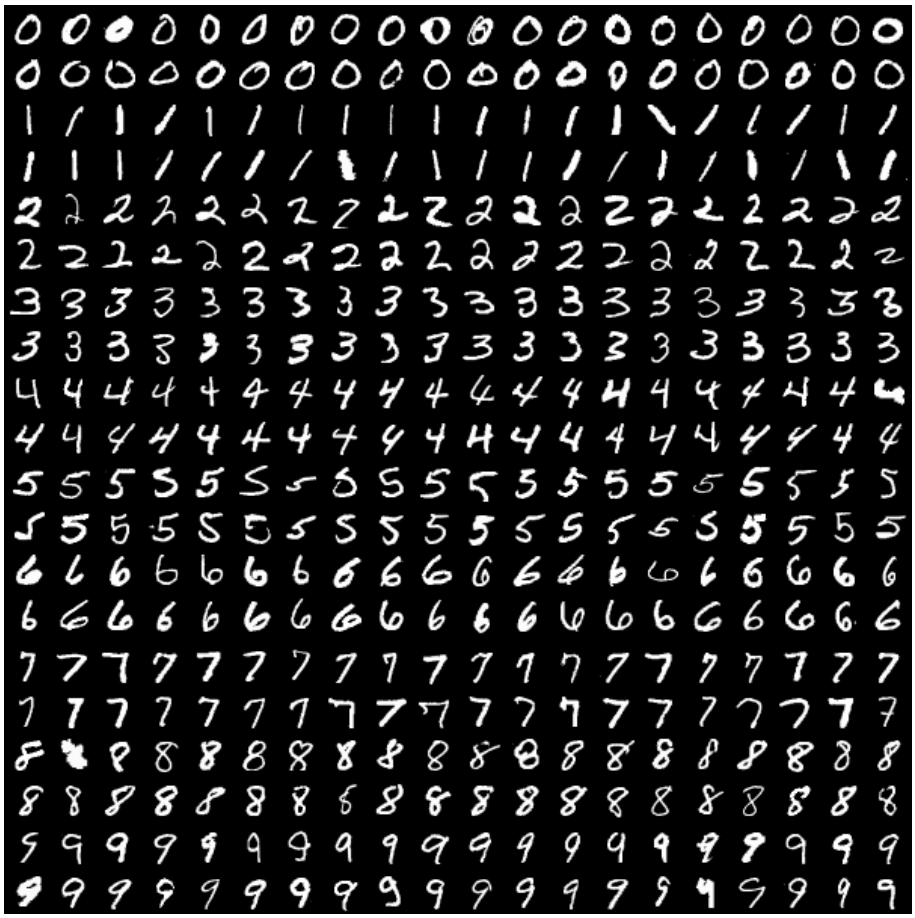


Figure 4.1: MNIST samples. Each class occupies two rows.

Digit	Train Set	Test Set
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009

Table 4.1: Number of samples in the train and test splits of the MNIST dataset.

we run our experiments on datasets constructed by selecting all samples from the original dataset belonging to two given classes. In the remaining parts of this document, we refer to each binary variant of the dataset by mnist-*xvy*, where *x* and *y* are the two digits present in the subset. In order to reduce the number of experiments to be executed, pairs of classes considered representative of

various difficulty problems were selected. Despite having a degree of subjectiveness, care in selecting examples with different difficulty levels for the classifiers was taken. Four pairs of classes were selected: 7 and 1, 8 and 0, 9 and 4, and 5 and 3.

4.1.2 Fashion MNIST

Despite some benefits that come with its simplicity, MNIST is not representative of more complex CV datasets, and as such, Xiao *et al*. [65] propose the Fashion MNIST dataset. It is intended to be a direct replacement to the MNIST dataset, also consisting of 70000 28×28 grayscale images belonging to one of ten classes, split into a train set with 60000 examples and a test set with 10000 examples. However, instead of digits, it consists of clothing items, which can be seen in Figure 4.2. The dataset is perfectly balanced with 6000 examples per class in the training set and 1000 in the test set. The ten classes are: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. As in the MNIST case, we use subsets consisting of examples belonging to two classes deemed to have different difficulty levels. The two pairs of classes selected were sneaker and sandal, and t-shirt/top and dress.

Figure 4.2: Fashion MNIST samples. Each class occupies two rows.

4.2 Model Architectures

In this section, we detail the design decisions made regarding the architecture of the neural networks used, both for the GAN used to generate images and the image classifiers. Table 4.2 contains the nomenclature used when describing the network architectures.

Name	Description
Conv(nf, $k \times k$, s, p)	A convolution with nf filters, kernel k stride s, and padding p. If p is omitted, then padding is applied such that the output has the same size as the input.
ConvTransp(nf, $k \times k$, s, p)	A transposed convolution with nf, kernel k, filters, stride s, and padding p.
MaxPool($k \times k$)	Max pooling operation with kernel k.
BatchNorm	Batch normalization.
FC(n)	Fully connected linear layer with n outputs.
ReLU	Rectified Linear Unit non-linearity.
LeakyReLU(α)	Leaky Rectified Linear Unit non-linearity with angle of the negative slope α.
Tanh	Tanh activation function.
Sigmoid	Sigmoid activation function.
Flatten	Reshape tensor to become one-dimensional.
Reshape(s)	Change tensor shape to s.

Table 4.2: Meaning of the abbreviations used in the description of neural network architectures.

4.2.1 GAN

The proposed approach aims to work with most variants by introducing minimal modifications to the foundational GAN responsible for realistic image generation so that it can leverage further improvements made to GANs. As such, we conduct our experiments using a GAN architecture that does not differ drastically from the original and, while being simple, achieves good enough performance in the used datasets, MNIST and Fashion MNIST. Despite our interest in images with good visual fidelity, it is not in the scope of this work to achieve state-of-the-art results in that regard, so we limit ourselves to using a well-established architecture with hyperparameters suggested in the literature.

The used GAN follows the DCGAN [46] architecture, specified in more detail in Table 4.3, with the non-saturating GAN loss proposed by Goodfellow *et al.* [20]. Following the findings from the original paper [46], we optimize both networks G and D with the Adam optimizer [29] with a learning rate of 0.0002, momentum β_1 of 0.5 and β_2 of 0.999. All network weights are

initialized using a normal distribution with a mean of 0 and a standard deviation of 0.02. As in the architecture used in the original paper for the LSUN dataset [66], we use convolutions with a kernel size of 5 and a stride of 2. However, we reduce the number of features in each convolution to 64 and remove one convolutional block since the MNIST images have lower dimensions and are grayscale instead of RGB. The noise distribution z from which the Generator samples has a dimension of 64.

Generator
FC(4096) BatchNorm ReLU Reshape($256 \times 4 \times 4$)
ConvTransp(128, 5×5, 2) BatchNorm ReLU
ConvTransp(64, 5×5, 2) BatchNorm ReLU
ConvTransp(1, 5×5, 2) Tanh

Discriminator
Conv(64, 5×5, 2, 2) LeakyReLU(0.2)
Conv(128, 5×5, 2, 2) BatchNorm LeakyReLU(0.2)
Conv(256, 5×5, 2, 2) BatchNorm LeakyReLU(0.2)
Flatten FC(1) Sigmoid

Table 4.3: Specification of the GAN used, following the DCGAN architecture.

4.2.2 Classifier

For classification, we use a simple CNN-based architecture, detailed in Table 4.4. The network contains only two convolutional blocks but, despite its simplicity, can achieve low errors in the used datasets and is constructed using operations that are common in DL-based image classifiers, such as convolutions and pooling. We vary the number of features of the convolutions (nf) in order to achieve classifiers with varying error rates and thus study the behavior of our approach depending on the classifier quality. Further details on the specific nf values used and the metrics obtained for each classifier are detailed in Section 4.3, p. 29.

4.3 Experiments

This section describes the experiments conducted to analyze our approach, including the training process of the classifiers used, the data gathered for quantitative and qualitative evaluation, the hyperparameters tested, and how the execution was carried out.

Classifier
Conv(nf, 3×3, 1,)
MaxPool(2×2)
Conv($2 \cdot nf$)
MaxPool(2×2)
Flatten
FC(1)
Sigmoid

Table 4.4: Specification of the image classifier architecture used.

4.3.1 Classifiers

The classifiers used in the experiments follow the architecture described in Section 4.2.2, p. 29. All classifiers were trained using $\frac{1}{6}$ of the train set for validation. The validation set is not used to update the network's weights but to track training progress by evaluating the network after each epoch. The batch size selected was 64. The network was optimized using Adam [29] with the learning rate set to the default, 0.001, and a β_1 of 0.9 and β_2 of 0.999. In order to obtain classifiers with different levels of performance, different values of the nf parameter (*cf.* Section 4.2.2, p. 29) were used for each binary subset of the original datasets (*cf.* Section 4.1, p. 25). The classifiers for the MNIST dataset were trained for one epoch using 1, 2, and 4 as nf values. The classifiers for the Fashion MNIST dataset were trained for three epochs using 4, 8, and 16 as nf values.

Table 4.5 reports the loss and accuracy values obtained for each classifier measured on the original dataset's test split. Unsurprisingly, for the same task, performance, both in terms of loss and accuracy, improves as the number of features nf increases. The highest loss value is 0.43for the classification of digits 9 and 4 with $nf = 1$, and the lowest is 0.04 when classifying digits 7 and 1 with $nf = 4$. The same classifiers correspond to the best and worst accuracies achieved, 98.9% and 82.3%, respectively.

4.3.2 Evaluation Metrics

In our usage scenario, we optimize our GAN, particularly the generator G, with two objectives: generating images that simultaneously look real and are classified with low confidence by the target classifier C. We then need two metrics to reflect both objectives, described in the remainder of this subsection.

4.3.2.1 Image Quality

We resort to the Frechét Inception Distance (FID) metric [26] to quantitatively measure image fidelity. As described in Section 2.2.2, p. 11, the FID metric works by obtaining a vector representation of the real and fake images by extracting the outputs of a given layer of an Inception network pre-trained on ImageNet. The Frechét distance between the distribution of real image

	Dataset	$C.nf$	Loss	Accuracy
MNIST	7 v. 1	1	0.10	97.7%
		2	0.07	97.8%
		4	0.04	98.9%
	5 v. 3	1	0.32	91.3%
		2	0.17	94.2%
		4	0.11	96.6%
	8 v. 0	1	0.33	95.8%
		2	0.10	96.5%
		4	0.05	98.3%
	9 v. 4	1	0.43	82.3%
		2	0.23	92.5%
		4	0.10	97.4%
Fashion MNIST	Dress v. T-shirt/top	4	0.18	93.3%
		8	0.16	94.7%
		16	0.13	95.6%
	Sneaker v. Sandal	4	0.11	95.9%
		8	0.10	96.5%
		16	0.08	97.1%

Table 4.5: Loss and accuracy of the classifiers used in the experiments.

representations and the distribution of fake image representations is computed, obtaining a score where a lower value indicates better image quality.

As mentioned in Section 2.2.2, p. 11, no quantitative measure of the quality of a GAN is without its drawbacks. For our scenario, a limitation of the FID is that the images used to pre-train the Inception network have different characteristics from those present in the datasets used in the current work. While ImageNet contains RGB images of isolated objects, the MNIST dataset contains grayscale images of digits. As such, the embeddings obtained using the Inception network may not accurately represent the characteristics of the data. However, we select the FID metric because both FID and Inception score [48], the two most commonly used metrics, suffer that limitation. Still, the FID is generally regarded as a better image quality indicator. Additionally, the Inception score directly uses the high discriminability of the generated images, *i.e.*, being classified with high confidence, as an indicator of image fidelity. However, that contradicts our second goal of generating images classified with low confidence.

4.3.2.2 Classifier Confusion

To evaluate the generator's ability to generate images that are classified with lower confidence by C, we measure the average value of the confusion distance (*cf.* Section 3.2, p. 20). When using the *cd* proposed and tested in this work, which computes the linear distance between C's output and the decision threshold of 0.5, it results in computing Equation 4.1, where N is the number

of generated examples used. Henceforth, we refer to this metric as average confusion distance (ACD).

$$\frac{1}{N}\sum_{i=1}^{N}|C(G(z_i))-0.5| \tag{4.1}$$

4.3.2.3 Remarks

As mentioned, quantitative measures of GAN image generation qualities are flawed, making a visual inspection of samples indispensable. Also, the ACD does not tell the whole story regarding the distribution of C's output on the generated images. For instance, given 4 generated images, the computed value would be the same if the outputs of C were 0.4, 0.4, 0.6, 0.6, and 0.3, 0.5, 0.5, 0.7[1]. As such, in addition to the quantitative metrics computed, we combine image visualization with assessing the performance of C in the generated images by generating an image where several examples are displayed in a manner that resembles a histogram. The visualization consists of ten groups of three columns, where the group in position i group contains images x such that $C(x) \in [(i-1)*0.1, i*0.1[$. The last group, in position 10, contains images such that $C(x) \in [0.9, 1]$. Figure 4.3 is an example of said visualization, where 200 samples of sevens and ones from the original MNIST dataset are displayed in columns depending on the prediction of the classifier with $nf = 4$ (*cf.* Section 4.3.1, p. 30) on those samples.

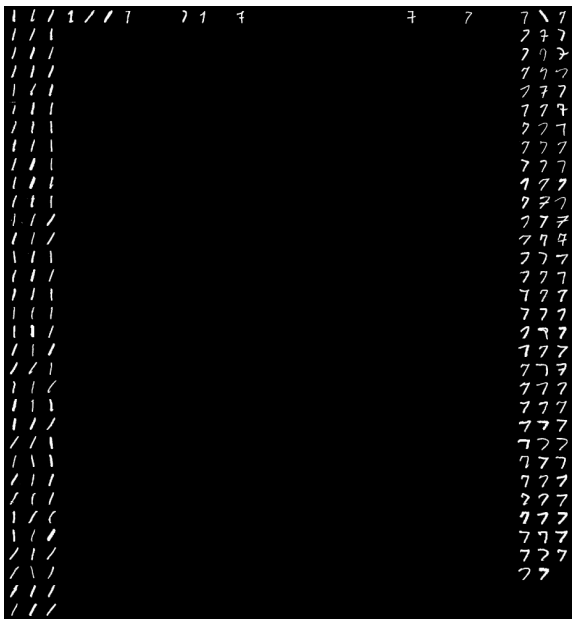


Figure 4.3: Example of a visualization of 200 samples of 1s and 7s from the MNIST dataset. The three leftmost columns consist of images classified between 0 and 0.1, and the three righmost columns consist of images classified between 0.9 and 1.

[1] $\frac{|0.5-0.4|+|0.5-0.4|+|0.5-0.6|+|0.5-0.6|}{4} = \frac{|0.5-0.3|+|0.5-0.5|+|0.5-0.5|+|0.5-0.7|}{4} = 0.1$

The evaluation metrics are computed after each epoch, *i.e.*, when the discriminator sees all data in the dataset once. To better compare the metrics across different epochs and different hyperparameters, we use the same set of z vectors for all experiments. We use 200 z vectors for visual inspection and 2000 vectors for computing the quantitative metrics and plotting the histogram. Using at least 10000 samples is recommended for calculating the FID score since a lower image number can underestimate the FID value. However, we use a smaller number to speed up computation since we are only interested in comparing FID values among our different experiment runs, where all metrics are computed under the same condition.

4.3.3 Hyperparameters

As mentioned in Section 4.3.1, p. 30 we are interested in studying the behavior of our proposal against classifiers with different performances. Additionally, we are interested in exploring the algorithm with different values for the hyperparameters α and the number of pre-train epochs. After preliminary experiments, we selected a possible range for the hyperparameters. The chosen values are presented in Table 4.6. Seven different values are used for parameter α. Setting the value to 0 allows direct comparison to the scenario where a regular GAN without a modified loss is trained. Four values are used for the number of pre-train epochs, including 0, to assess if pre-training is advantageous. All combinations of the mentioned hyperparameters are tested for each classifier. In total, and considering three different classifiers for each binary dataset, $3 * 7 * 4 = 84$ hyperparameter and classifier combinations are tested for each dataset.

Hyperparameter	Values
Weight (α)	0, 5, 10, 15, 20, 25, 30
Num. pre-train epochs	0, 2, 5, 10

Table 4.6: Values explored for each GASTeN hyperparameter.

4.3.4 Execution Pipeline

The experiments are run in two phases to avoid repeating work. For each dataset, the first training phase is executed only once, storing checkpoints at each epoch. Doing that allows starting the second phase with the desired number of pre-train epochs for the different classifiers and hyperparameters. The second phase of training, *i.e.*, including the confusion distance loss, is always started using the same random number generator seed for the same dataset, which makes comparisons between the algorithm's behavior with different classifiers and hyperparameters fairer. Two seeds are used per experiment: one for the first phase and one for the second. We run each experiment three times with different seeds.

4.4 Implementation Details

The described experiments were implemented using the Python 3 programming language so that they can be executed in an easy-to-configure and reproducible manner. Python is a widely used language in ML research due to its ease of use and the vast number of existing libraries and frameworks. Neural networks were implemented using the PyTorch [44], a framework that supports tensor operations with GPU acceleration and automatic differentiation of said operations. Additionally, PyTorch includes implementations of several optimization algorithms, neural network operations, and several utility modules for common tasks required in a Deep Learning pipeline, such as data loading. Automatic differentiation in PyTorch enables easy implementation of custom neural network architectures as the researcher only needs to focus on the forward pass by chaining the desired operations. The framework handles differentiation allowing the backward pass of the backpropagation algorithm to be implicitly defined. The FID evaluation metric [26] was adapted from a PyTorch port [49] of the official TensorFlow [1] implementation in order to use an Inception network as close as possible to the originally employed for computing the statistics. We use the library Pandas [63] for handling and analysing collected data, and Seaborn [62] and Matplotlib [27] for plotting purposes.

To enable several experiments to be run with different configurations, we parametrize them using a YAML file[2]. The YAML file allows complete configuration of the experiment: dataset to use, which two classes to use, directories to read data and store model checkpoints, neural network architecture, and hyperparameters, including those regarding GASTeN. The configuration file also allows specifying seeds to initialize the random number generators of the libraries used, which allows for reproducibility of the experiments. An example configuration file is provided in Appendix A (p. 58).

The artifacts generated during execution, such as generator and discriminator checkpoints, are stored in the computer's filesystem for later analysis. During training, the computed evaluation metrics and generated sample images, in addition to being stored in the computer's filesystem, are logged to the WandB (Weights and Biases) platform [5] using WandB's Python API. WandB allows experiment tracking via an online dashboard and persists the metrics for visualization and analysis. We also store other experiment metadata, such as RNG seeds, configuration, and commit identifiers, which allow access to a specific source code version managed by the git version control system. Thus, all metadata required for reproducibility of experiments is stored. The source code is available on GitHub[3].

The experiments were run using an NVIDIA GeForce RTX 2080 Ti GPU in a server running the Ubuntu Linux distribution with an 8-core CPU.

[2] YAML is a human-readable data serialization format often used to specify configuration files.

[3] https://github.com/luispcunha/gasten

4.5 Summary

This chapter explains the experimental setup used in this work for evaluating our proposal. We present the datasets used (*cf.* Section 4.1, p. 25), which are binary subsets of MNIST and Fashion MNIST, since we are addressing binary classification. We use the DCGAN architecture and CNN based classifiers trained to obtain different loss values (*cf.* Section 4.2, p. 28).(*cf.* Section 3.2, p. 20). For evaluation, we measure the FID score and the average value of the confusion distance (ACD) to compare across runs with different hyperparameters (*cf.* Section 4.3, p. 29). Section 4.4, p. 34 we describe the tools and libraries used to implement the experiments.

Chapter 5

Results and Discussion

This chapter reports the results of the conducted experiments and our findings regarding our proposal. For comparison, we start by presenting the results when using an unmodified GAN in Section 5.1. Section 5.2 analyses the algorithm's behavior subject to different hyperparameter (weight α and pre-train duration) values and against classifiers with various performances by discussing the collected experiment metrics and examples of generated images. Section 5.5 we discuss aspects that may affect the conclusions drawn from these results.

5.1 Unmodified GAN

In Table 5.1, we report the FID scores of the chosen GAN architecture for all datasets, along with the average confusion distance (ACD) for each target classifier. The metrics are collected after training for 50 epochs without any modifications to the loss function, *i.e.*, with $\alpha = 0$. These values serve as a benchmark for later analysis. Ideally, our approach should train G to generate data that is not only classified with low confidence by a target classifier (reduces the ACD) but also realistic looking, with FID values close to those achieved without GAN modifications. In addition, we also report the ACD values for each classifier using the original dataset test data, which are comparable to the ACD with the synthetic samples.

Note that, as expected, lower ACD values are achieved for classifiers with less capacity, which is more noticeable for the MNIST-5v3, MNIST-9v4, and MNIST-8v0 cases, where the classifiers with $nf = 1$ have ACD values around 0.2. These tasks are also the ones with the largest difference between the loss achieved with $nf = 1$ and the loss achieved for higher nf values (*cf.* Table 4.5. For instance, the classifier with $nf = 1$ for MNIST-5v3 has a loss of 0.317, which improves by 0.151 to 0.166 when $nf = 2$. The classifiers with the same parameters for MNIST-7v1 have losses of 0.0972 and 0.0732, respectively (a smaller difference of only 0.024). The relation between the ACD values achieved by default and the loss of the targeted classifier is expected because the loss used (binary cross-entropy) reflects how confidently correct the classifier is on the test set. Since the synthetic images resemble those from the test set, the classifier will naturally predict with higher confidence, *i.e.*, closer to 0 or 1. An exception occurs for dresses and t-shirts on FMNIST,

where the ACD is slightly higher for $nf = 4$ than $nf = 8$. Note, however, that the losses for those classifiers are very similar (0.175 and 0.159, respectively).

Figure 5.1 shows the evolution of the FID score during training of our GAN architecture for classes 7 and 1 of MNIST and classes "Dress" and "T-shirt/top" of Fashion MNIST. We observe that our GAN architecture behaves as desired, improving the FID score as training progresses. Figure 5.2 presents examples of synthesized images using the base GAN architecture used as the foundation for GASTeN in this work. The samples are displayed using the visualization described in Section 4.3.2.3, p. 32, based on the classification by the classifiers with $nf = 4$ for MNIST and $nf = 16$ for FMNIST.

	Dataset	↓ FID ($\mu \pm \sigma$)	$C.nf$	↓ ACD ($\mu \pm \sigma$)	↓ ACD (test set)
MNIST	7 v. 1	9.55 ± 0.21	1	0.426 ± 0.0028	0.428
			2	0.452 ± 0.0025	0.455
			4	0.476 ± 0.0016	0.480
	5 v. 3	10.64 ± 0.86	1	0.258 ± 0.0029	0.271
			2	0.388 ± 0.0018	0.399
			4	0.415 ± 0.0038	0.431
	9 v. 4	12.57 ± 1.19	1	0.207 ± 0.0038	0.215
			2	0.345 ± 0.0026	0.349
			4	0.421 ± 0.00045	0.436
	8 v. 0	9.92 ± 1.39	1	0.231 ± 0.007	0.234
			2	0.434 ± 0.001	0.445
			4	0.460 ± 0.0015	0.471
Fashion MNIST	Dress v. T-shirt/top	17.96 ± 0.97	4	0.427 ± 0.0019	0.433
			8	0.422 ± 0.0031	0.428
			16	0.431 ± 0.0017	0.443
	Sneaker v. Sandal	16.3 ± 0.86	4	0.432 ± 0.002	0.451
			8	0.442 ± 0.0014	0.460
			16	0.449 ± 0.00097	0.468

Table 5.1: FID and ACD values obtained after training for 50 epochs without alterations to G's loss, for each of the used datasets and classifiers, averaged over three runs. ACD values obtained in the original test sets with each classifier are also reported.

5.2 Algorithm Behavior

As stated, the goal with GASTeN is to obtain synthetic data classified with low confidence by a target classifier, in order to try to understand the conditions under which the classifier is less reliable. In the studied case of binary classification, the goal is that the classifier's output on the data is 0.5. By looking for data classified as close as possible to 0.5, we are looking for

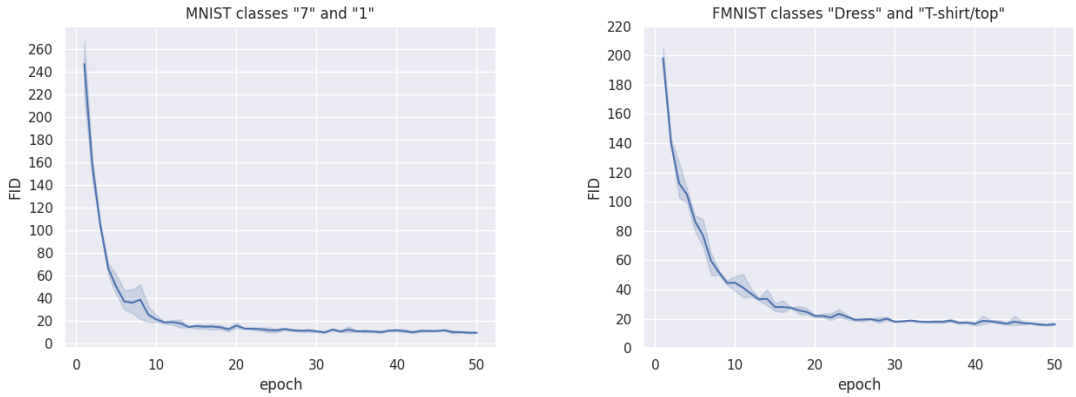


Figure 5.1: Progress of FID during training using classes "7" and "1" of the MNIST dataset, on the left, and using classes "Dress" and "T-shirt/top" of the FMNIST dataset, on the right. Plotted values are the mean over three runs, and the shadowed area ranges from the minimum and maximum values obtained.

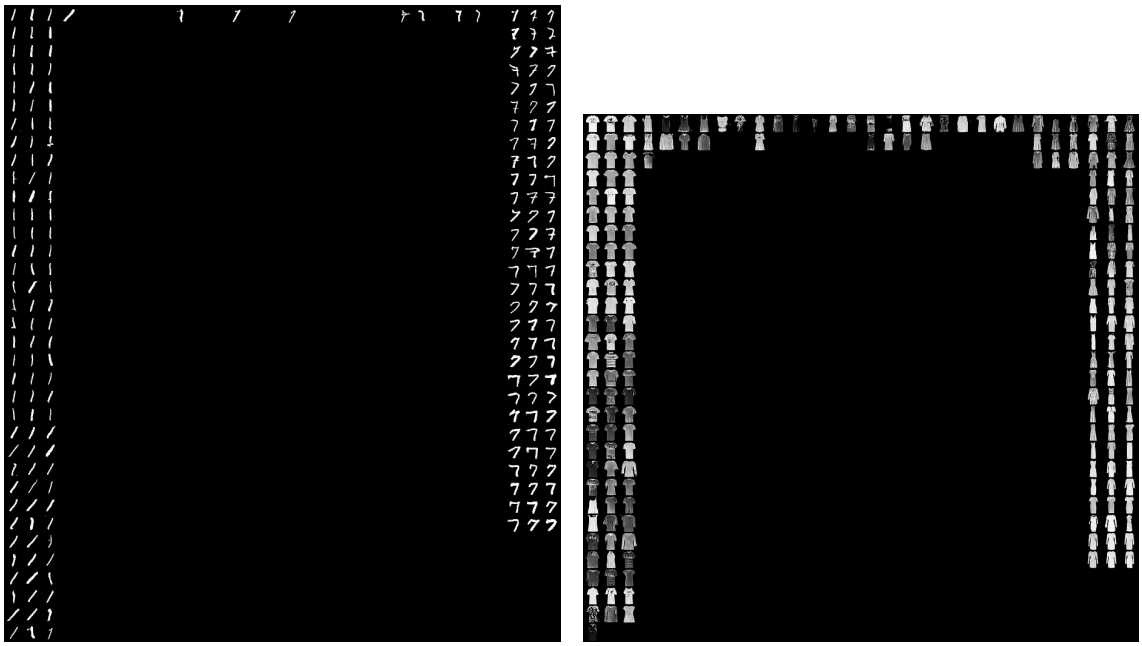


Figure 5.2: Synthesized samples for the "7" and "1" MNIST dataset, on the left, and for the "T-shirt/top" and "Dress" FMNIST dataset, on the right.

examples that sit in the frontier between two classes, and the classifier cannot clearly discern to which they belong. In order to understand how our method behaves, we start by addressing **O2** and **O3**. As such, this section discusses the algorithm's behavior for different α values, pre-training duration, and target classifier capacity. We support our analysis with the metrics collected during experiments (*cf.* Section 4.3.2, p. 30), the FID, and the ACD, which we aim to minimize. Additionally, we present relevant generated images.

5.2.1 Confusion Distance Weight

We start by addressing the classification of 7s and 1s on the MNIST dataset (MNIST-7v1), targetting the classifier with $nf = 2$, using 10 epochs of pre-training without modifying the loss function. The evolution of the collected metrics is depicted in Figure 5.4. Unsurprisingly, the choice of α influences the algorithm's behavior. The influence for α values lower than 25 is subtle, and is more noticeable for $\alpha = 30$. With $\alpha = 30$, the generator can confuse C by sacrificing the image's realness (the FID increases vastly), which is not aligned with our objectives. For lower values of α, the algorithm behaves similarly: G ends up not being able to fool C, and the FID keeps improving, which means that the loss term responsible for image realness ends up dominating the other. For $\alpha = 25$, despite behaving similarly to $\alpha = 30$ initially in some runs, after 40 epochs of training, the results resemble the results with lower α values. Figure 5.3 presents samples generated after the 40 epochs training with $\alpha = 30$ and $\alpha = 15$, using the visualization approach described in Section 4.3.2.3, p. 32. Failing to achieve a balanced trade-off is notorious, since with $\alpha = 30$ the images are noisy and unrealistic. Despite some resembling a realistic digit, the background is unlike the background found in the set of original images. With $\alpha = 15$, the images are realistic but are classified with confidence by the classifier.

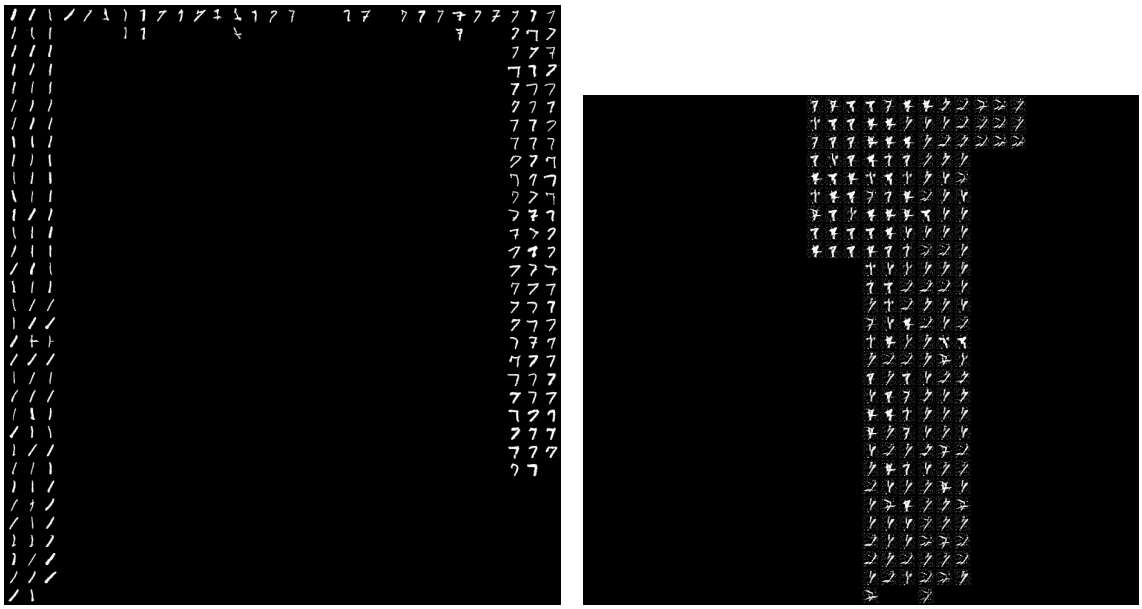
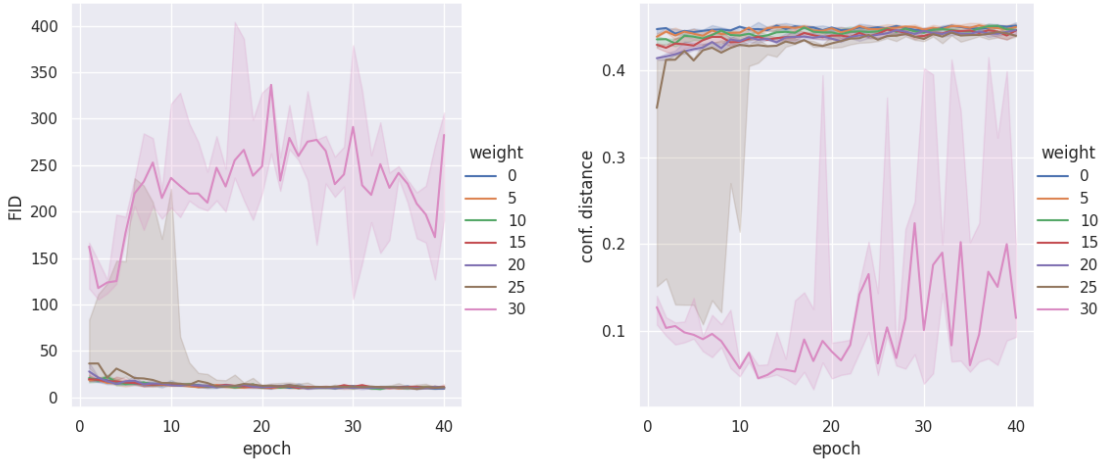


Figure 5.3: Samples of MNIST-7v1 synthesized by the G at the end of 40 epochs of training, with 10 epochs of pre-training, against the classifier with $nf = 2$. The α was 15 in the example on the left, and 30 in the example on the right.

Training of GASTeN against the classifier with $nf = 8$ for classes "Dress" and "T-shirt/top" of Fashion MNIST is depicted in Figure 5.5. Like in the previously described scenario, different results are obtained for different weight values. However, the differences are more subtle in this scenario. The higher the α, the lower the ACD is after each training epoch. With $\alpha = 0$, *i.e.*, the same as training a regular GAN, the ACD is the highest, as expected, since the loss term used to

minimize the ACD is effectively not used. Also, with higher α, the achieved FID increases but remains in similar magnitude values. Despite the tendency of achieving smaller average confusion distance values with higher α, that is not always the case. For instance, for the same dataset but targetting the classifier with $nf = 4$ and without pre-training, the run with $\alpha = 25$ achieves a lower ACD than the run with $\alpha = 30$, as seen in Figure 5.6. Note that runs on the same plot are executed with the same seed, thus the α parameter is the only difference between them.



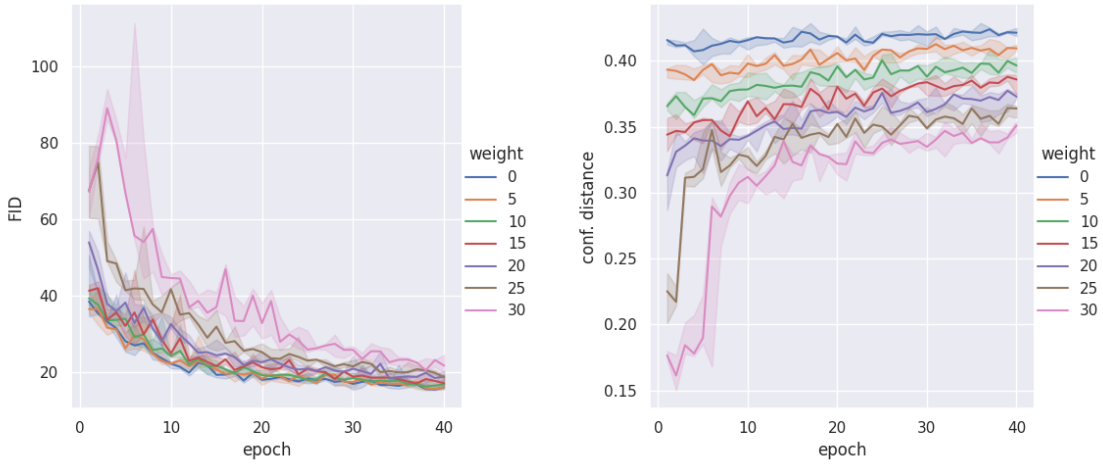
(a) FID evolution during training.

(b) Average confusion distance during training.

Figure 5.4: Metric evolution during training targetting CNN-2 for MNIST-7v1. Each line corresponds to a different α weight parameter, and the number of pre-train epochs is set to 10. Plotted values are the median over three runs, and the shadowed area ranges from the minimum and maximum values obtained.

5.2.2 Pre-train

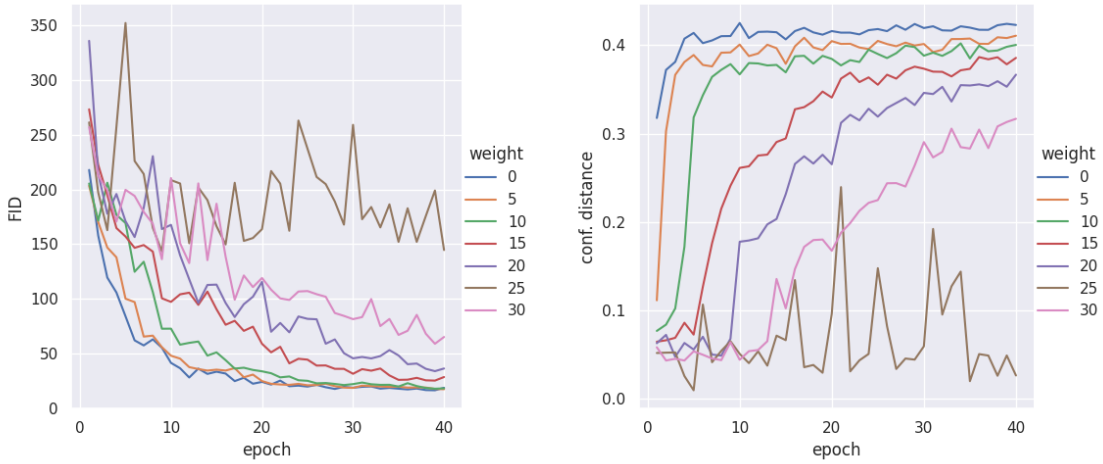
In the previously presented experiments, the number of pre-train epochs was set to 10. In order to understand the effect of pre-training, we examine the method's behavior for different amounts of pre-training. Figure 5.7 depicts training GASTeN for the same scenario as in Figure 5.4, *i.e.*, targetting a classifier for MNIST-7v1 with $nf = 2$. When the GAN is pre-trained for 5 epochs, the results are mostly similar to when pre-trained for 10 epochs: with a high enough weight ($\alpha = 30$), G can achieve low average confusion distance values at the cost of image fidelity. However, in one of the runs, the image realness objective dominated. This is unexpected since the GAN was trained solely on that objective for fewer epochs before introducing the modified loss. When not performing any pre-training, runs with low α (5, 10, and 15) still achieve low FIDs, with the smaller values converging faster. Values 15 and 20 had high variability between runs. With $\alpha \in \{25, 30\}$, high FIDs and low average confusion distance were obtained.



(a) FID evolution during training.

(b) Average confusion distance during training.

Figure 5.5: Metric evolution during training targetting CNN-8 for classes "Dress" and "T-shirt/top" of FMNIST. Each line corresponds to a different α parameter, and the number of pre-train epochs is set to 10. Plotted values are the median over three runs, and the shadowed area ranges from the minimum and maximum values obtained.



(a) FID evolution during training.

(b) Average confusion distance during training.

Figure 5.6: Metric evolution during training targetting CNN-4 for classes "Dress" and "T-shirt/top" of FMNIST. Plotted values are the median over three runs, and the shadowed area ranges from the minimum and maximum values obtained.

5.2.3 Target Classifier

GASTeN's training is adaptive to the target classifier, so the results achieved by GASTeN depend on the predictive performance of C. Figure 5.8 depicts training targetted at different classifiers ($nf \in 1,2,4$, with classifiers with higher nf having lower loss on the test sets) on MNIST-7v1,

Figure 5.7: Metric evolution during training targetting CNN-2 for the MNIST-7v1 dataset. Each line corresponds to a different α parameter, and each column in the plot grid corresponds to a different pre-train duration. Plotted values are the median over three runs, and the shadowed area ranges from the minimum and maximum values obtained.

using pre-train of 10 epochs. With the better classifier out of the three, the confusion distance has a negligible effect even with the highest α tested. For the classifier with the highest loss, all runs end with a low FID and high ACD, which is surprising. With a higher loss, intuition says finding examples that confuse the model should be easier. In contrast, with $\alpha = 30$, G achieves low ACD for the classifier with $nf = 2$ but not for the classifier with $nf = 1$.

5.3 Global Assessment

In Section 5.2, p. 37, we discussed the influence different parameters and classifiers have on algorithm runs. This section presents an analysis of the global GASTeN results, accross the multiple tested datasets, supported by the gathered metrics in Section 5.3.1, p. 42. In Section 5.3.2, p. 45, we visualize examples of synthesized data.

5.3.1 Quantitative Metrics

There are scenarios where the metrics oscillate significantly during training. Additionally, the number of epochs we perform in each run is fixed. Ideally, training should halt if it reaches a point where no improvement is achieved. However, as we work in a multi-objective scenario,

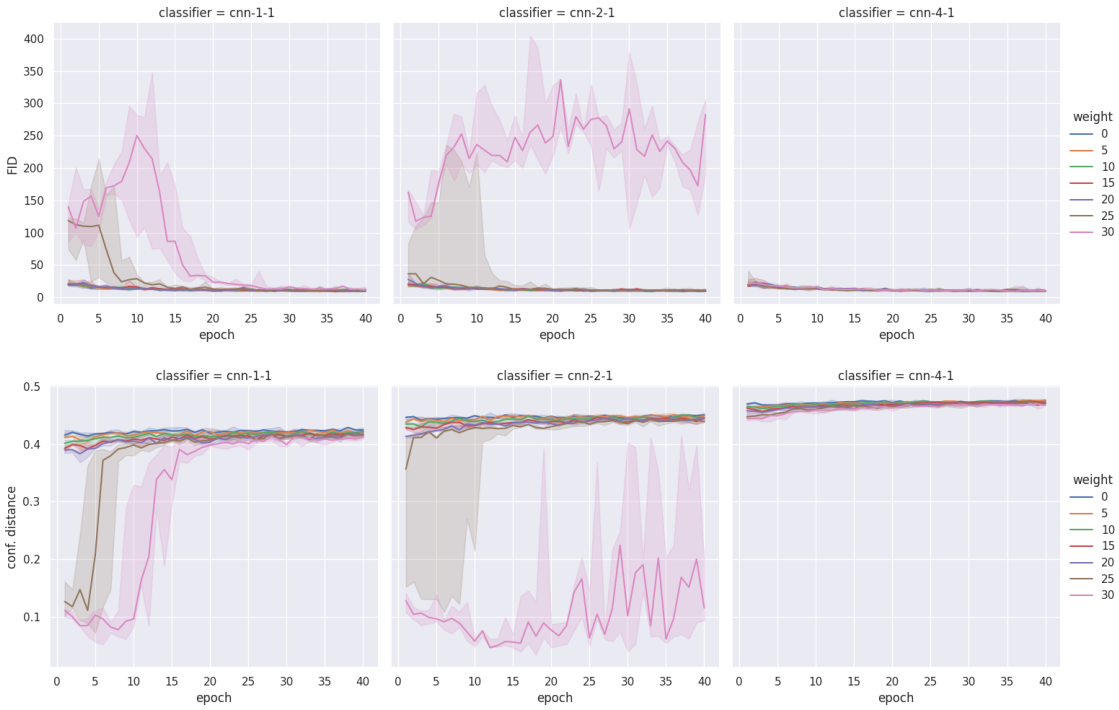


Figure 5.8: Metric evolution during training targetting several classifiers (each column corresponds to a different classifier) for the MNIST-7v1 dataset. Plots in the top row depict FID, while those in the bottom row depict ACD. Each color corresponds to a different α parameter, and the pre-train duration is set to 10 epochs. Plotted values are the median over three runs, and the shadowed area ranges from the minimum and maximum values obtained.

defining stopping criteria is not trivial, especially since both metrics have different magnitudes. The metrics also have some limitations (*cf.* Section 4.3.2, p. 30), and we specify training duration to a value large enough so that we can analyze the behavior of the algorithm. Since the number of training epochs is somewhat arbitrary, the most advantageous G is not necessarily the one obtained after the last one. Due to that, we analyze the algorithm globally by considering its performance spectrum after every epoch of the runs with different hyperparameters.

In order to better assess the results in terms of the measured metrics, we combine all metrics obtained with all hyperparameter combinations in a scatterplot, highlighting the points that are Pareto efficient, *i.e.*, where no objective can be improved without worsening the other. For the ongoing example discussed, MNIST-7v1, the results of one run are plotted in Figure 5.9. This figure illustrates the inability of GASTeN to achieve a desirable compromise between image quality and confusion distance for this exampleA hypothesis for this behavior is that, for this scenario, both optimization objectives are incompatible, and improving one worsens the other. Figure 5.10 depicts the same plot for the MNIST-5v3 dataset. In this case, however, a decrease in ACD does not come with such an abrupt increase in FID.

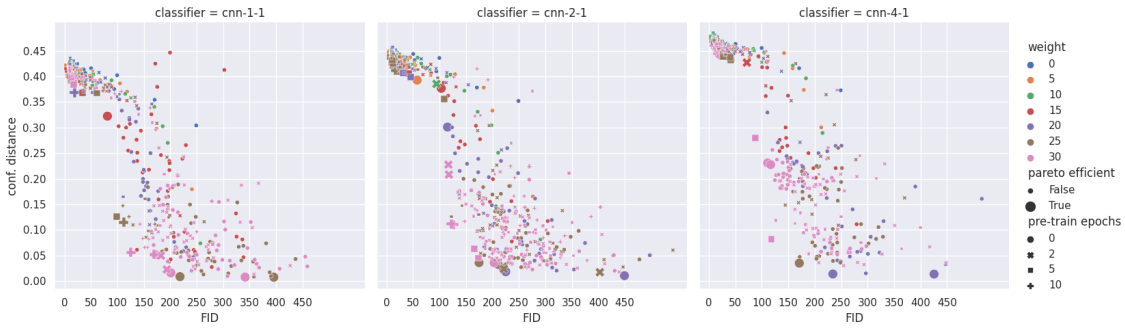


Figure 5.9: FID and ACD after every epoch, for all tested hyperparameters, for the MNIST-7v1 dataset. Each plot refers to a different classifier targetted.

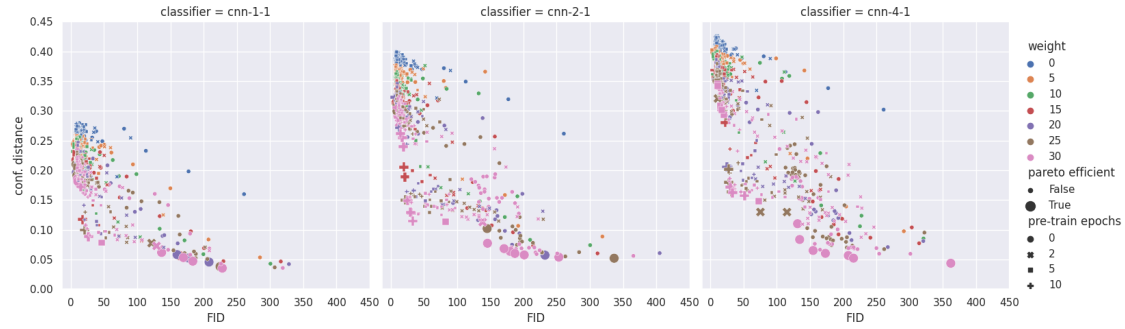


Figure 5.10: FID and ACD after every epoch, for all tested hyperparameters, for the MNIST-5v3 dataset. Each plot refers to a different classifier targetted.

Building on the notion of picking the most advantageous epoch from the runs accross all hyperparameters, Table 5.2 shows, for each classifier, the best FID such that, for the same G, ACD

is less or equal to a given threshold. The thresholds used range from 0.5 to 0.1. Note that setting the threshold to 0.5 effectively removes the constraint since that is the maximum ACD value. So, the FID value in the column that refers to that threshold is the best FID achieved for the given dataset. For clarification, the values in Table 5.2 differ from the values reported in Table 5.1 since the latter reports the result after the last epoch using $\alpha = 0$, while the former reports the minimum value across all epochs and hyperparameters. The minimum FID value varies depending on the classifier since some values may result from runs with α other than 0. This is unsurprising given previous observations that, especially for low values of α, the original GAN loss term still dominates and seems to be the only objective optimized.

From Table 5.2, we note that confusing higher capacity classifiers requires images that are more distant from the original data distribution, thus, with higher FID values. There are, similar to in Section 5.2.3, p. 41, cases where we obtain higher FIDs when targetting worse classifiers. Examples of such exceptions are MNIST-8v0 with a threshold equal to or less than 0.3 and MNIST-7v1 with a threshold of 0.3 and 0.1.

The results also show that reducing ACD will inevitably lead to high FID increases. Setting the threshold to 0.1, FID stays at values lower than 30 only for the MNIST-9v4 and MNIST-7v3 cases. That, however, happens only for the worst considered classifiers, which, without any modification to GANs, already achieve low ACD values (*cf.* Table 5.1, p. 37). Thus, it seems unlikely that a G can be obtained that almost always confuses the target classifier by generating realistic images that, according to the FID measure, are realistic. Despite not achieving arbitrarily low ACD values while maintaining a satisfactory FID, there are cases where there is some decrease in ACD compared to the images generated without a modified G loss function. Those cases, where FID values are within a 100% increase of the values in Table 5.1, for an ACD threshold smaller than the reported ACD, are highlighted in bold in Table 5.2.

It is also noteworthy to address the variability of the results obtained in different runs. Despite being negligible when the threshold is set to 0.5, there are cases where variability is quite significant, particularly for the results with lower confusion distance. For instance, the best MNIST-8v0 FID with ACD below 0.1, when targetting the best classifier, has a standard deviation of 51 over three runs. The best result had $FID = 48.5$ and $ACD = 0.09$ in one initialization, but a much worse best result with another initialization ($FID = 167.6$ and $ACD = 0.08$).

5.3.2 Visual Inspection

Besides analysing the collected metrics, it is also relevant to look into samples obtained by the image generation models trained. We visualize the images as explained in Section 4.3.2.3, p. 32, displaying 200 images simultaneously such that the position (the column where the image is) depicts the prediction of the targetted model on it.

Figure 5.11 presents images generated by running GASTeN against the three considered classifiers, exemplifying a scenario where the achieved generators have an acceptable FID and an ACD below 0.2. Figure 5.11a displays the samples for the classifier with $nf = 1$, for which the best G was the one at the end of epoch 38 of training with $\alpha = 30$ and without pre-training. The FID is

	Dataset	$C.nf$	0.5	0.4	0.3	0.2	0.1
MNIST	7 v. 1	1	8.53 ± 0.12	**11.9** ± 0.79	$78.5 \pm 17.$	$78.5 \pm 17.$	$106. \pm 20.$
		2	8.68 ± 0.11	40.0 ± 4.2	$101. \pm 14.$	$103. \pm 16.$	$127. \pm 27.$
		4	8.40 ± 0.084	88.6 ± 0.69	88.6 ± 0.69	$111. \pm 15.$	$123. \pm 3.4$
	8 v. 0	1	7.37 ± 0.47	7.37 ± 0.47	7.37 ± 0.47	**8.89** ± 0.35	$74.6 \pm 28.$
		2	7.50 ± 0.53	**9.63** ± 0.66	53.1 ± 3.7	53.1 ± 3.7	$136. \pm 17.$
		4	7.44 ± 0.48	**16.6** ± 3.0	41.2 ± 3.0	41.2 ± 3.0	$97.2 \pm 51.$
	5 v. 3	1	6.68 ± 0.098	6.68 ± 0.098	6.68 ± 0.098	**8.10** ± 0.36	26.1 ± 8.7
		2	6.63 ± 0.15	6.63 ± 0.15	**9.24** ± 0.19	**20.8** ± 0.90	$136. \pm 7.2$
		4	6.69 ± 0.14	**6.70** ± 0.13	**18.0** ± 2.1	25.8 ± 1.4	$136. \pm 5.7$
	9 v. 4	1	7.49 ± 0.036	7.49 ± 0.036	7.49 ± 0.036	**7.78** ± 0.31	**20.0** ± 0.65
		2	7.37 ± 0.38	7.37 ± 0.38	**8.12** ± 0.36	26.7 ± 3.4	$123. \pm 14.$
		4	7.38 ± 0.12	**7.80** ± 0.56	**22.9** ± 1.1	30.9 ± 0.84	$163. \pm 10.$
Fashion MNIST	Dress v. T-shirt/top	4	15.2 ± 0.23	**15.9** ± 0.38	49.0 ± 7.3	60.8 ± 4.1	$128. \pm 4.7$
		8	15.3 ± 0.085	**16.0** ± 0.57	46.3 ± 4.7	66.8 ± 5.2	$117. \pm 7.5$
		16	15.4 ± 0.075	**17.0** ± 0.40	52.2 ± 1.7	$64.5 \pm 12.$	$133. \pm 2.3$
	Sneaker v. Sandal	4	16.2 ± 0.28	**17.4** ± 0.95	56.9 ± 5.5	70.6 ± 5.6	$155. \pm 13.$
		8	16.2 ± 0.28	**19.6** ± 1.3	65.4 ± 7.9	$113. \pm 9.2$	$153. \pm 13.$
		16	16.3 ± 0.22	**21.9** ± 0.35	59.6 ± 2.3	$129. \pm 7.3$	$159. \pm 19.$

Table 5.2: Best FID obtained by a generator that has an ACD less or equal to a given threshold (values between 0.5 and 0.1 in the table header) for all datasets and classifiers. Results averaged over three runs.

of 8.51 and the ACD of 0.18. The best G for $nf = 2$ (Figure 5.11b) was achieved with 1 epoch of training with $\alpha = 20$ and 10 epochs of pre-training, and has an FID of 19.50 and ACD of 0.15. For $nf = 4$, the best G was also achieved with 1 epoch of training and 10 of pre-training, however with a α of 25. The FID is 25.7 and the ACD is 0.18. The computed quantitative metrics indicate that, as the classifiers have more capacity, the less real the images that fool it appear. For the classifier with $nf = 1$ (Figure 5.11a), digits such as the one highlighted in blue is an example of a perfectly unambiguous digit, which is classified correctly with low confidence by the classifier with worse performance. Despite the higher FID, several digits created by the G that targets the classifier with $nf = 4$ that are able to confuse it look realistic. The examples highlighted in blue in Figure 5.11b are interesting cases where the digits that confuse the classifier are somewhat a mix between a 3 and a 5 (the lower half of the digit could be part of both, but the horizontal dash in the upper part is extended both towards the left and right). Overall, the generated digits look less perfectly drawn and noisier, which is further aggravated for $nf = 4$ (the images highlighted by a red square in Figure 5.11c are examples of it). The examples highlighted in green are also interesting since they feature the lower half of the digit that could belong to a 5 or a 3. However, the upper dash is not the bottom part, making the image ambiguous (the digit could be either a three or five depending if the connecting stroke was drawn on the right or left, respectively). We argue that such examples

could prove helpful in analyzing the classifier.

Figure 5.12 shows samples of the MNIST-8v0 dataset generated by a G with an FID of 40.8 and an ACD of 0.11 for the classifier with $nf = 4$ (significantly lower than the ACD on the test set, 0.471). Note that such decrease in ACD was only possible by sacrificing the FID, which is 9.92 when the GAN is used without modifications. The G is a result of 10 epochs of pre-training and 10 epochs of training with $\alpha = 30$. In this case, the goal of generating images that are closer to the border is achieved. Out of the 200 samples, only one is classified between 0 and 0.2, and only one is classified between 0.8 and 1, while most generated images are classified between 0.4 and 0.6, which is desired. As suggested by the high FID, some of these images look less like noisy digits (for instance, those highlighted in red). However, most of the examples are interesting, such as those highlighted in green, which are not clearly identifiable as 0s or 8s, but look plausible and could, for instance, be digits drawn in a hurry.

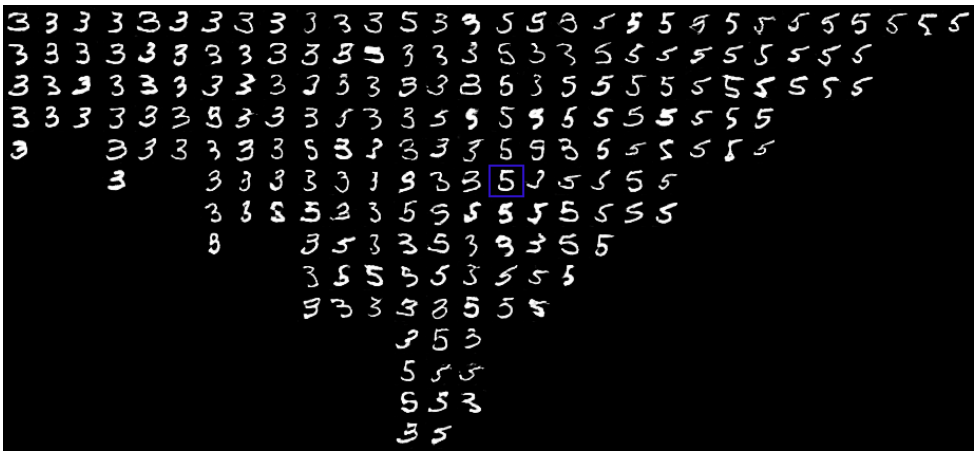
Figure 5.13 depicts samples of MNIST-9v4 from a G with $FID = 30.9$ and $ACD = 0.15$ for a classifier with $nf = 4$, obtained after 2 epochs of train with $\alpha = 30$ and 10 epochs of pre-train. As in the example of the 8s and 0s, the generator is able to create images that are mostly classified with low confidence. Of the images that are confusing to the classifier, some are unrealistic and have noise-like marks disconnected from the actual digit (as the examples highlighted in red), while others have characteristics, which, intuitively, makes them confusing, even for humans. The examples highlighted in green look like unfinished 9s, *i.e.*, 9s with the upper part disconnected, similar to hand drawn 4s.

Figure 5.14 shows an example for the Fashion MNIST dataset, with classes "Dress" and "T-shirt/top". The G whose generated images are depicted achieved ACD of 0.20 but with a high FID of 52.4. The images are notoriously unrealistic. Additionally, the images are quite small for the complexity of the represented objects, making it difficult to analyse them.

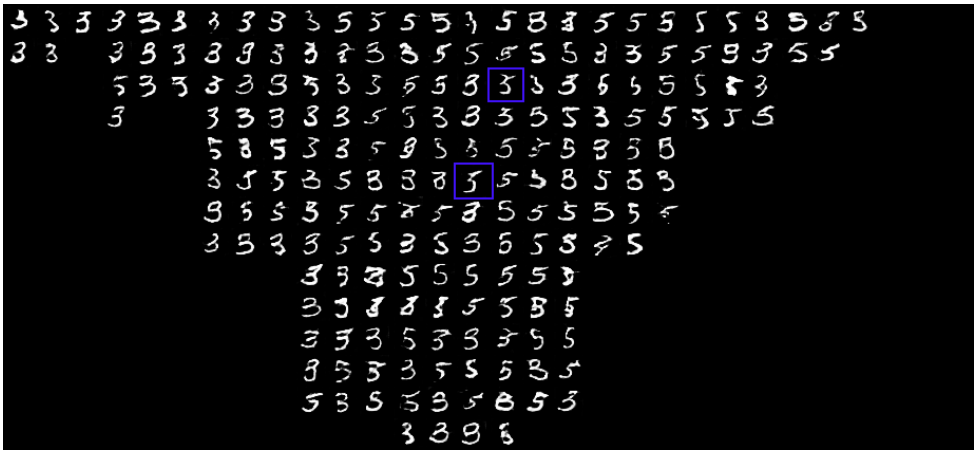
5.4 Remarks

From the analysis, we conclude that, with the chosen hyperparameters, it is difficult to optimize for both objectives simultaneously. Over a single run, training converges to one loss term dominating the other most of the time. With a small α, the generated data becomes very realistic without a significant reduction of the ACD. However, in the first epochs after α is introduced, there are moments before training stabilizes at low FID values and high ACD where the metrics have different values. For higher values of α, training is quite unpredictable, and the ACD and FID values oscillate significantly.

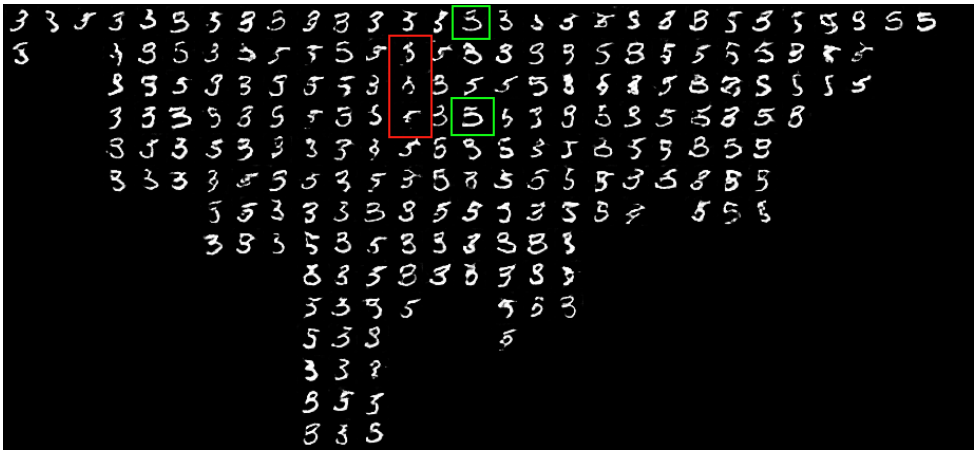
However, despite the described difficulty, and by taking advantage of the generator's state after epochs before the metrics stabilize, some generators have been found that achieve the desired purpose. These generators can synthesize images that are, on average, closer to the 0.5 decision threshold than the images in the test set, and the FID is not significantly increased when compared to the FID achieved by the same GAN architecture without modifications. By doing this, we show that it is within the capabilities of GASTeN to fulfill our goal. However, selecting the best



(a) Targetting the classifier with $nf = 1$.



(b) Targetting the classifier with $nf = 2$.



(c) Targetting the classifier with $nf = 4$.

Figure 5.11: Samples of MNIST-5v3 synthesized by the G with the best FID among those with $ACD \leq 0.2$ for each of the classifiers considered. Some unrealistic digits are highlighted in red, and some ambiguous digits are highlighted in green. Examples highlighted in blue are commented on in the main text.

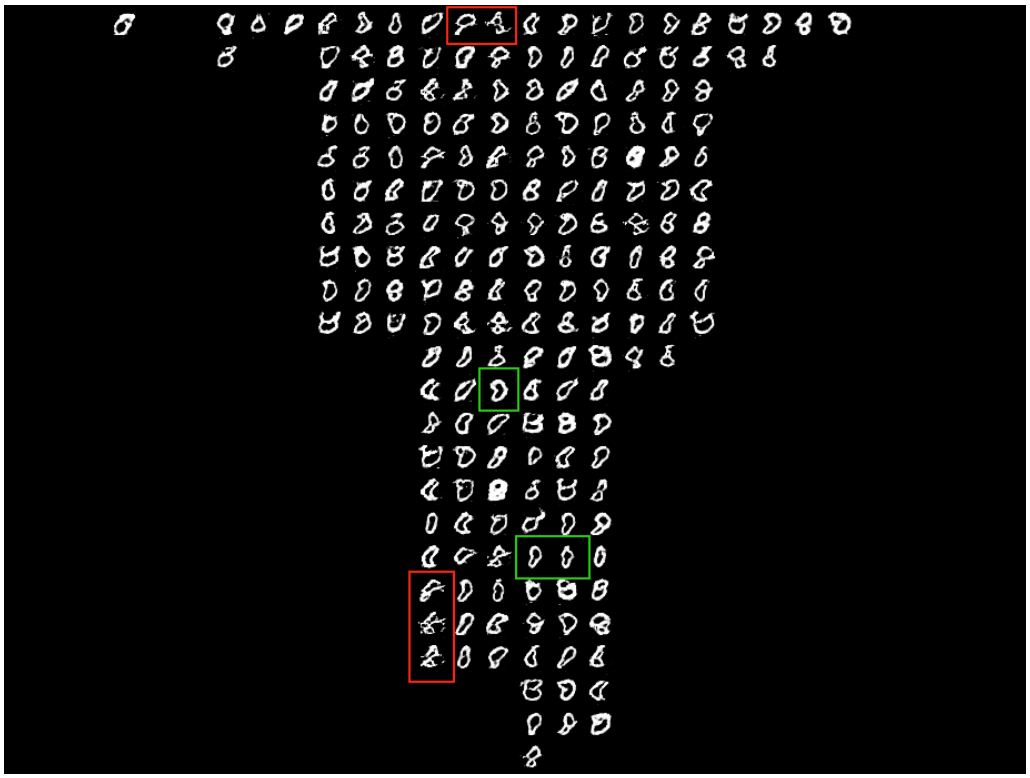


Figure 5.12: Samples of MNIST-8v0 synthesized by the G with the best FID among those with $ACD \leq 0.2$ for the strongest classifier ($nf = 4$).

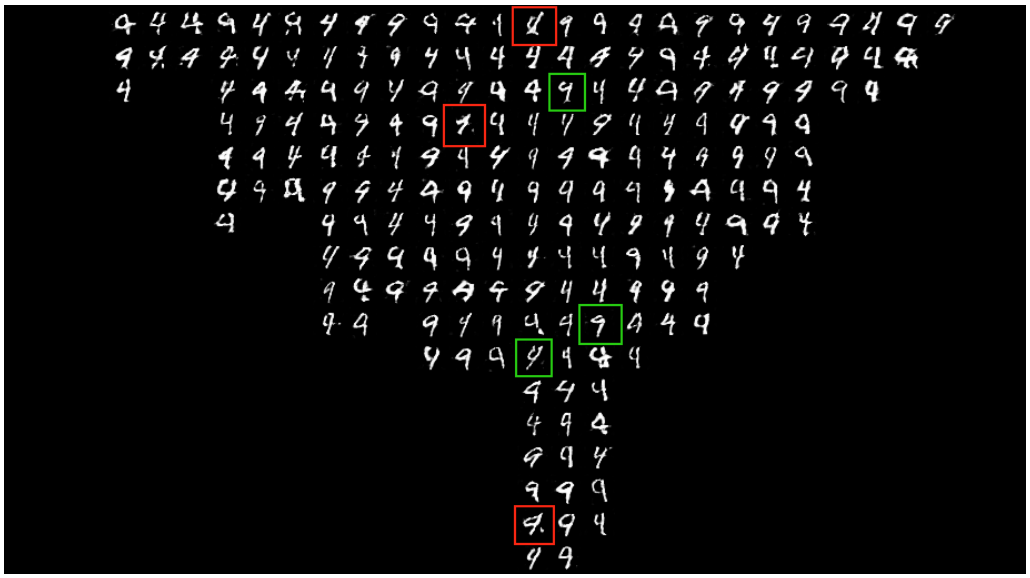


Figure 5.13: Samples of MNIST-9v4 synthesized by the G with the best FID among those with $ACD \leq 0.2$ for the strongest classifier ($nf = 4$).

generator from runs with different hyperparameter combinations can be prohibitive due to long training times.

Figure 5.14: Samples of dresses and t-shirts/tops synthesized by the best generator with $ACD \leq 0.2$.

5.5 Threats to Validity

We identify some limitations that may hinder the conclusions drawn from the empirical study of GASTeN. Some limitations concern the conducted experiments and may affect the conclusions drawn from this study, while others concern the evaluation method employed.

Regarding the experiments, the datasets utilized in the reported experiments consist of grayscale images of small dimensions (28×28). Due to their simplistic nature, we cannot generalize how the algorithm will behave with other datasets, particularly those with colorized images with larger dimensions. Also, in recent years, many GAN variants have been proposed in the literature (*cf.* Section 2.2.1, p. 8). In addition, training GANs for image generation usually requires careful hyperparameter selection. Due to time and computation hardware constraints, in this work, we instantiate GASTeN with only one architecture with a configuration that achieves satisfactory results in realistic image generation in the studied datasets. As such, we cannot conclude anything regarding the behavior of GASTeN when coupled with different GAN architectures. Again due to time constraints, the number of hyperparameter combinations tested may not fully represent all the possibilities.

Regarding the evaluation methodology, we measured the quality of the generated images using a commonly used metric in the GAN literature, the FID. As previously mentioned (*cf.* Section 4.3.2, p. 30), the metric has some limitations. Additionally, we use the whole set of generated images to compute the FID, not only the images with the characteristics that interest us. As such, an increase in FID, even if it stays within acceptable values, could be due to the images with the desired characteristics. An alternative could be using only the desired images to compute the FID. We tackle this limitation by also inspecting the generated images visually. Despite that, visual inspection of images performed by us could be biased, and our evaluation could benefit from a human study to evaluate if the images generated by our approach are realistic.

Chapter 6

Conclusions

As ML and DL thread into domains that can directly affect people's lives, there is a need for techniques that allow their responsible usage. Model cards are an approach for documenting the model regarding several aspects of its performance and intended use-case. Motivated by improving the creation process of model cards and better understanding the behavior of classifiers, we propose a method for generating data classified with low confidence by a classifier, from which information regarding the model's performance could be extracted.

In this dissertation, we focus on the data generation aspect of our stated objective and propose an approach that leverages GANs, a class of generative models that has achieved great success in recent years. Since the prominent application cases of GANs are in CV, we empirically evaluate our general approach for CV tasks. Our proposal, dubbed GASTeN, consists of modifying the original GAN framework by introducing a new term added to the loss of the generator, which we refer to as confusion distance. Two hyperparameters are introduced in our approach: a weight α that is multiplied by the new loss term and the number of pre-train epochs of the GAN without modifications.

We empirically experiment with our approach for the binary classification scenario on subsets of standard CV testing datasets, MNIST and Fashion MNIST, with several hyperparameter combinations and against classifiers trained to have varying performances. We evaluate it using two quantitative metrics, one for image realness (the FID score) and one to measure if our objective is being achieved (an average of the confusion distance). We compare the achieved results against the FID achieved by the same GAN architecture without modifications and the ACD achieved by the classifier on the original test set. GASTeN fails to achieve satisfactory FIDs with low average confusion distance consistently, and trt aining either oscillates or stabilizes in a scenario where one loss term dominates the other. However, by analyzing several runs of GASTeN with different hyperparameters and considering the generator's state after every epoch, we find examples where GASTeN achieves the desired results, showing that it is possible to achieve our goal using GASTeN.

6.1 Future Work

Regarding data generation, we plan to compare our proposal against adaptations (to generate data that is classified in the frontier instead of misclassified) of approaches present in the adversarial attack literature. We also intend to study our approach further, introducing an automatic way to select hyperparameters and experimenting with a mechanism to dynamically select the weight α during training. We also intend to experiment with different confusion distance functions, both for the same goal of finding images classified with low confidence and for other purposes. A natural future step for our approach would be to evaluate it with a new confusion distance that works for the multiclass classification scenario. Future work should also include scaling the experiments to different GAN architectures, more complex datasets, and different classifiers.

Additionally, several directions for analyzing the generated data can be pursued. A natural next step would be to study how the insights gathered from the complex images can be compiled to document a classifier, for instance, in a model card [41]. Interpretability techniques could also be applied to the generated images to get more insights into the model's behavior. Also, other types of analysis could be conducted, for instance, to directly compare two different models and the cases where their behavior, in terms of prediction confidence, is similar or differs.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. volume 70, pages 214–223. PMLR, 6 2017.

[3] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 6 2020.

[4] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *CoRR*, abs/1703.09387, 2017.

[5] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[6] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2 2019.

[7] Ali Borji. Pros and cons of gan evaluation measures: New developments. *Computer Vision and Image Understanding*, 215:103329, 1 2022.

[8] Michael Buhrmester, Tracy Kwang, and Samuel D. Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, 6:3–5, 1 2011.

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *Proceedings - IEEE Symposium on Security and Privacy*, pages 39–57, 8 2016.

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.

[11] Yunjey Choi, Minje Choi, Munyoung Kim, Jung Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 11 2017.

[12] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[13] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 99–108, New York, NY, USA, 2004. Association for Computing Machinery.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. pages 248–255, 3 2010.

[16] Isaac Dunn, Hadrien Pouget, Tom Melham, and Daniel Kroening. Adaptive generation of unrestricted adversarial inputs. 5 2019.

[17] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling CNNs with simple transformations, 2019.

[18] Maurice Fréchet. Sur la distance de deux lois de probabilité. *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences*, 244(6):689–692, 1957.

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

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. volume 27. Curran Associates, Inc., 2014.

[21] Ian J Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[23] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. volume 30. Curran Associates, Inc., 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 6 2016.

[25] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. Attgan: Facial attribute editing by only changing what you want. *IEEE Transactions on Image Processing*, 28:5464–5478, 11 2017.

[26] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. volume 30. Curran Associates, Inc., 2017.

[27] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[28] Ameya Joshi, Amitangshu Mukherjee, Soumik Sarkar, and Chinmay Hegde. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October:4772–4782, 4 2019.

[29] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[31] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, 7 2016.

[32] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 11 2016.

[33] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. *Advances in Neural Information Processing Systems*, 2017-December:5968–5977, 6 2017.

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

[35] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[36] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 2004 60:2*, 60:91–110, 11 2004.

[37] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. volume 31. Curran Associates, Inc., 2018.

[38] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 6 2017.

[39] Masoud Mahdianpari, Bahram Salehi, Mohammad Rezaee, Fariba Mohammadimanesh, and Yun Zhang. Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery. *Remote Sensing*, 10:1119, 07 2018.

[40] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[41] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Deborah Raji, and Timnit Gebru. Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019.

[42] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. volume 70, pages 2642–2651. PMLR, 6 2017.

[43] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pages 582–597, 8 2016.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[45] Haonan Qiu, Chaowei Xiao, Lei Yang, Xinchen Yan, Honglak Lee, and Bo Li. Semanticadv: Generating adversarial examples via attribute-conditioned image editing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12359 LNCS:19–37, 2020.

[46] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 11 2015.

[47] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prenctice Hall Press, USA, 3rd edition, 2009.

[48] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. volume 29. Curran Associates, Inc., 2016.

[49] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.2.1.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[51] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. volume 31. Curran Associates, Inc., 2018.

[52] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. 6 2016.

[53] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 12 2013.

[54] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[55] Xiang Tao, Liu Hangcheng, Guo Shangwei, Gan Yan, and Liao Xiaofeng. Egm: An efficient generative model for unrestricted adversarial examples. *ACM Transactions on Sensor Networks (TOSN)*, 10 2021.

[56] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *CoRR*, abs/1511.01844, 2016.

[57] Chih-Fong Tsai, A Hernandez, P Kokol, J Wang, and S Zhu. Bag-of-words representation in image annotation: A review. *International Scholarly Research Network ISRN Artificial Intelligence*, 2012, 2012.

[58] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[59] Abdul Waheed, Muskan Goyal, Deepak Gupta, Ashish Khanna, Fadi Al-Turjman, and Plácido Pinheiro. Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection. *IEEE Access*, PP:1–1, 05 2020.

[60] Xiaosen Wang, Kun He, and John E Hopcroft. At-gan: A generative attack model for adversarial transferring on generative adversarial nets. *CoRR*, abs/1904.07793, 2019.

[61] Zhengwei Wang, Qi She, and Tomas E Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(2):1–38, 2021.

[62] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[63] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[64] Chaowei Xiao, Bo Li, Jun Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:3905–3911, 1 2018.

[65] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[66] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[67] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 10 2017.

Appendix A

Experiment Configuration

This appendix details the configuration of the experiments run via presenting an example file. Listing 1 (p. 59) is an example of a configuration file. In addition to specifying the dataset, GASTeN hyperparameters and the classifiers to target, it allows specifying other hyperparameters regarding training, the GAN architecture, and the optimizer. The configuration file also provides information regarding directories for reading data and storing checkpoints. Some fields are optional. For instance, it is not required to specify seeds for the runs, in which case a random seed will be used and stored with other experiment artifacts. It is, however, useful in order to reproduce experiment runs.

```yaml
project: "gasten"
name: "mnist-5v3"
out-dir: "/out/"
data-dir: "/data/"
test-noise: "/data/z/z_2000_64"
fixed-noise: 200
device: cuda:0
num-workers: 6
fid-stats-path: "/data/fid-stats/stats.inception.mnist.5v3.npz"

num-runs: 1
step-1-seeds:
    - 6248
step-2-seeds:
    - 4123

dataset:
  name: "mnist"
  binary:
    pos: 5
    neg: 3

model:
  z_dim: 64
  architecture:
    name: "dcgan"
    g_filter_dim: 64
    d_filter_dim: 64
    g_num_blocks: 3
    d_num_blocks: 3
  loss:
    name: "ns"

optimizer:
  lr: 0.0002
  beta1: 0.5
  beta2: 0.999

train:
  step-1:
    epochs: 10
    batch-size: 64
  step-2:
    epochs: 40
    step-1-epochs:
      - 0
      - 5
    batch-size: 64
    disc-iters: 1
    classifier:
      - "/classifiers/mnist.5v3/cnn-2-1.86706"
      - "/classifiers/mnist.5v3/cnn-4-1.92101"
    weight:
      - 5
      - 10
```

Listing 1: Example YAML configuration file.

Appendix B

Full GASTeN Metrics

This appendix presents all metrics gathered in the conducted experiments. The shown plots contain the FID and average confusion distance metrics computed after each training epoch for all datasets, hyperparameter configurations, and classifiers described in Section 4.3, p. 29. Since we are optimizing for two objectives, points that exhibit Pareto optimality are highlighted by being represented by a bigger symbol. The classifier naming convention used is cnn-*x*-*y*, where *x* is the nf parameter (*cf.* Section 4.2.2, p. 29) and *y* is the number of training epochs. Section B.1, p. 60 contains results on the MNIST dataset and Section B.2, p. 93 contains results on the Fashion MNIST dataset.

B.1 MNIST

B.1.1 7 vs. 1

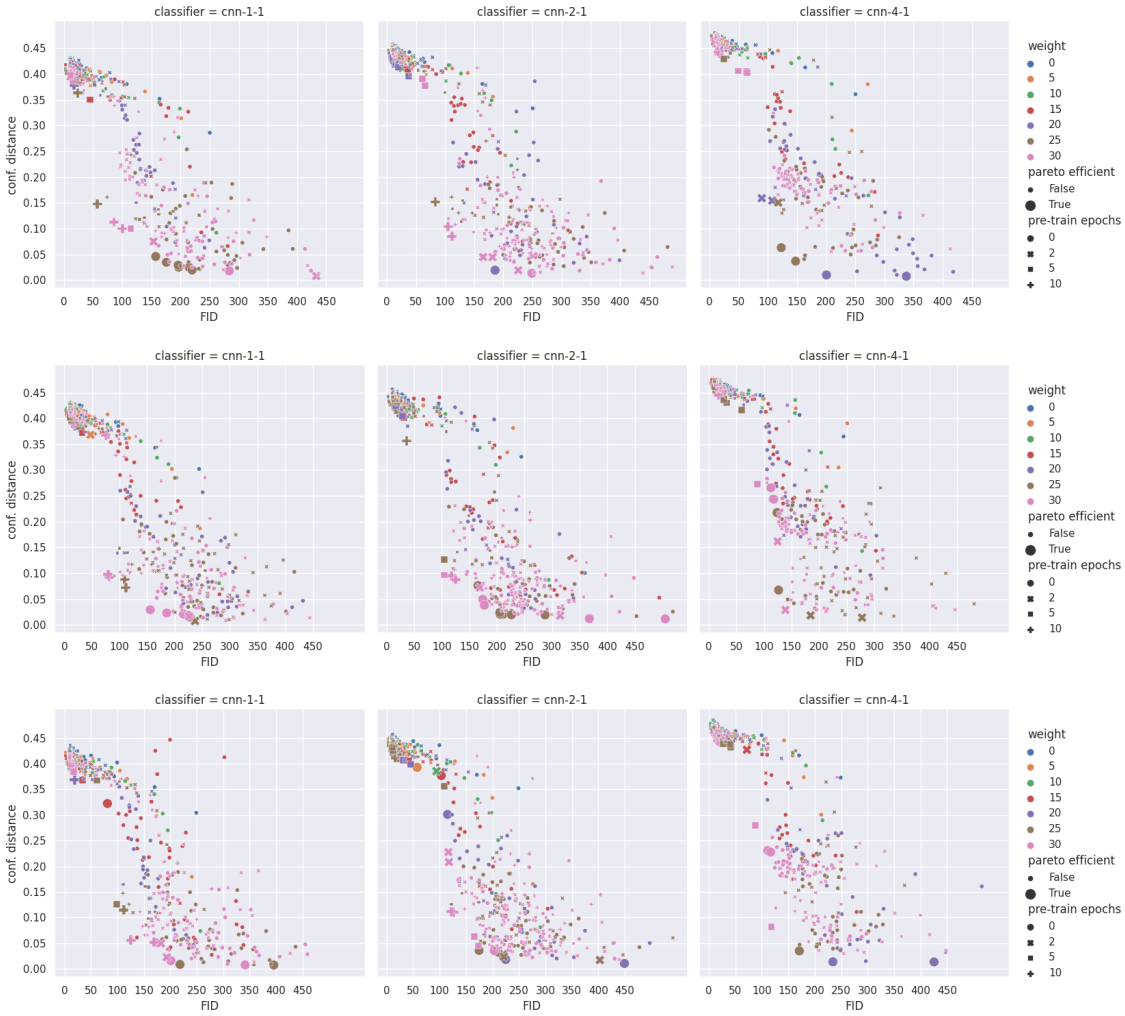


Figure B.1: All metrics gathered using the dataset consisting of examples of classes 7 and 1 of the MNIST dataset.

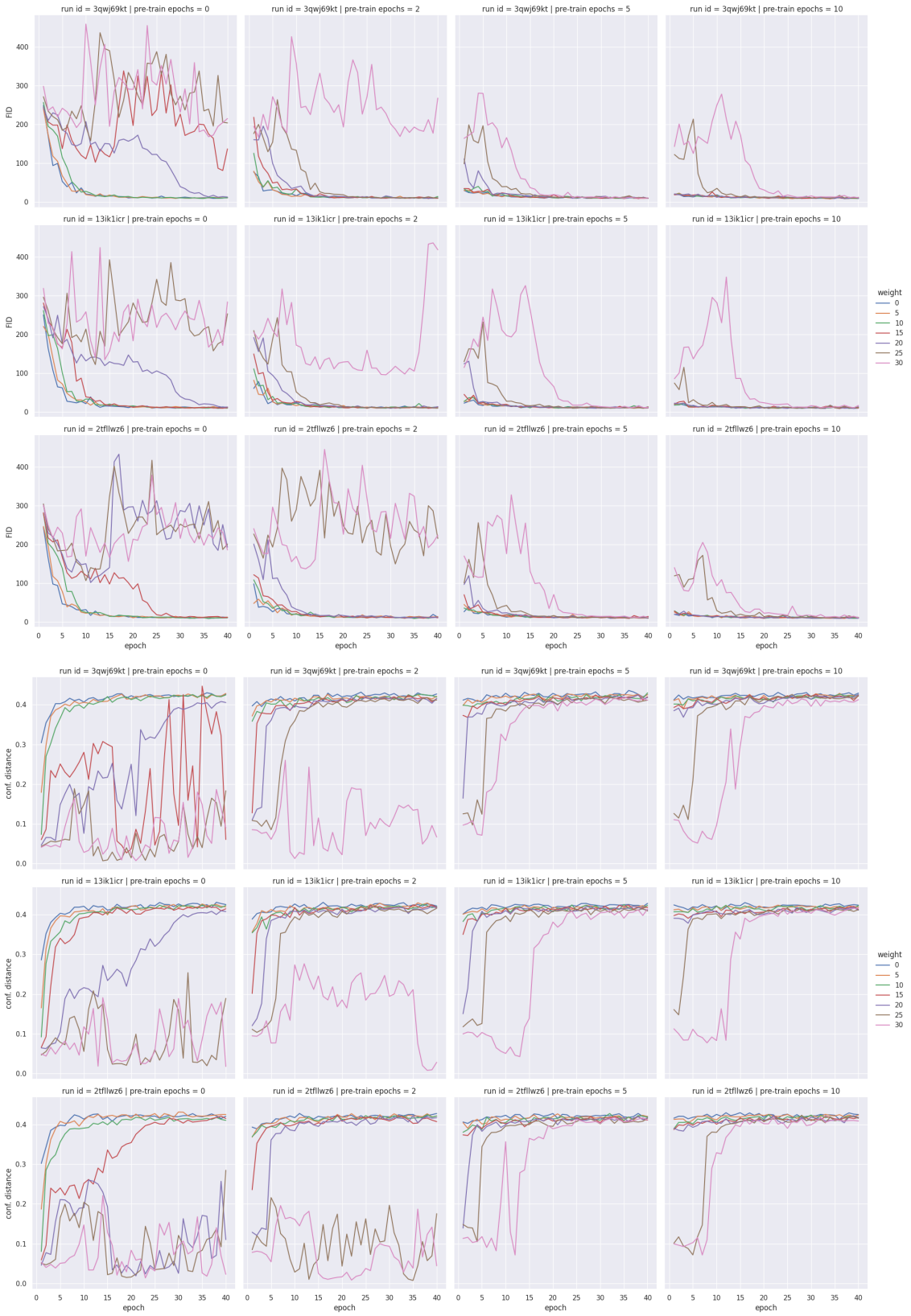


Figure B.2: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a classifier with $nf = 1$.

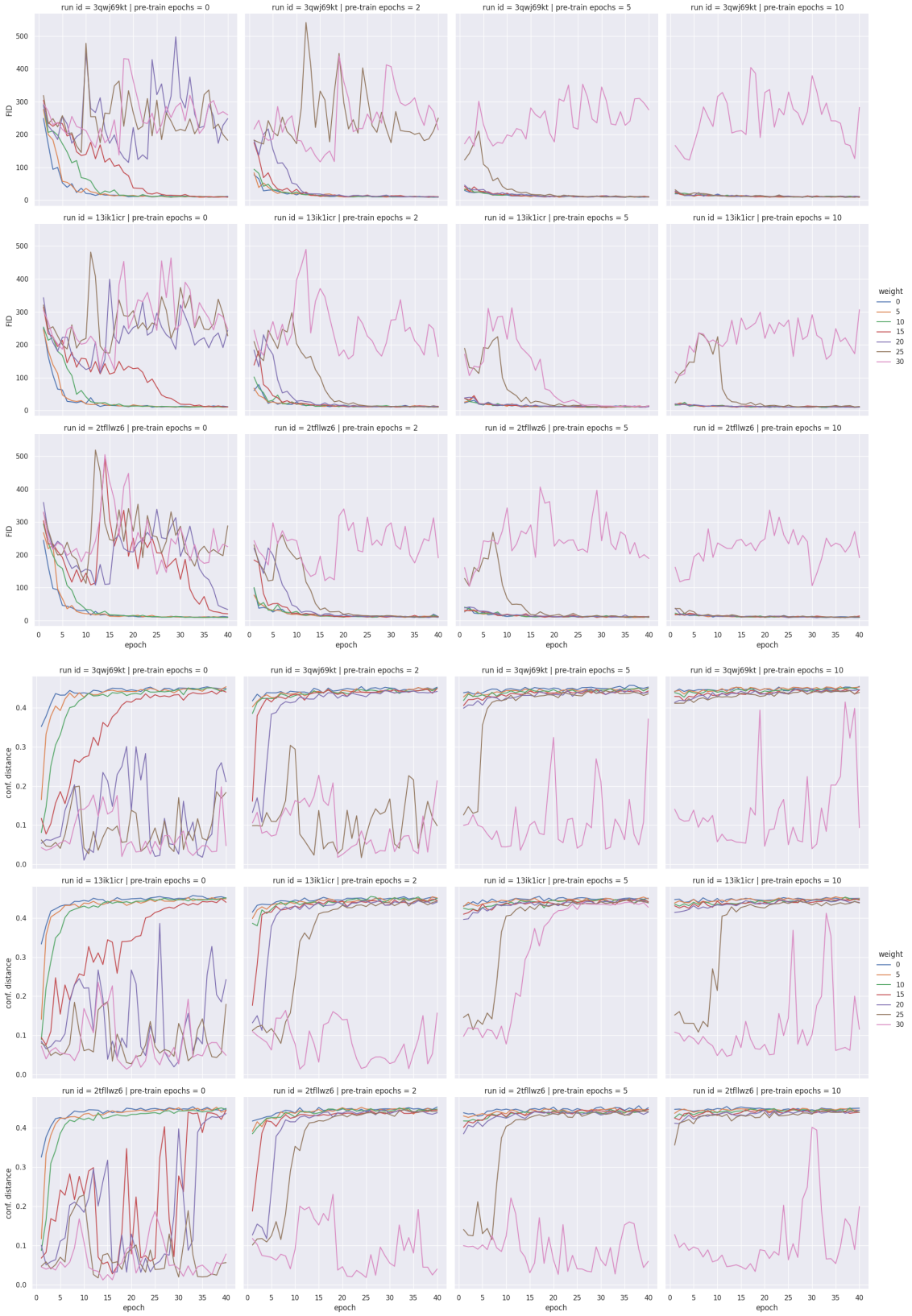


Figure B.3: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a classifier with $nf = 2$.

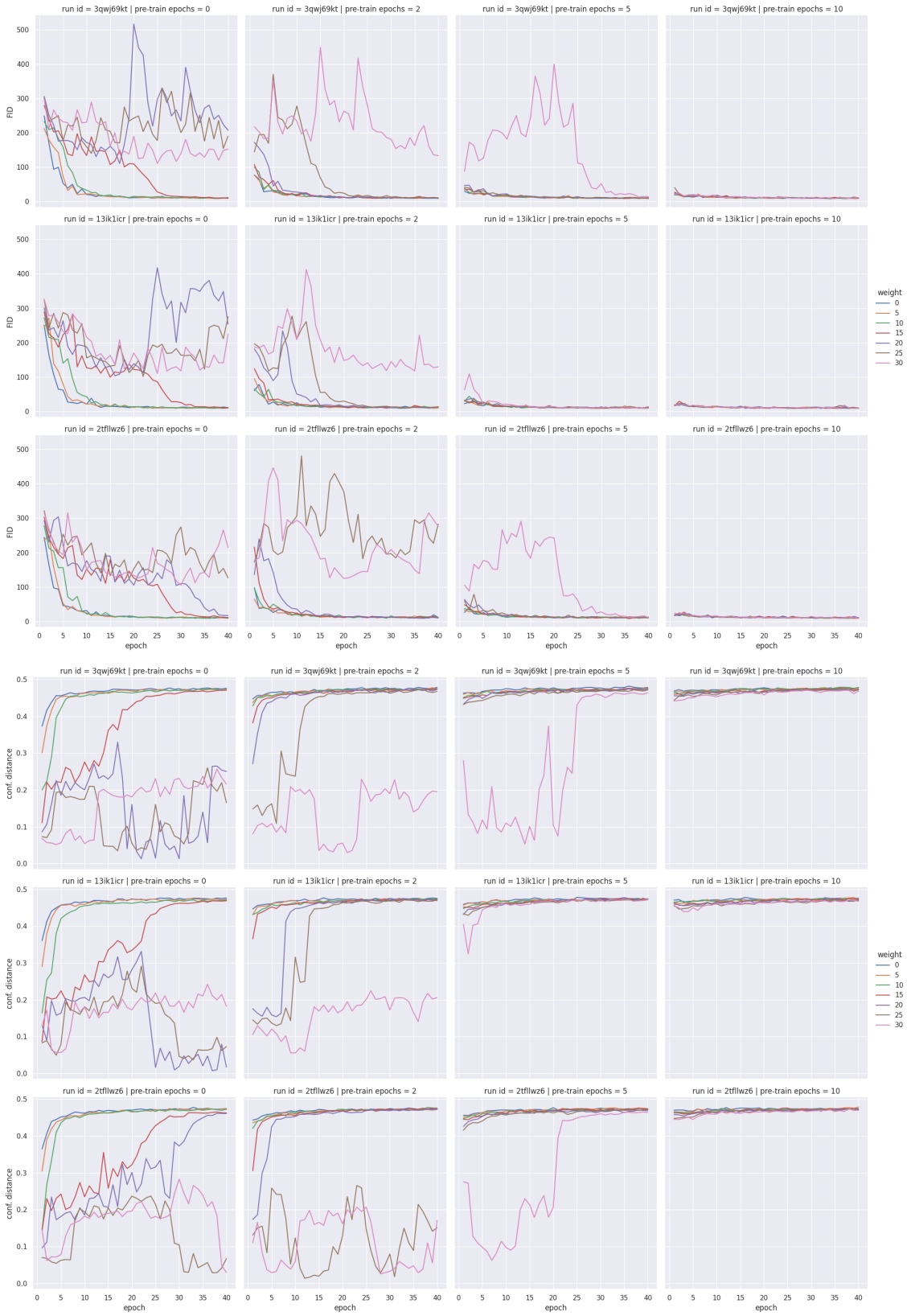


Figure B.4: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a classifier with $nf = 4$.

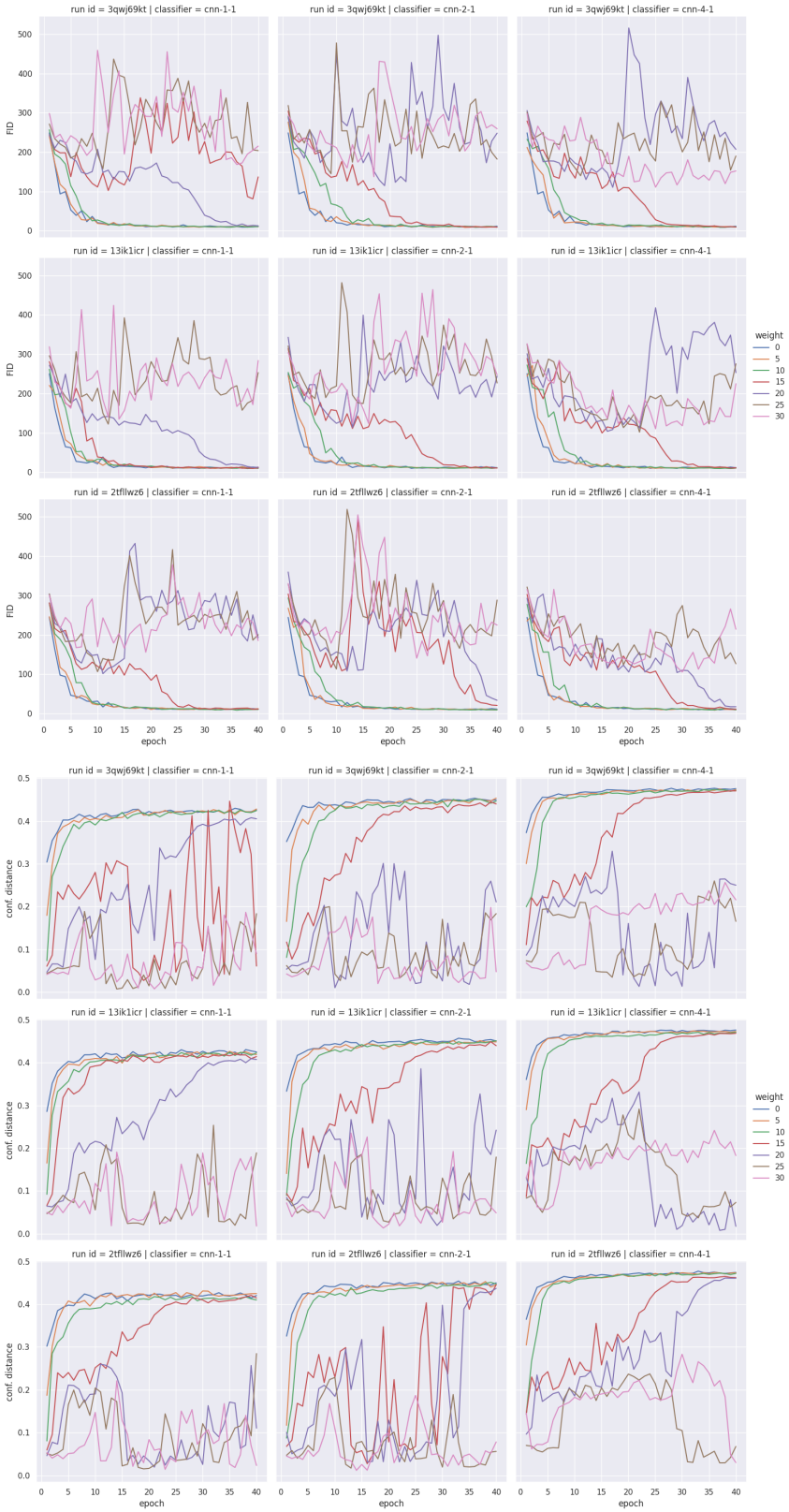


Figure B.5: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a number of pre-train epochs of 0.

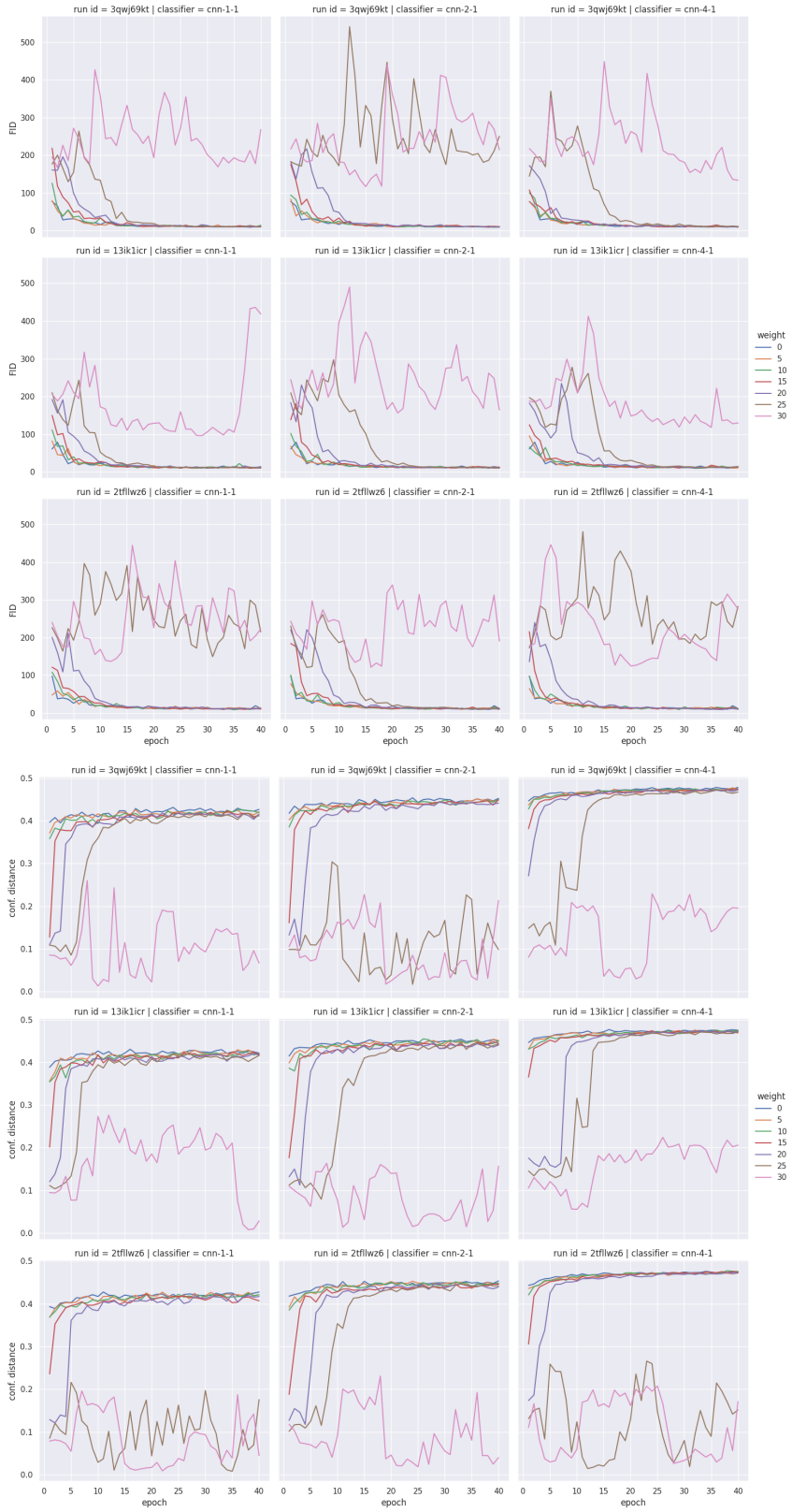


Figure B.6: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a number of pre-train epochs of 2.

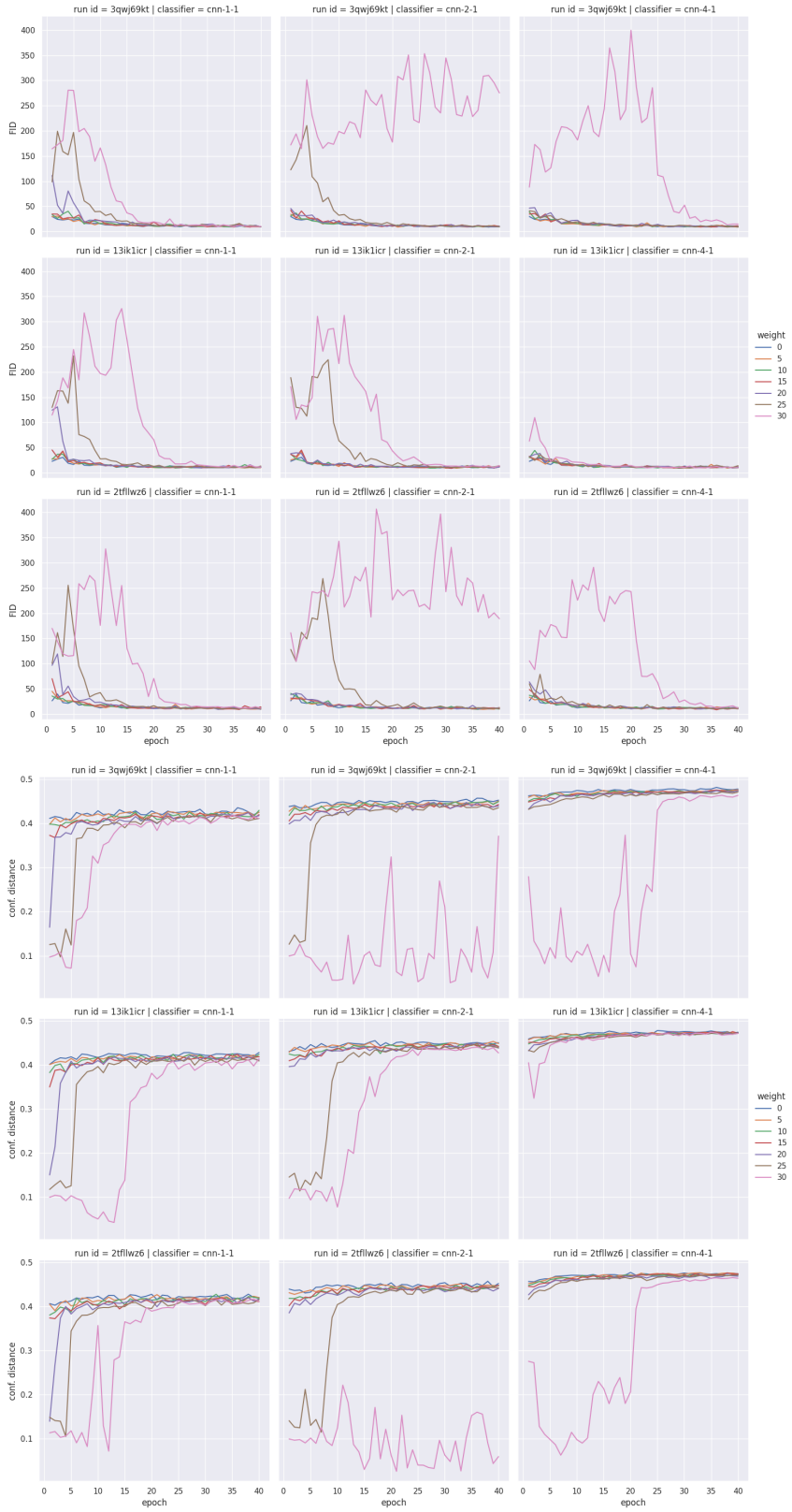


Figure B.7: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a number of pre-train epochs of 5.

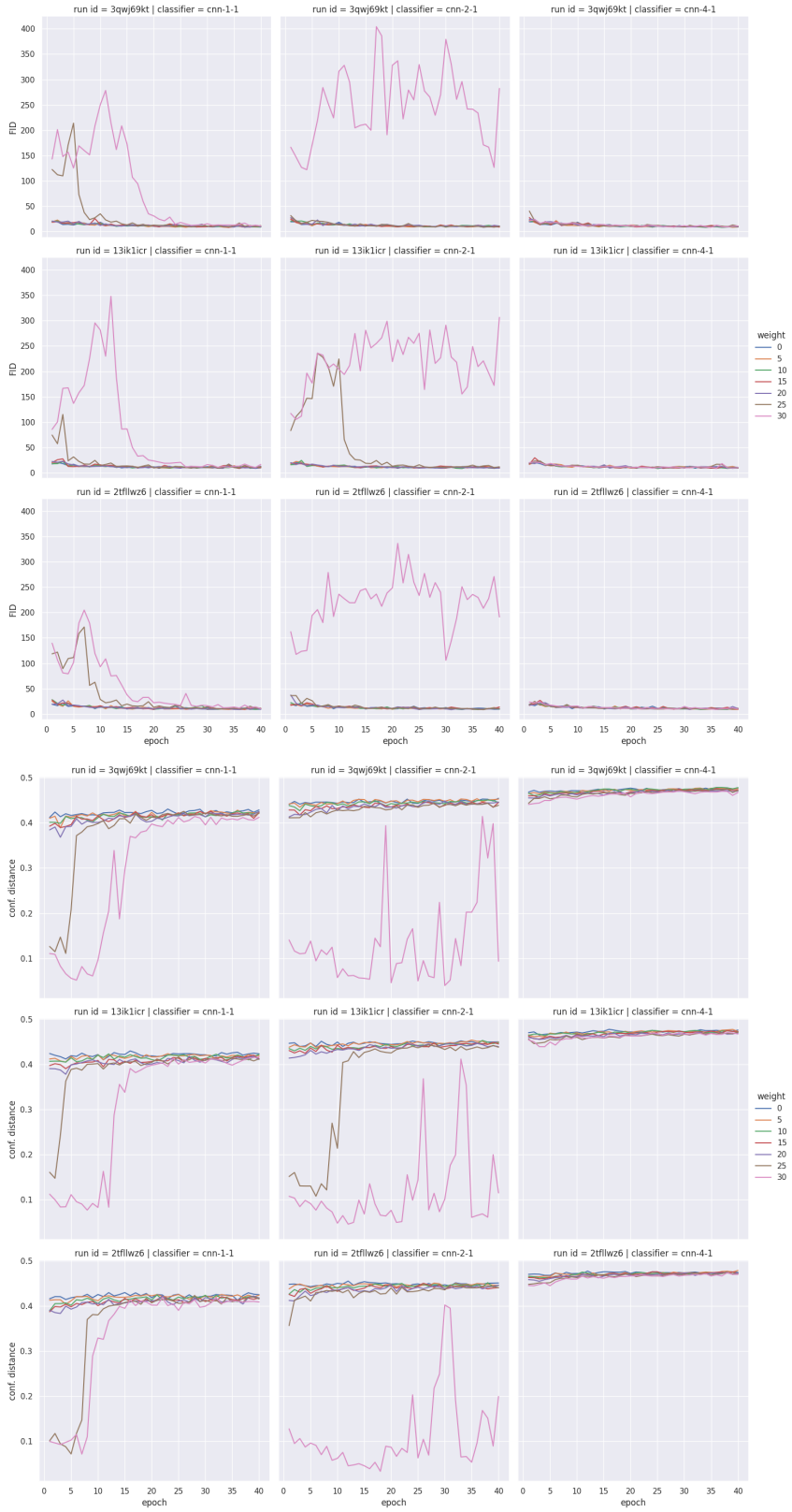


Figure B.8: FID and average confusion distance evolution for classes 7 and 1 of MNIST using a number of pre-train epochs of 10.

B.1.2 5 vs. 3

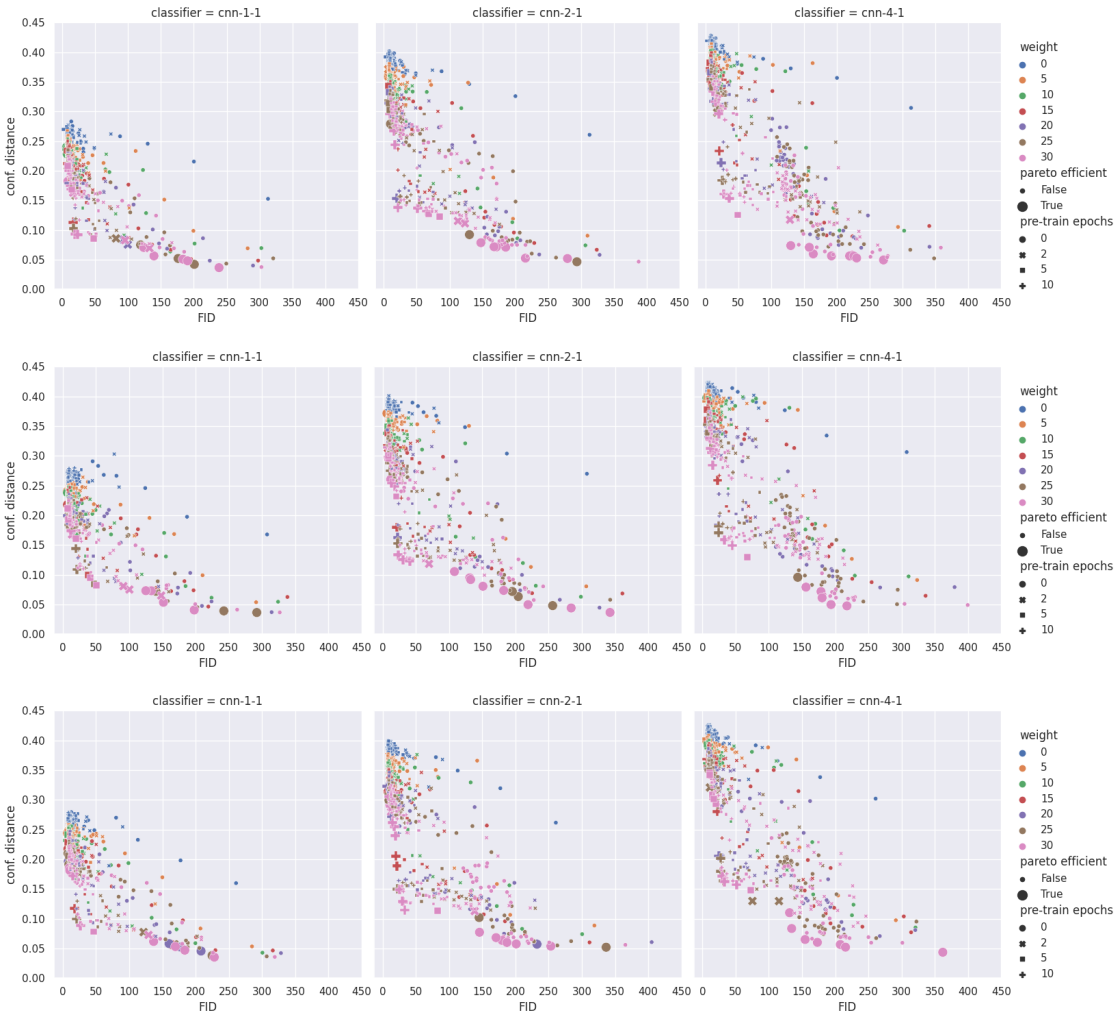


Figure B.9: All metrics gathered using the dataset consisting of examples of classes 5 and 3 of the MNIST dataset. Each row corresponds to a run using a different RNG seed.

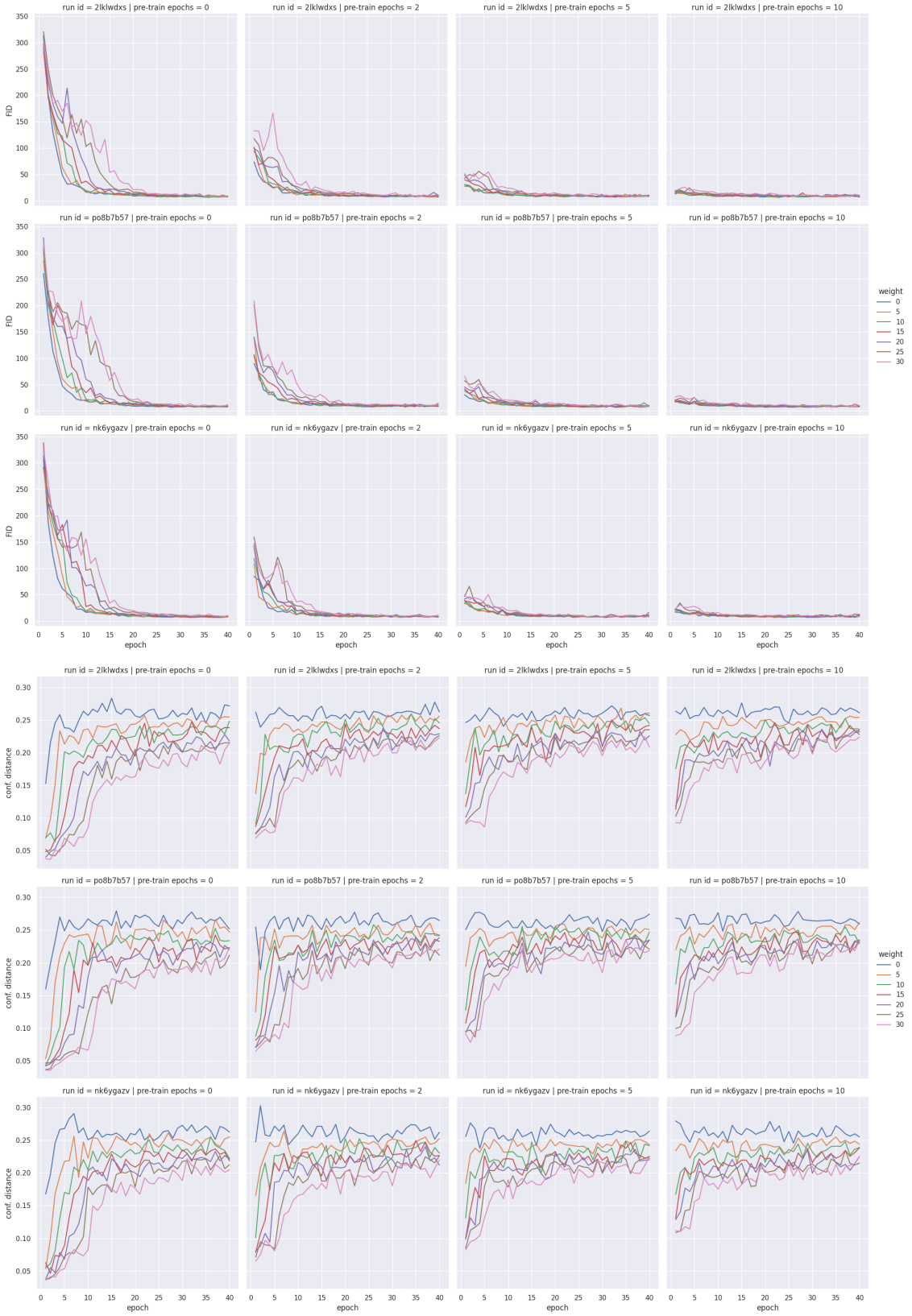


Figure B.10: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a classifier with $nf = 1$.

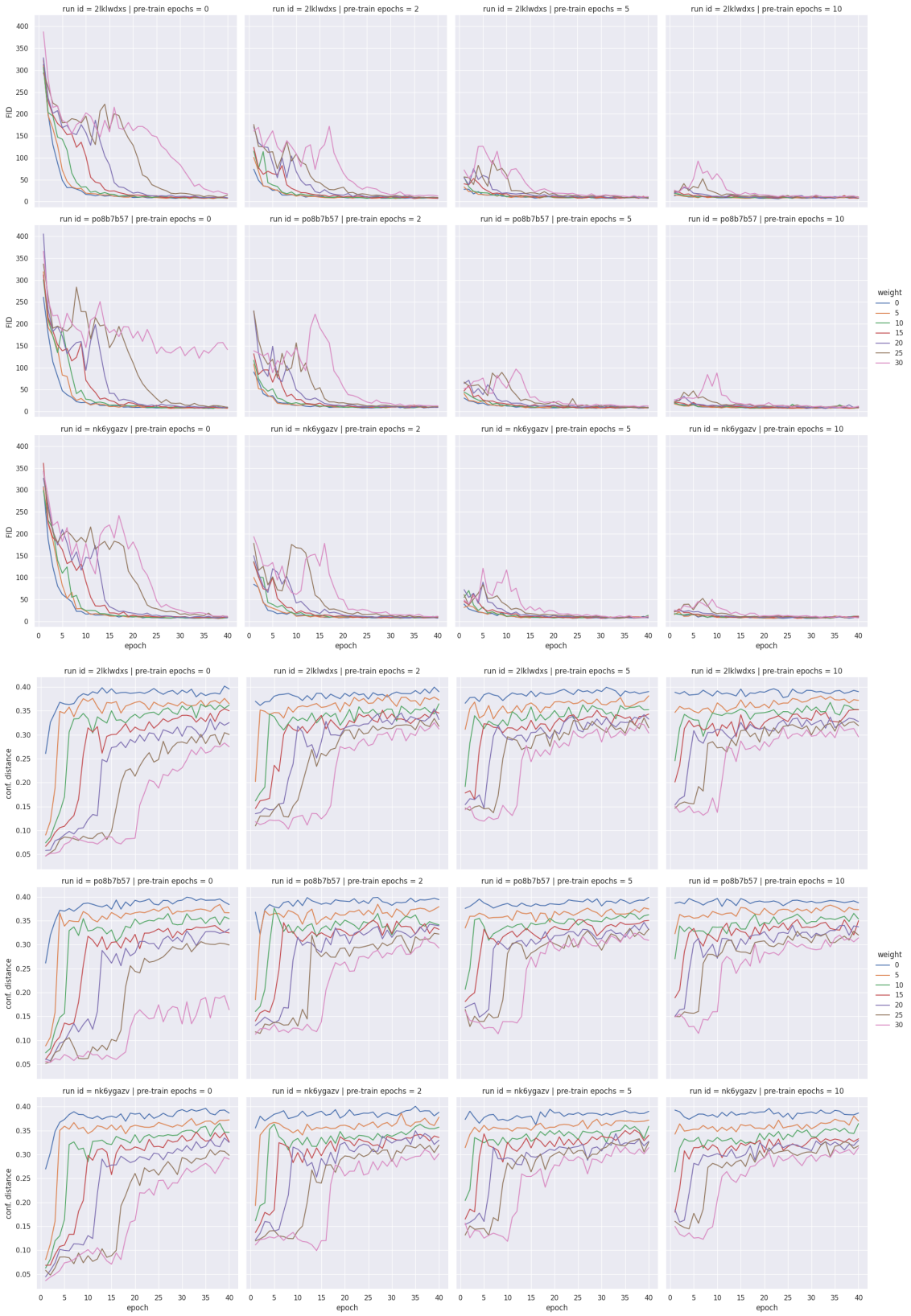


Figure B.11: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a classifier with $nf = 2$.

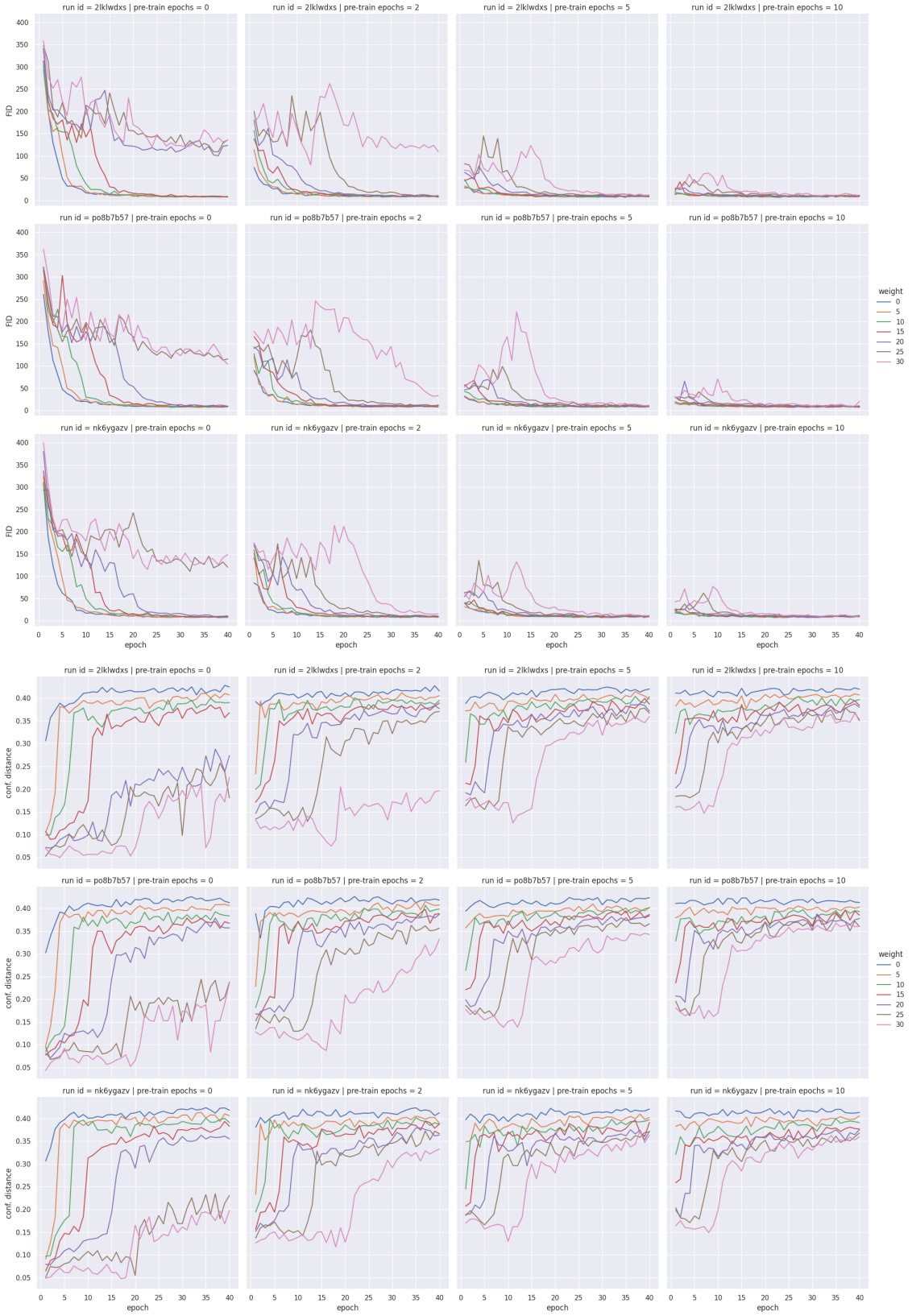


Figure B.12: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a classifier with $nf = 4$.

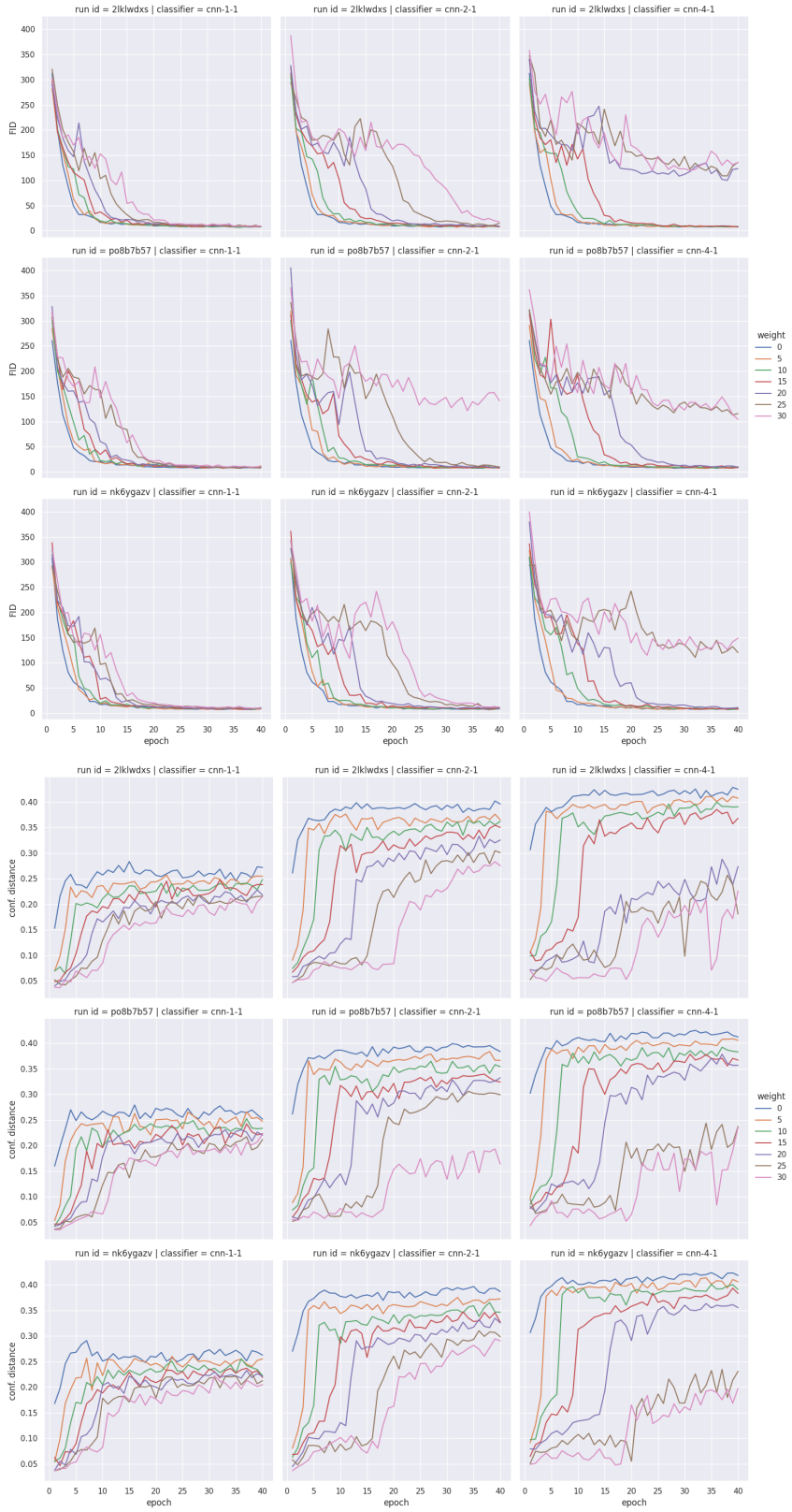


Figure B.13: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a number of pre-train epochs of 0.

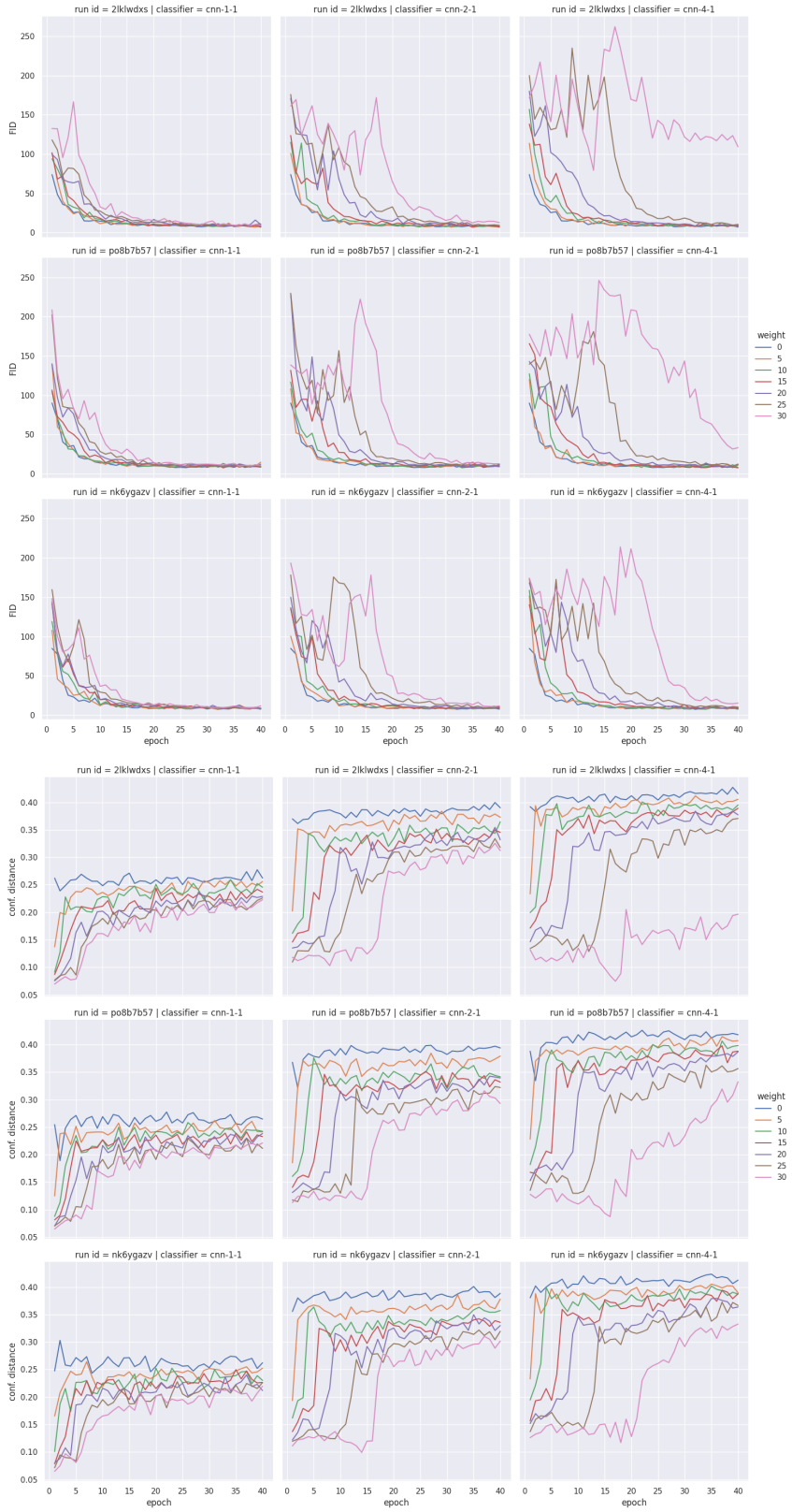


Figure B.14: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a number of pre-train epochs of 2.

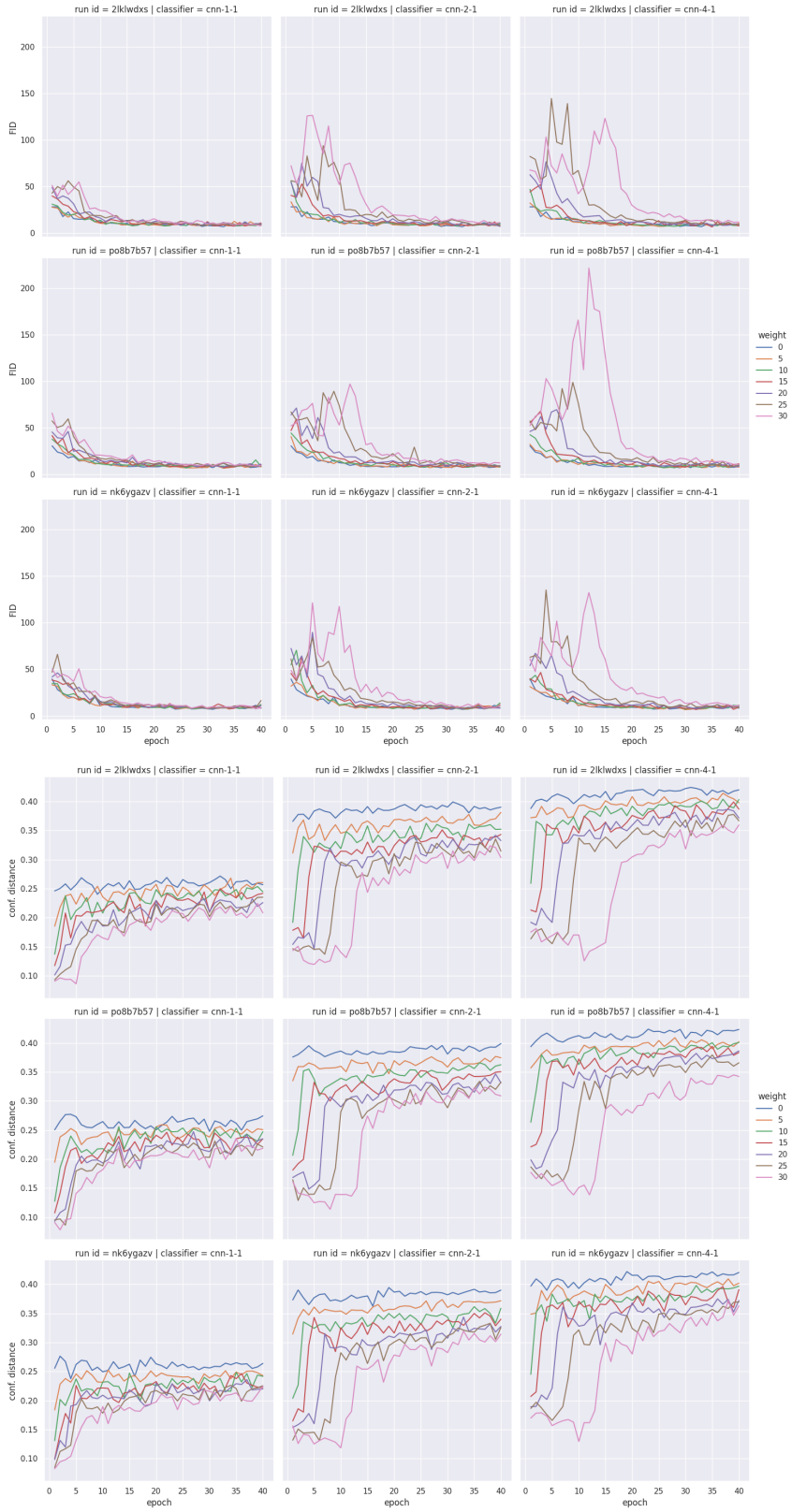


Figure B.15: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a number of pre-train epochs of 5.

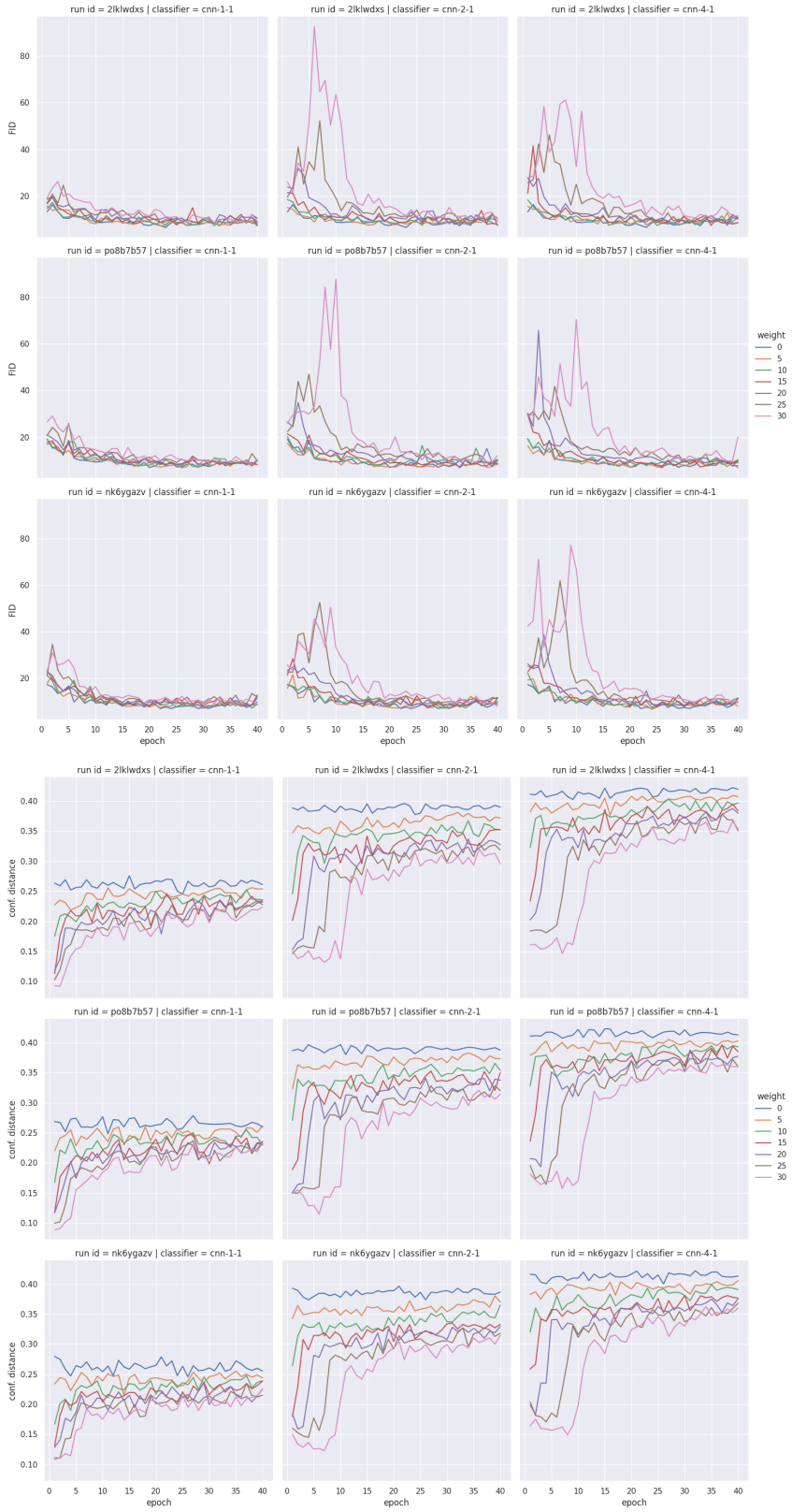


Figure B.16: FID and average confusion distance evolution for classes 5 and 3 of MNIST using a number of pre-train epochs of 10.

B.1.3 8 vs. 0

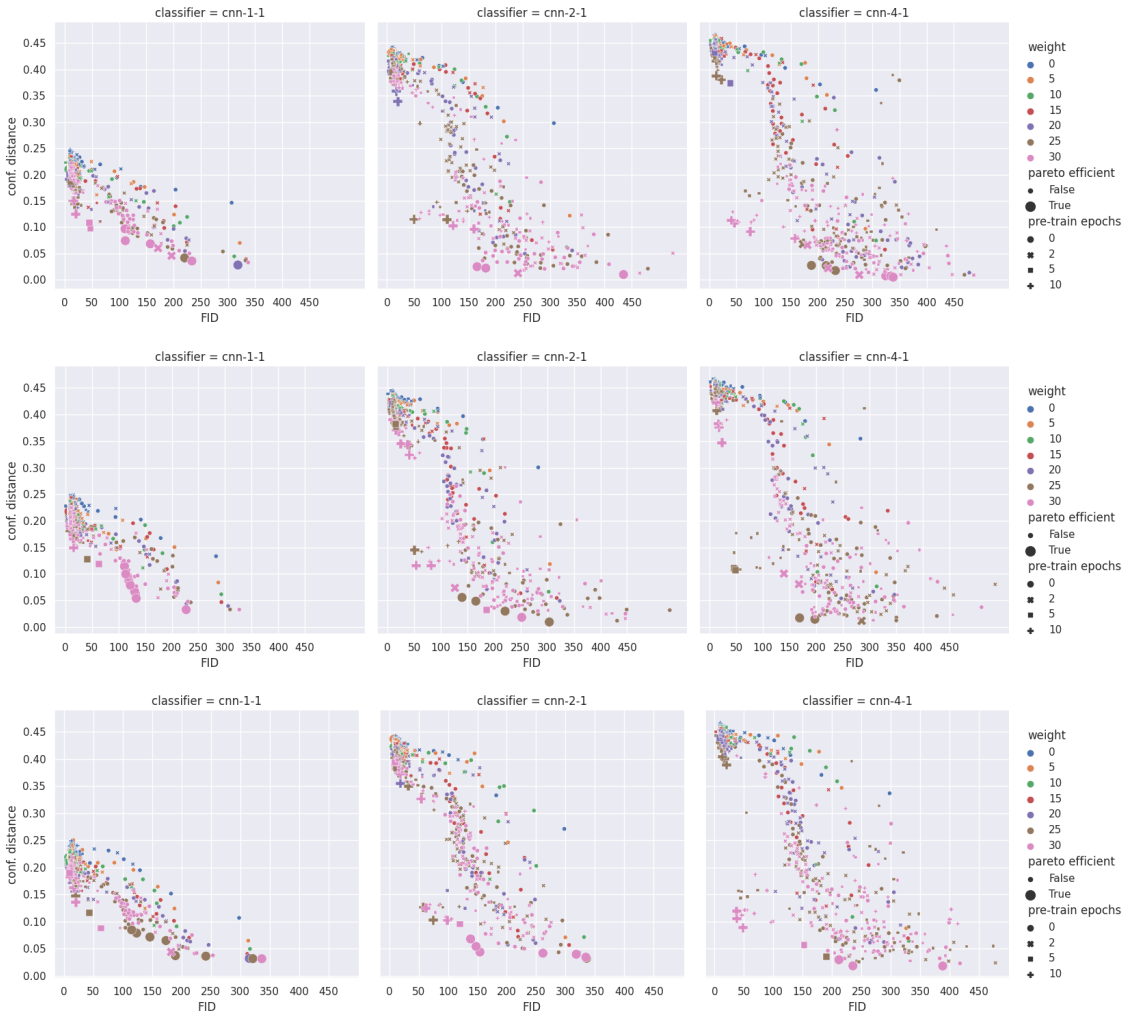


Figure B.17: All metrics gathered using the dataset consisting of examples of classes 8 and 0 of the MNIST dataset. Each row corresponds to a run using a different RNG seed.

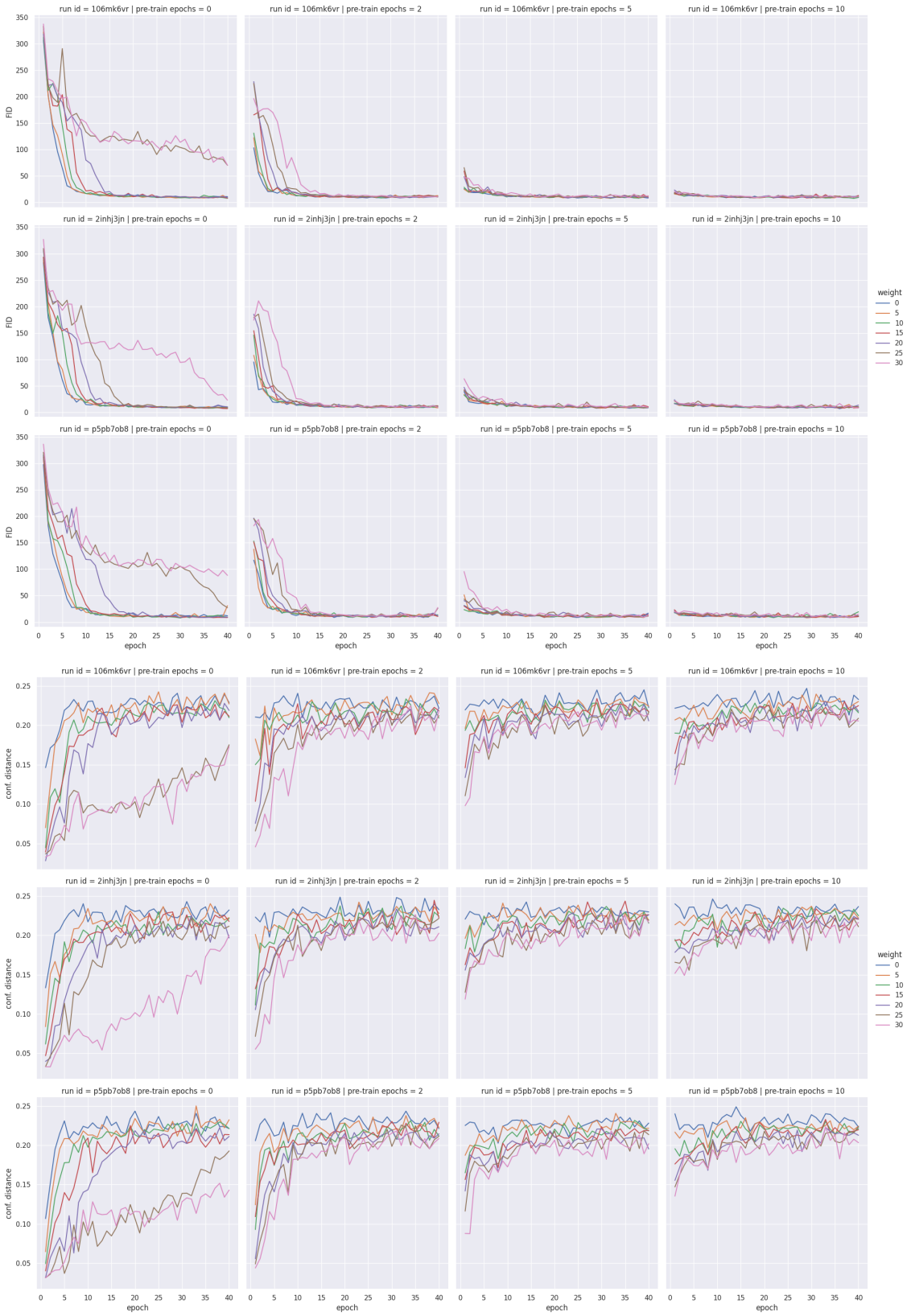


Figure B.18: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a classifier with $nf = 1$.

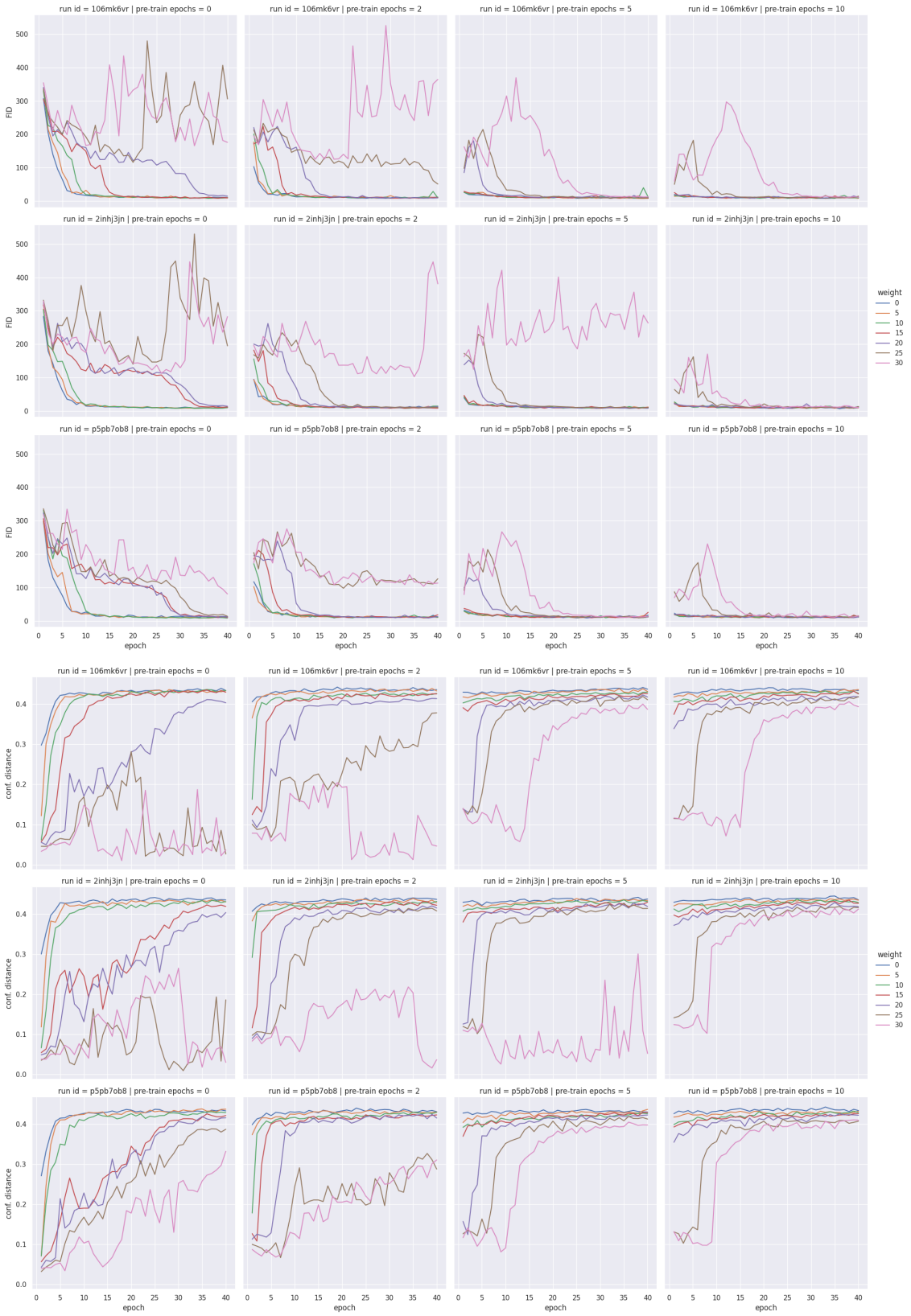


Figure B.19: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a classifier with $nf = 2$.

Figure B.20: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a classifier with $nf = 4$.

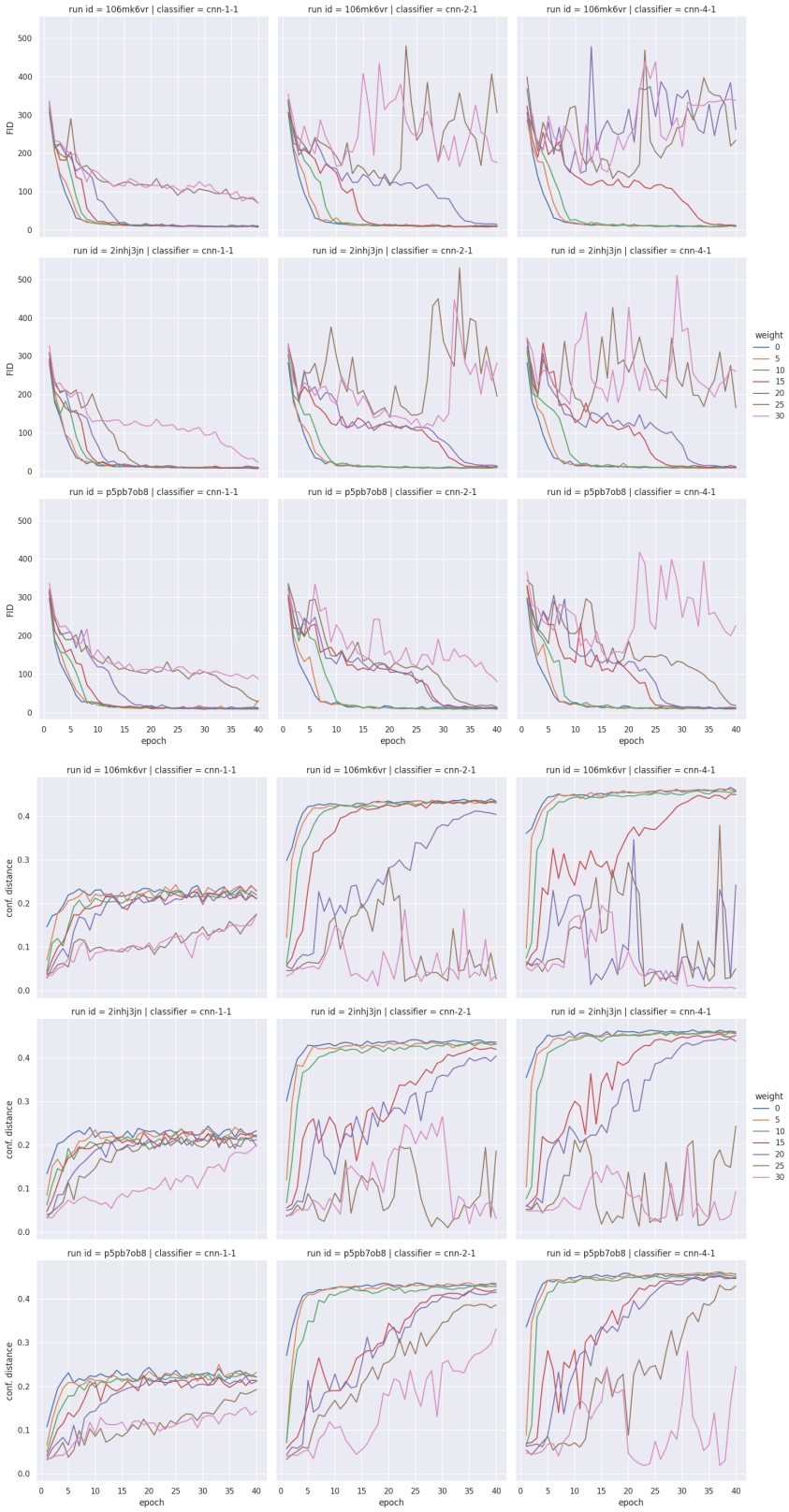


Figure B.21: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a number of pre-train epochs of 0.

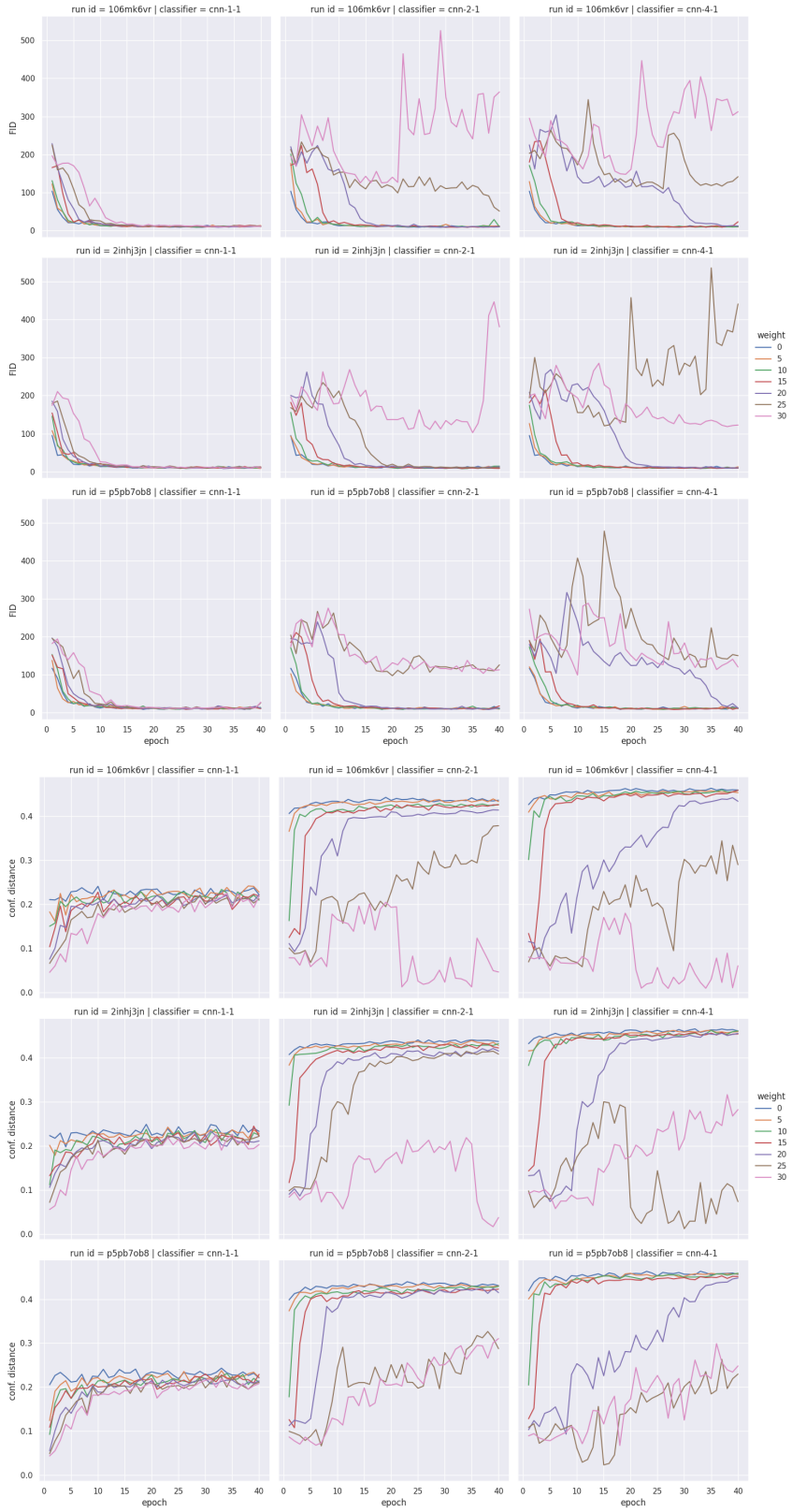


Figure B.22: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a number of pre-train epochs of 2.

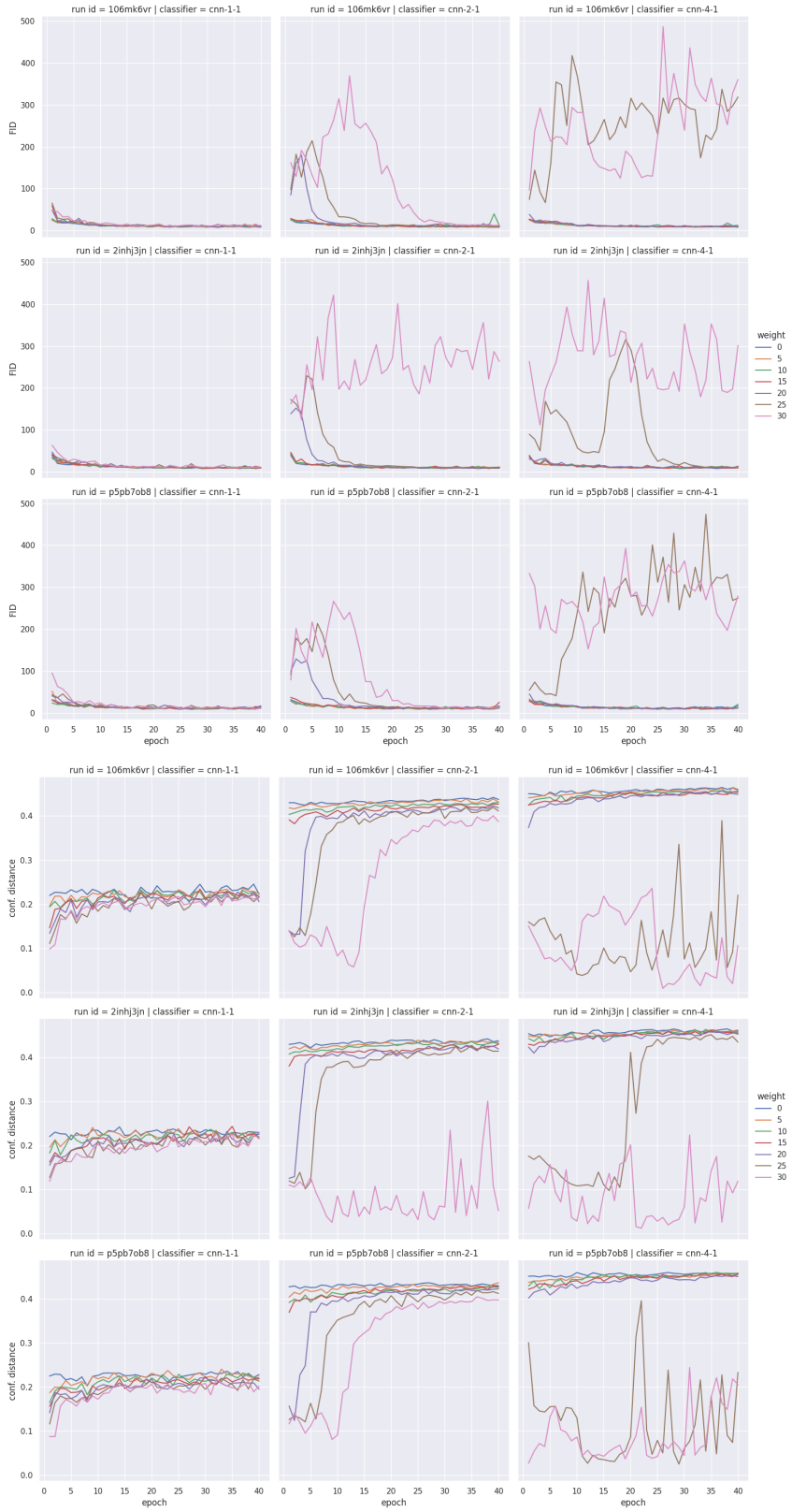


Figure B.23: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a number of pre-train epochs of 5.

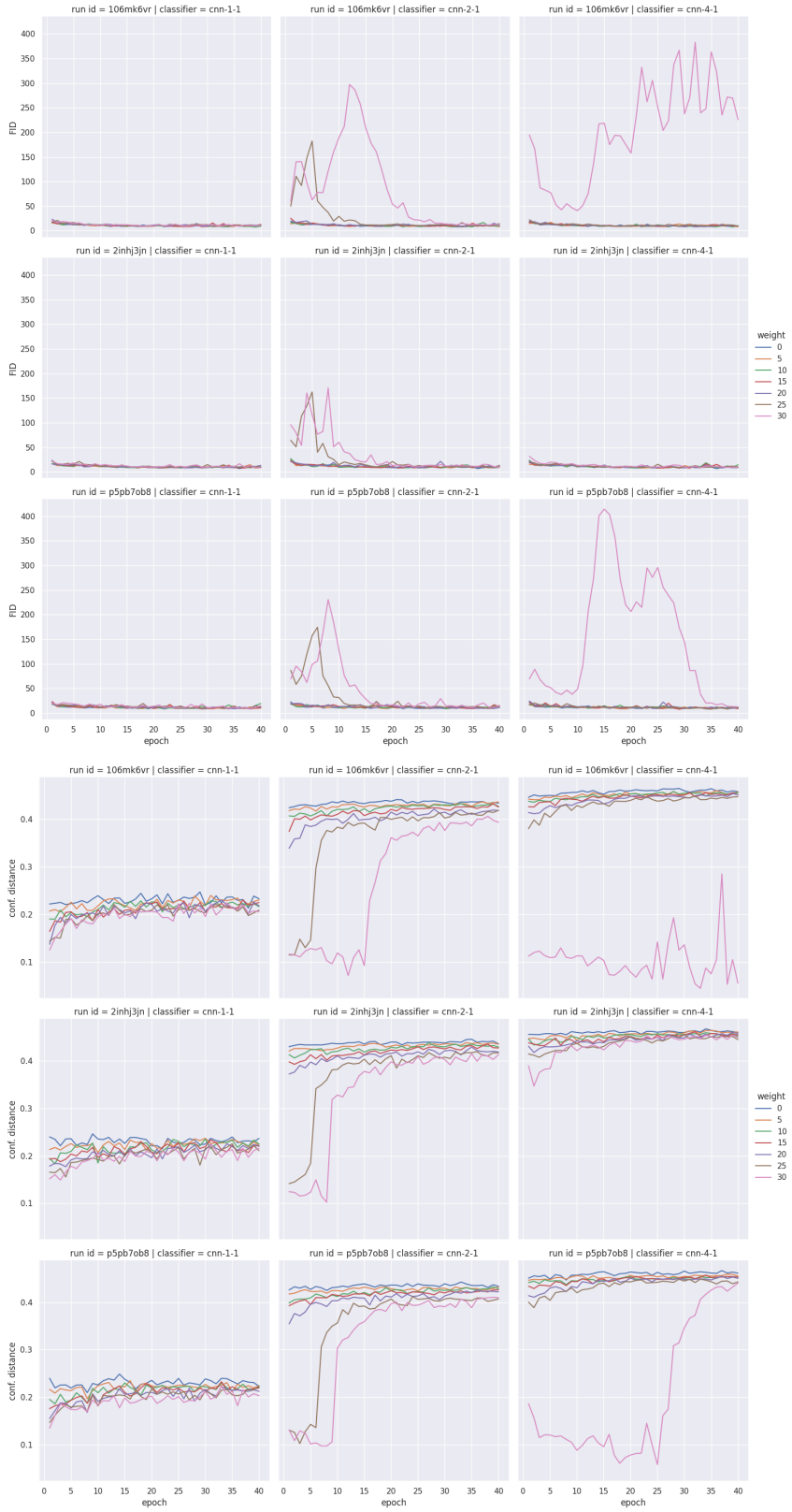


Figure B.24: FID and average confusion distance evolution for classes 8 and 0 of MNIST using a number of pre-train epochs of 10.

B.1.4 9 vs. 4

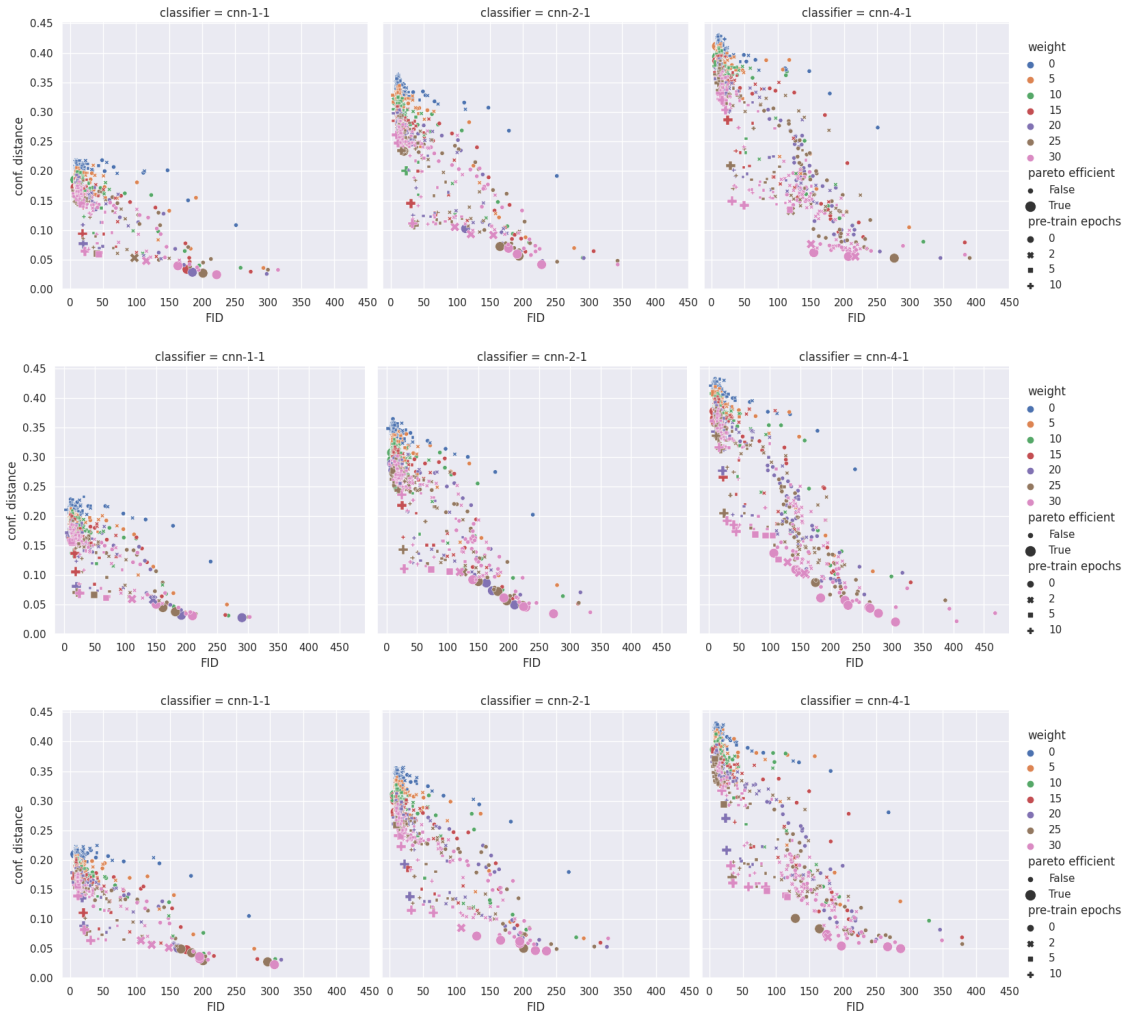


Figure B.25: All metrics gathered using the dataset consisting of examples of classes 9 and 4 of the MNIST dataset. Each row corresponds to a run using a different RNG seed.

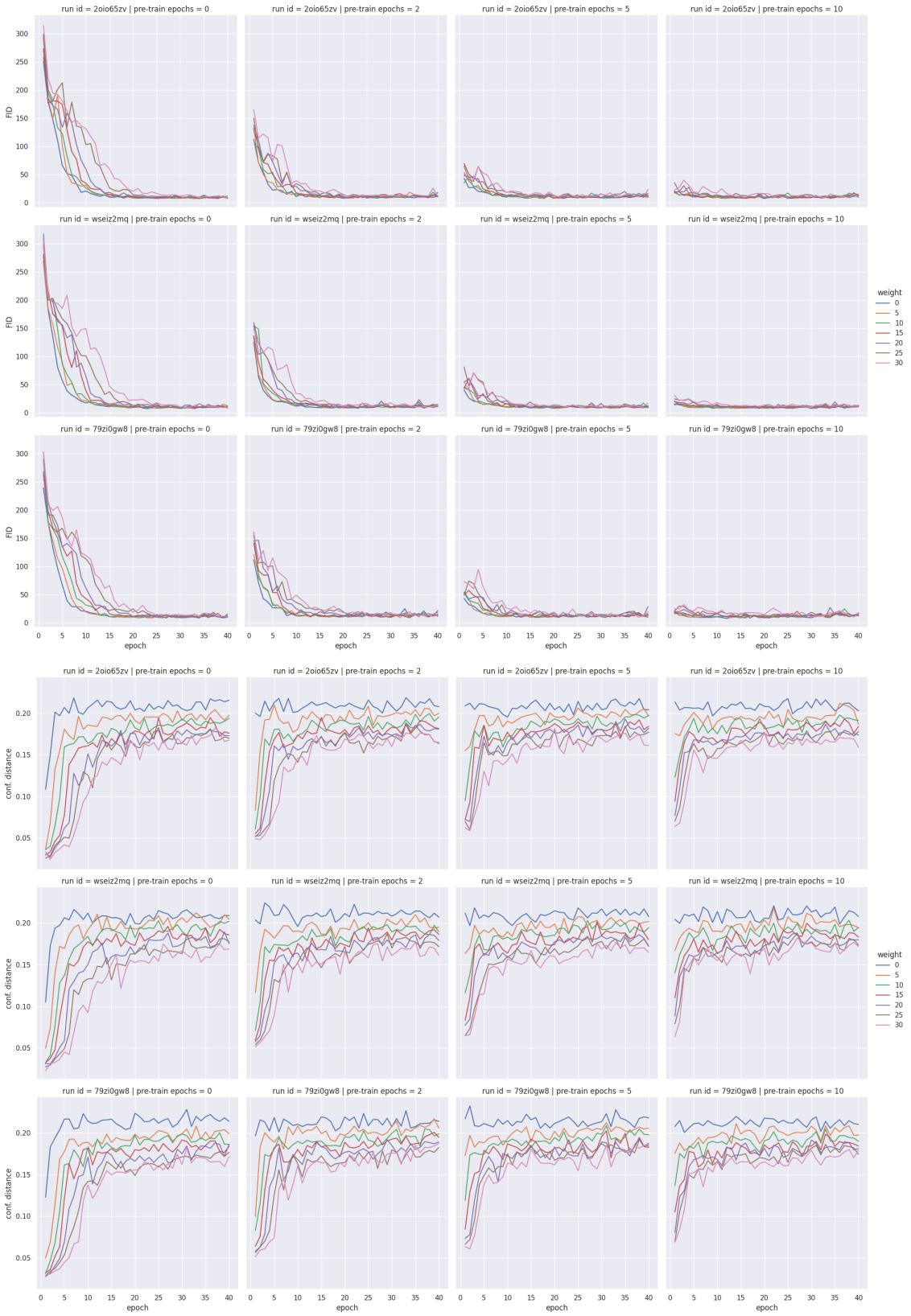


Figure B.26: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a classifier with $nf = 1$.

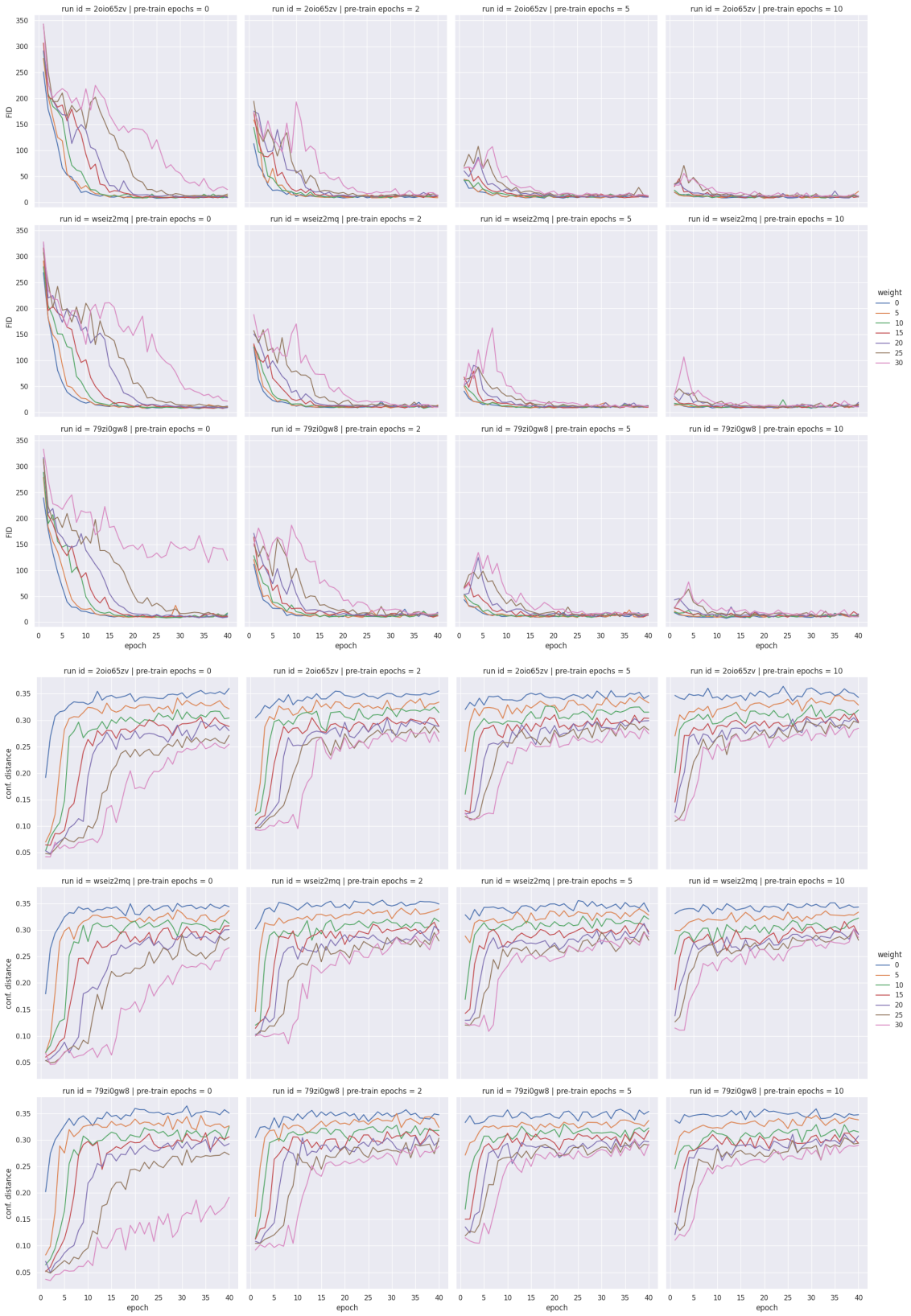


Figure B.27: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a classifier with $nf = 2$.

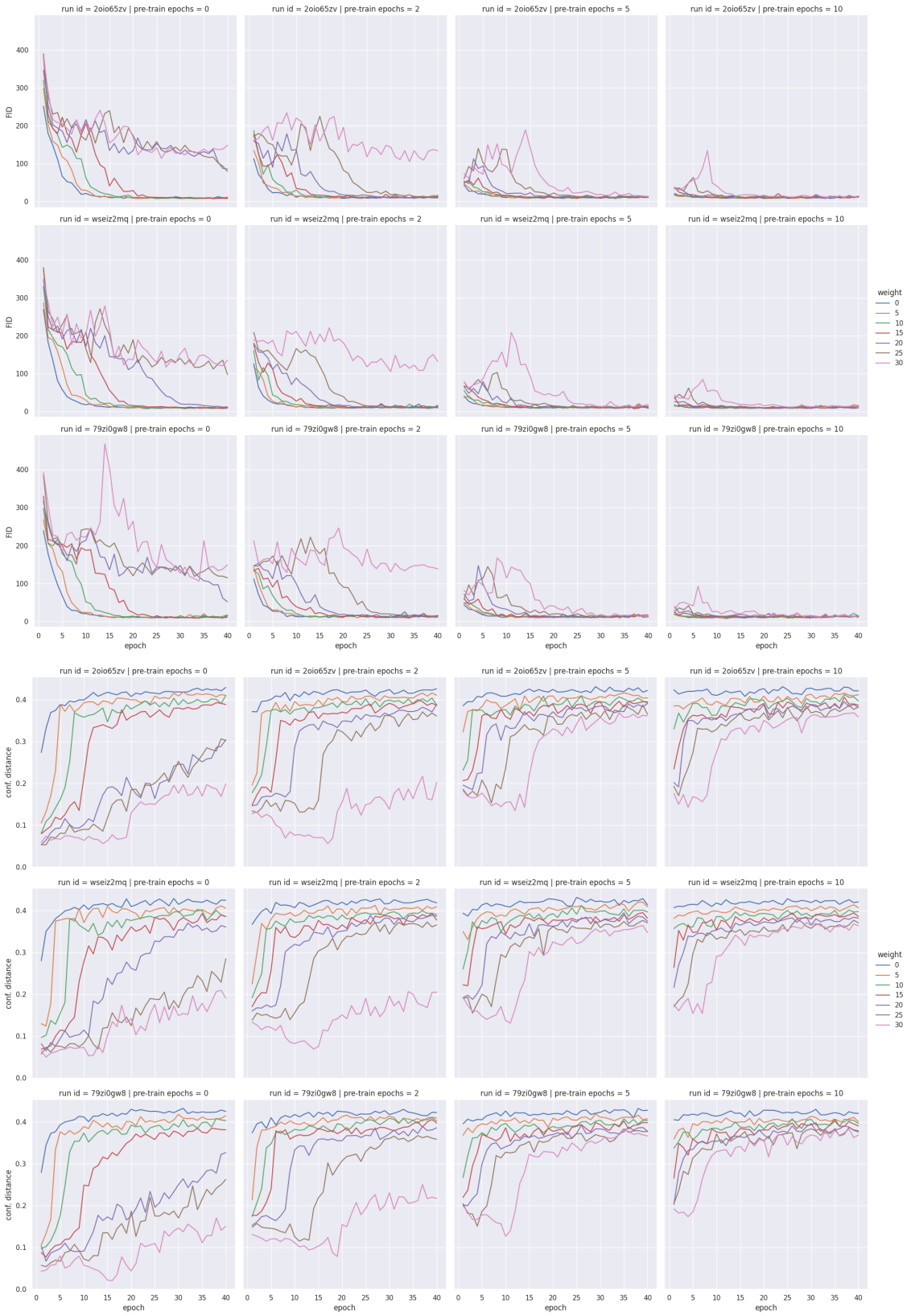


Figure B.28: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a classifier with $nf = 4$.

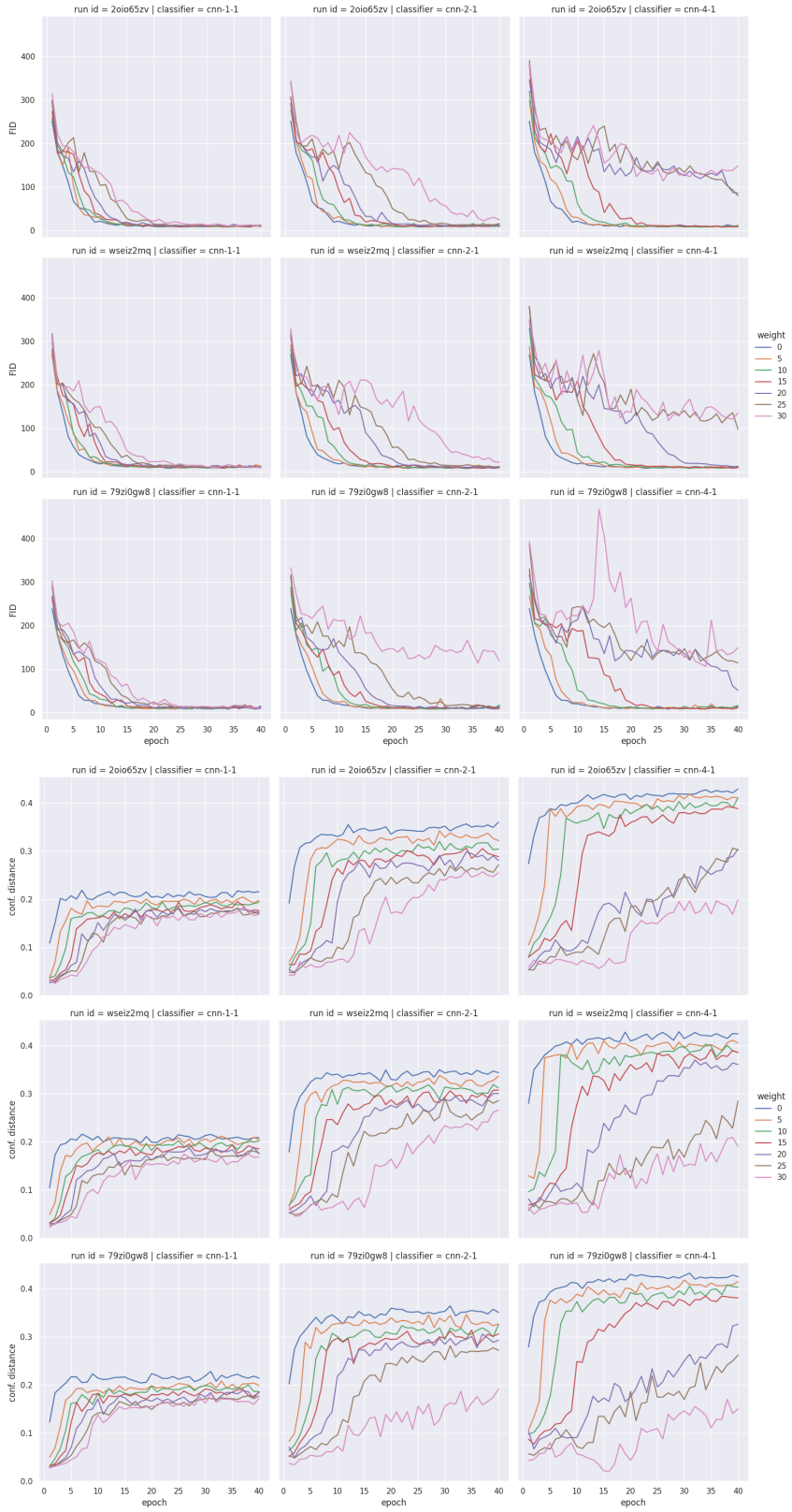


Figure B.29: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a number of pre-train epochs of 0.

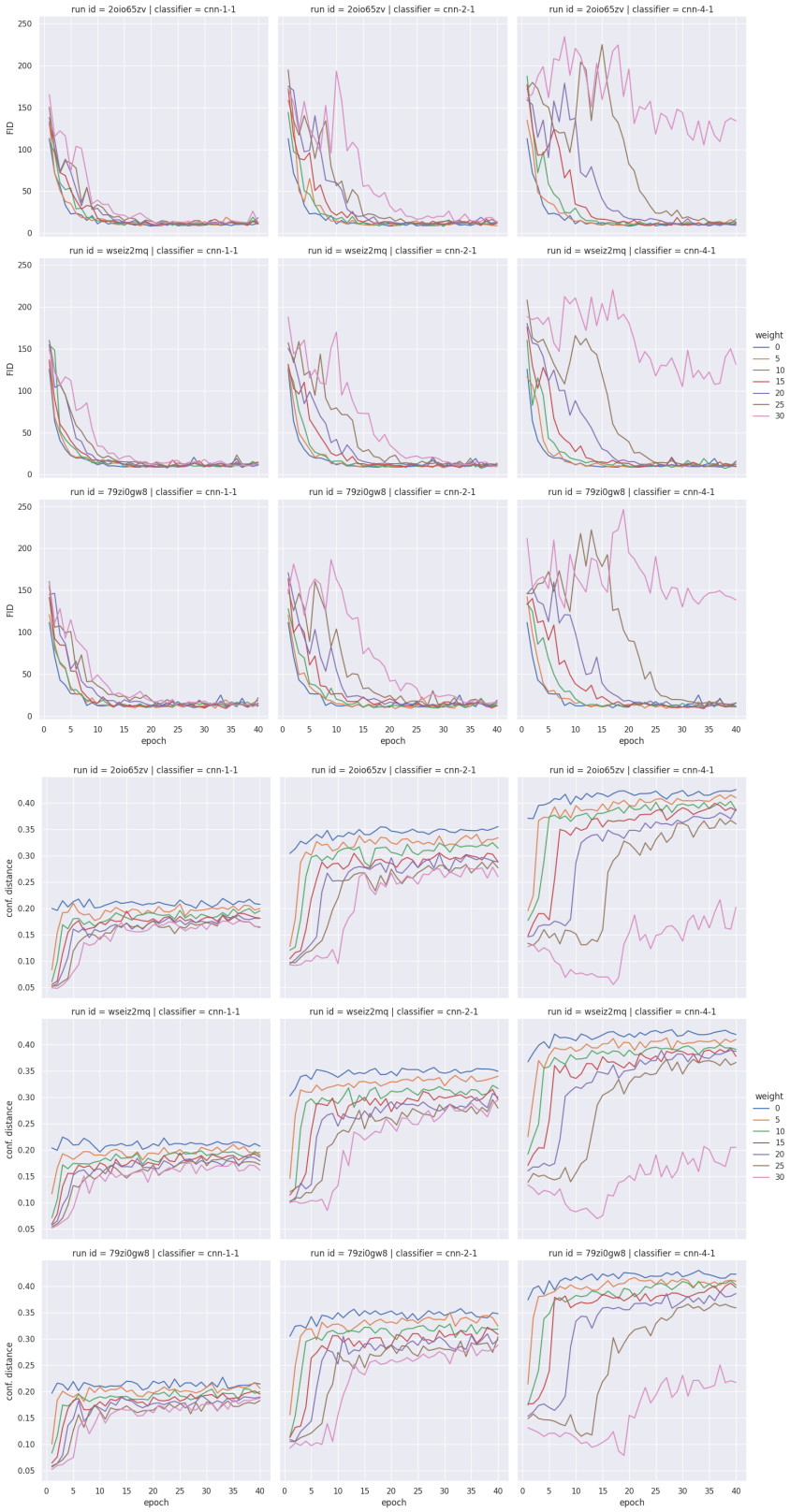


Figure B.30: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a number of pre-train epochs of 2.

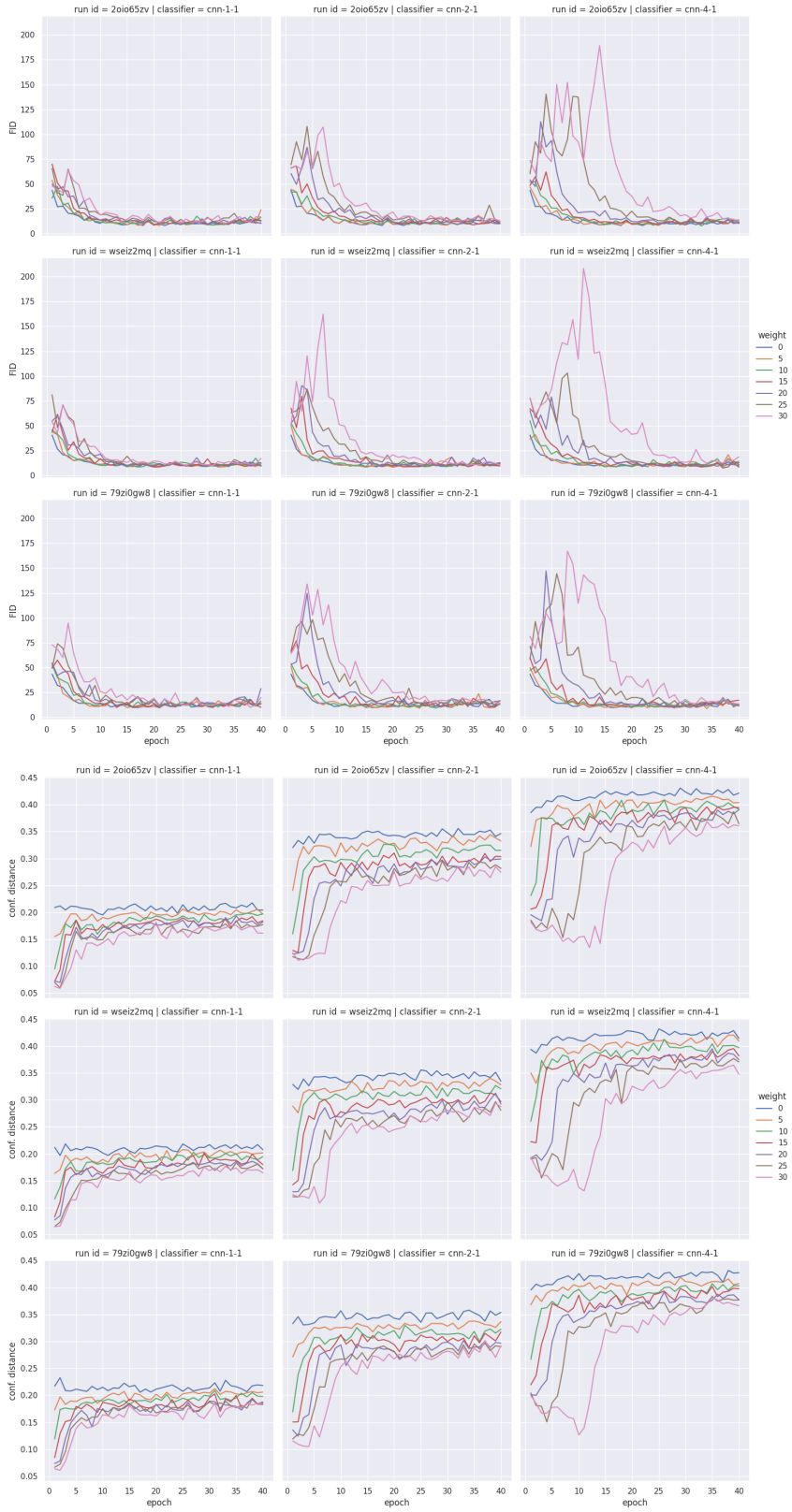


Figure B.31: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a number of pre-train epochs of 5.

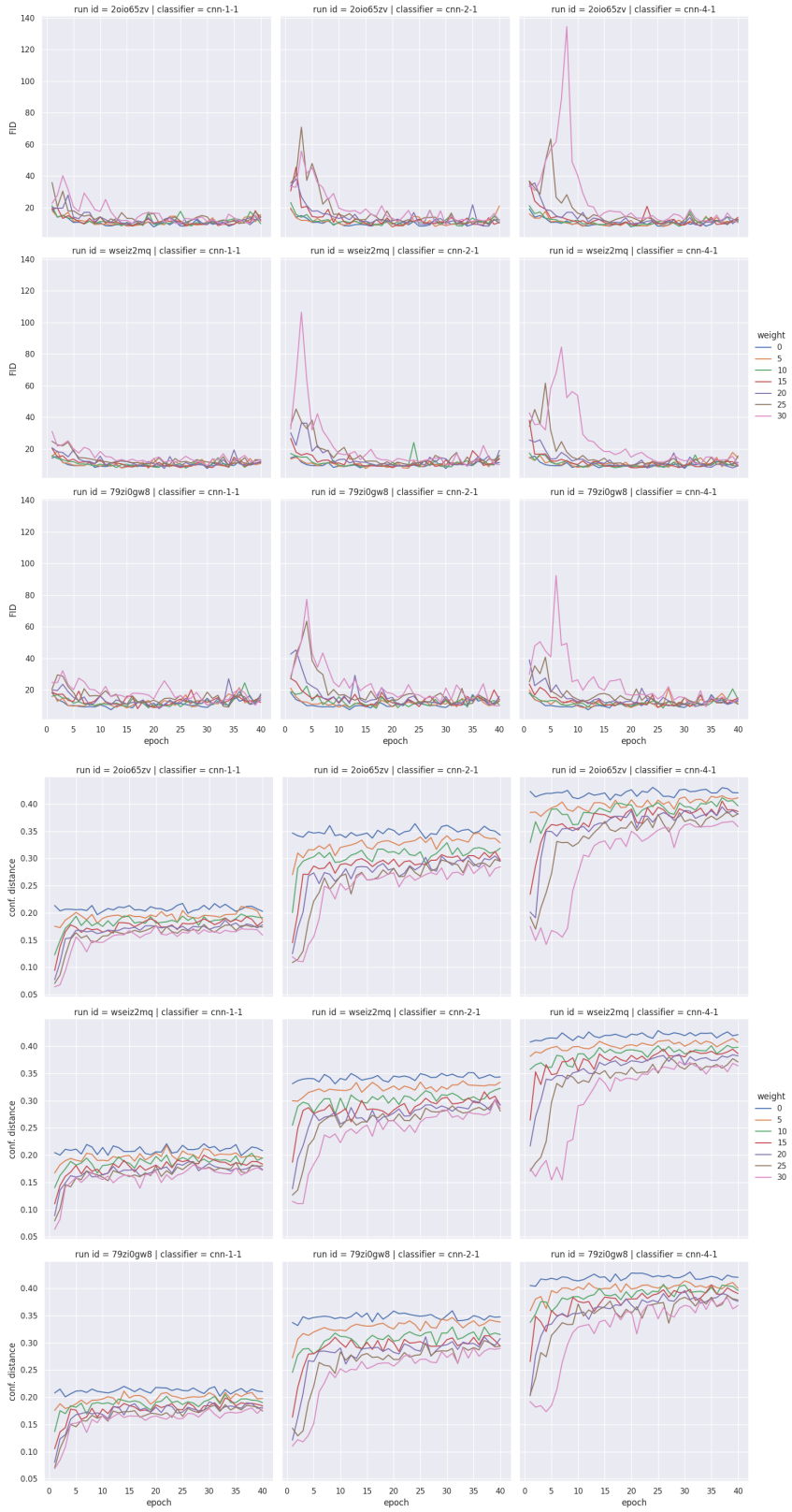


Figure B.32: FID and average confusion distance evolution for classes 9 and 4 of MNIST using a number of pre-train epochs of 10.

B.2 Fashion MNIST

B.2.1 "Dress" vs. "T-shirt/top"

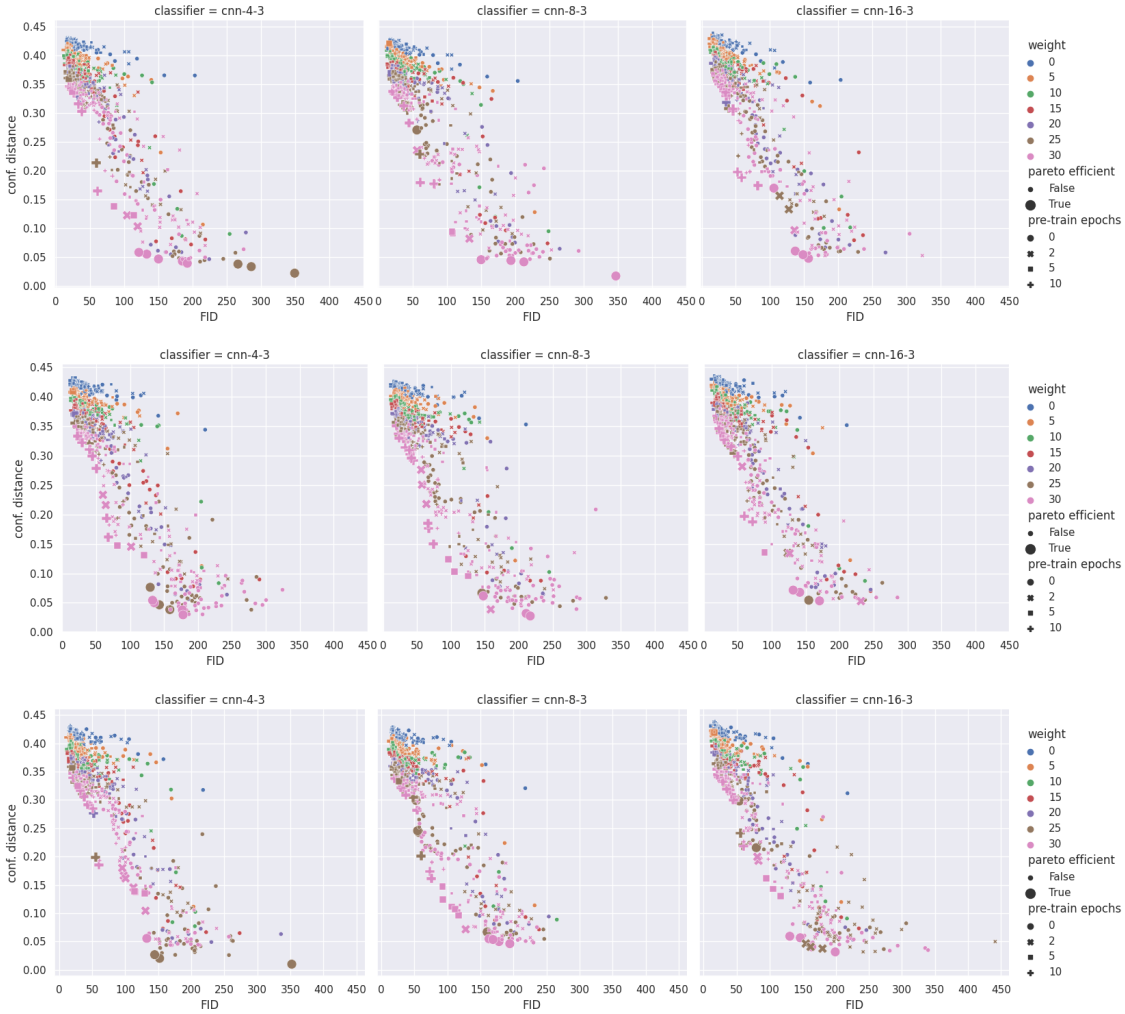


Figure B.33: All metrics gathered using the dataset consisting of examples of classes "Dress" and "T-shirt/top" of the Fashion MNIST dataset. Each row corresponds to a run using a different RNG seed.

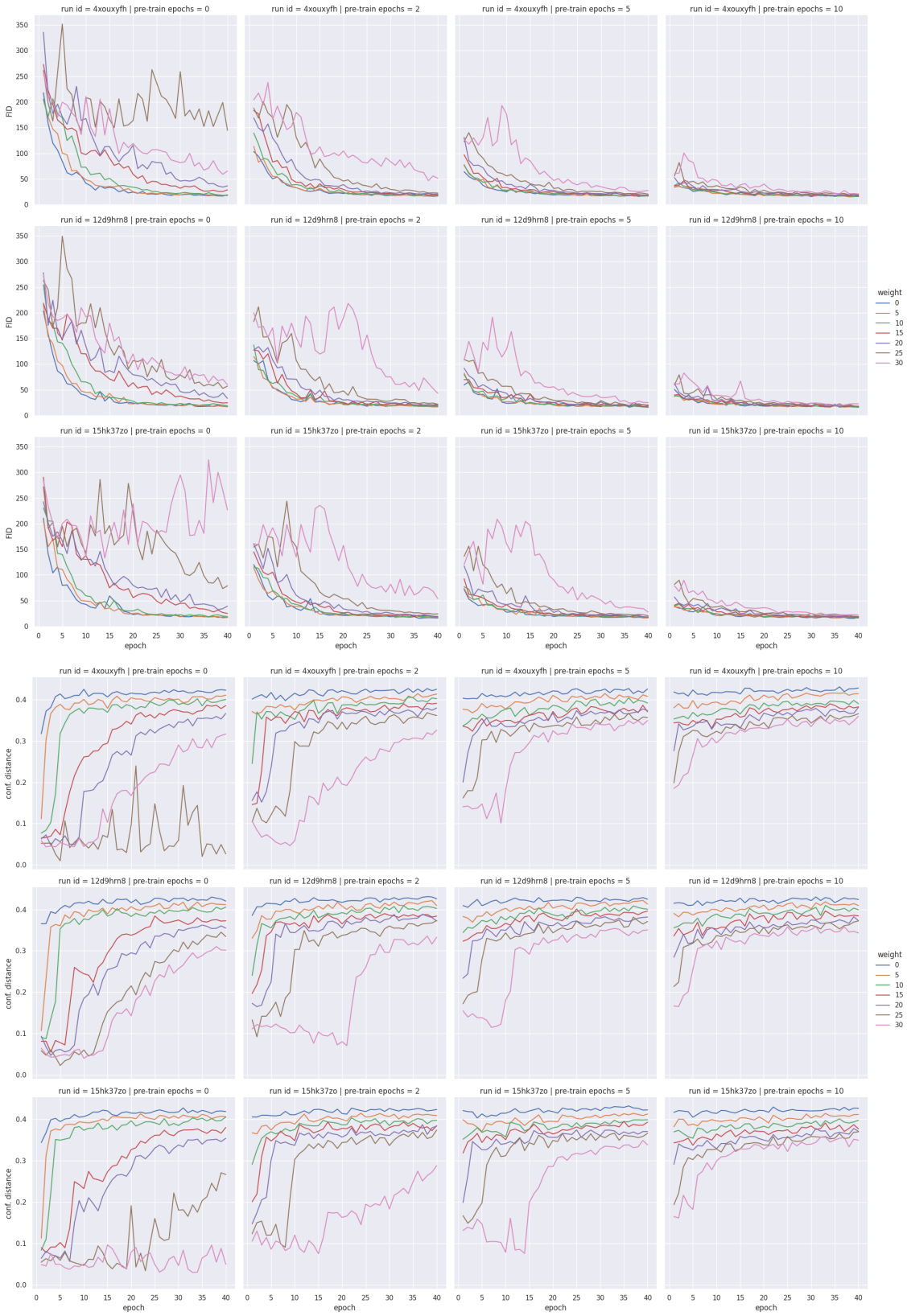


Figure B.34: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST using a classifier with $nf = 4$.

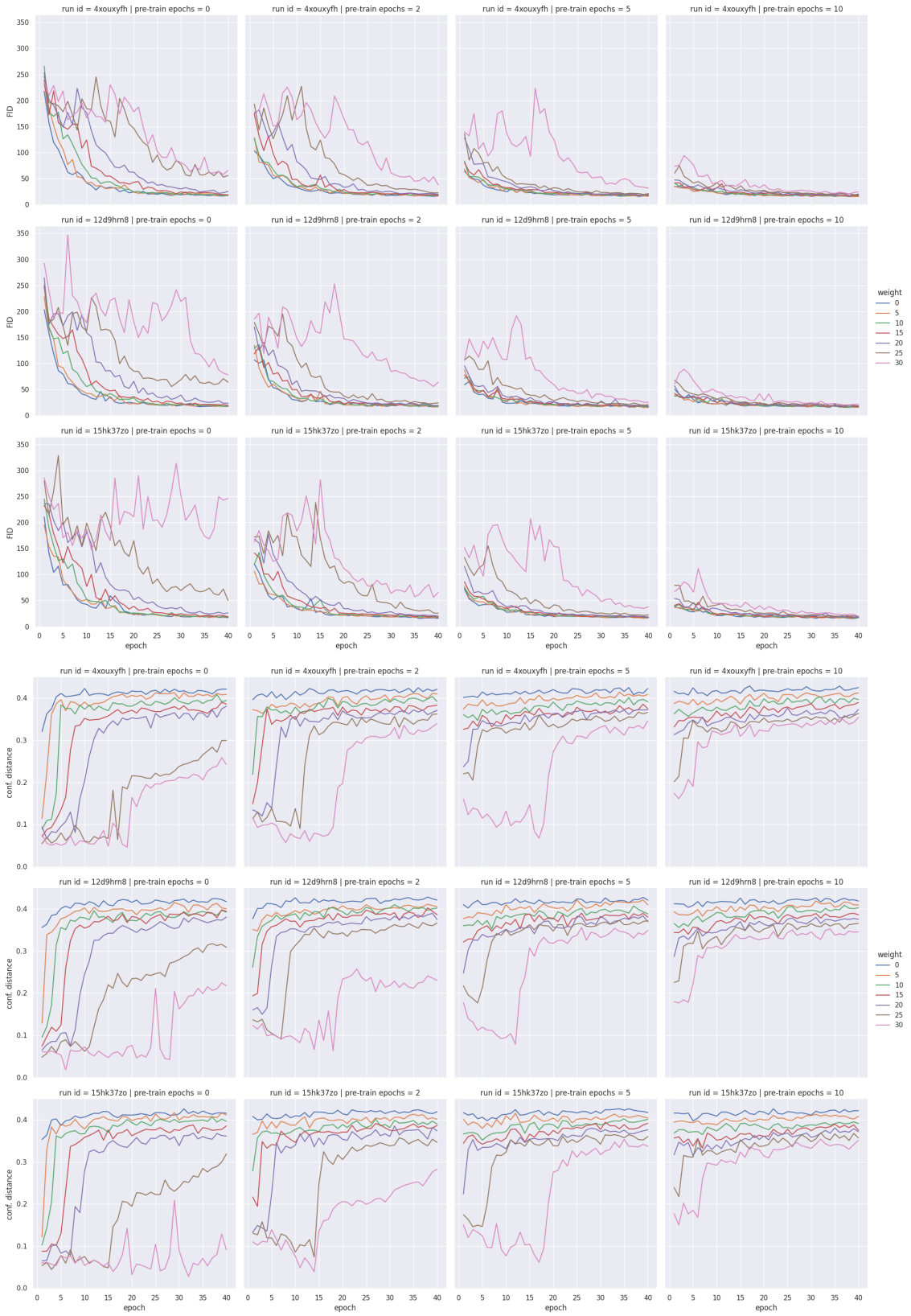


Figure B.35: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST using a classifier with $nf = 8$.

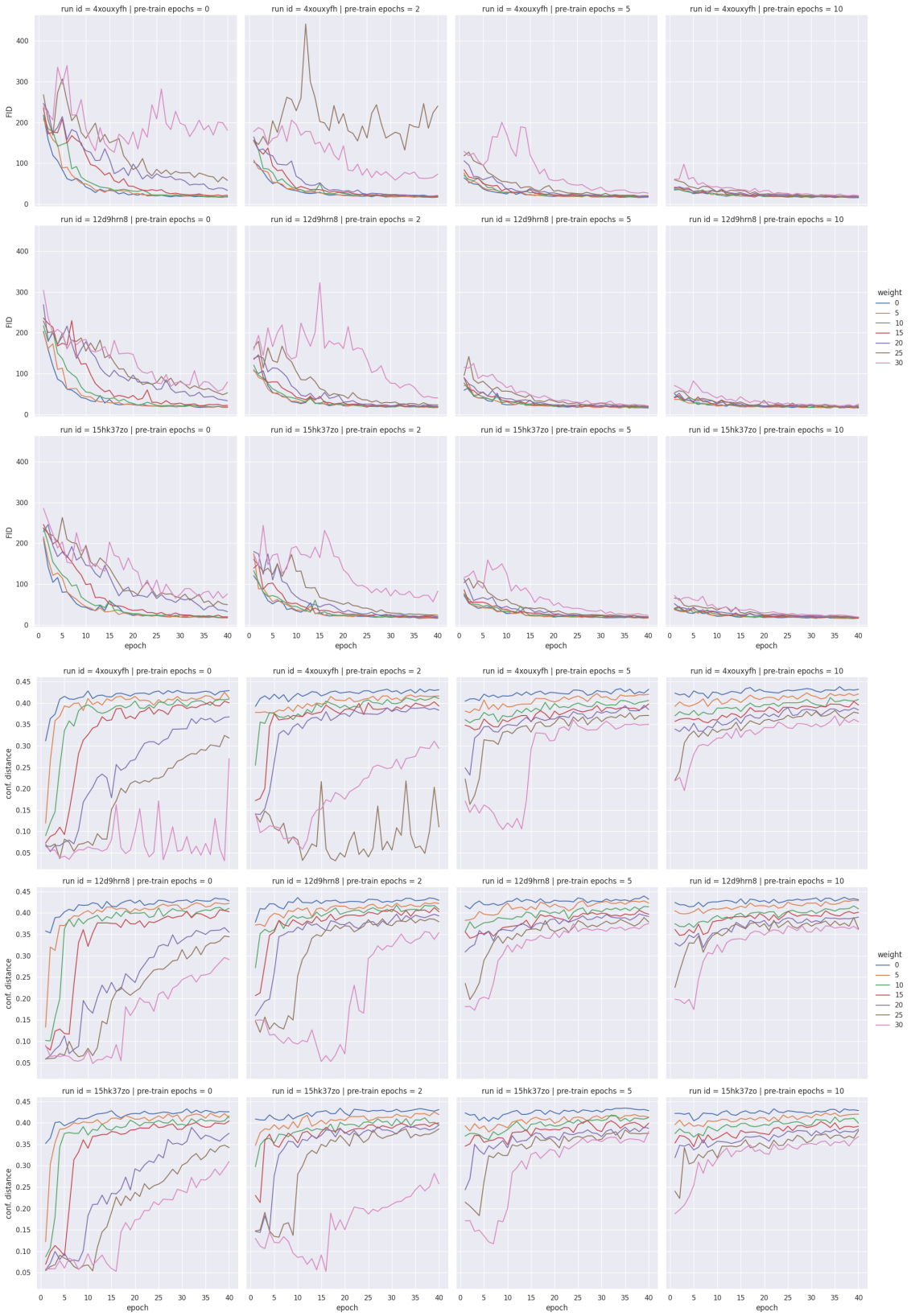


Figure B.36: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST using a classifier with $nf = 16$.

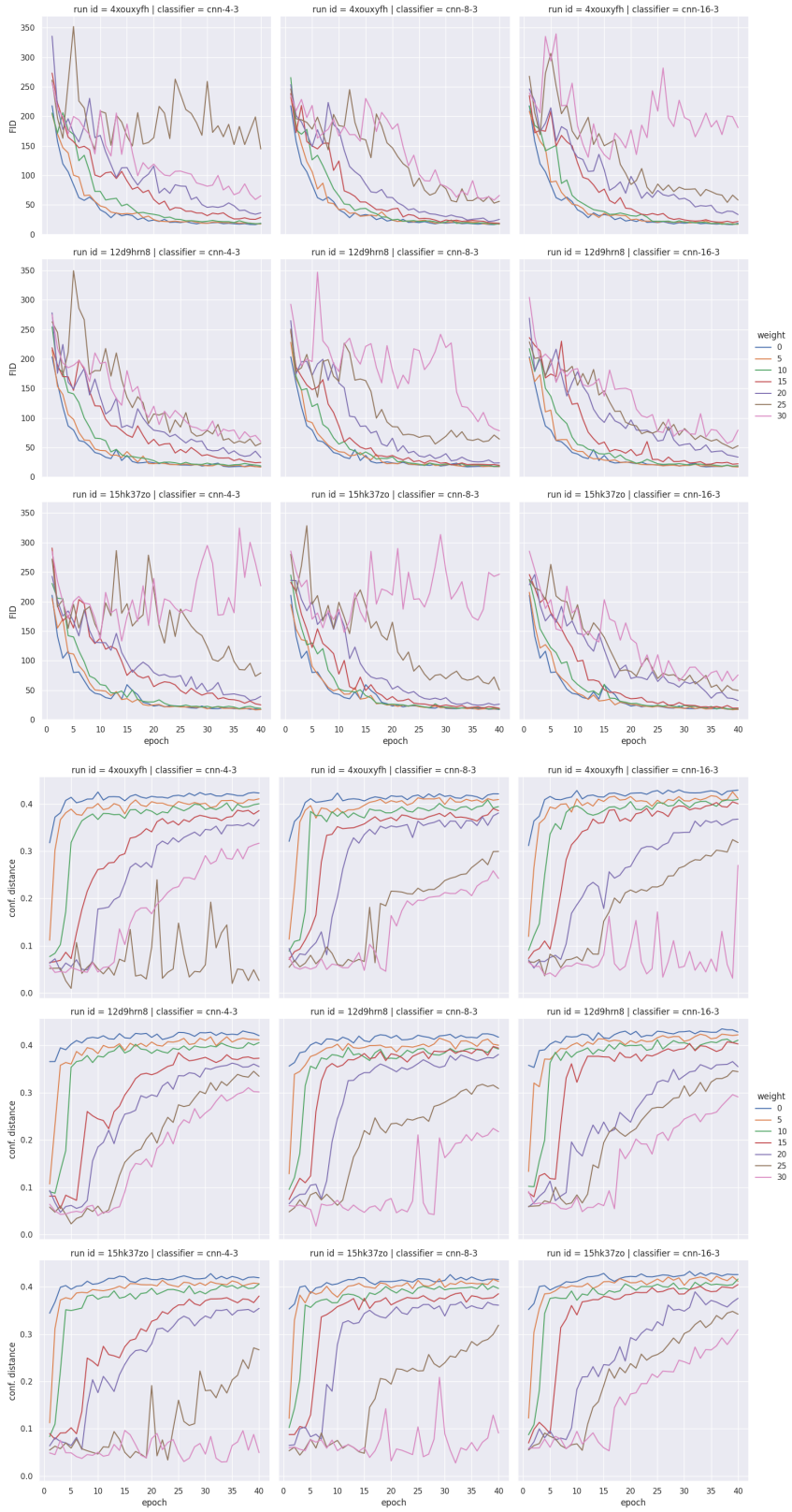


Figure B.37: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST a number of pre-train epochs of 0.

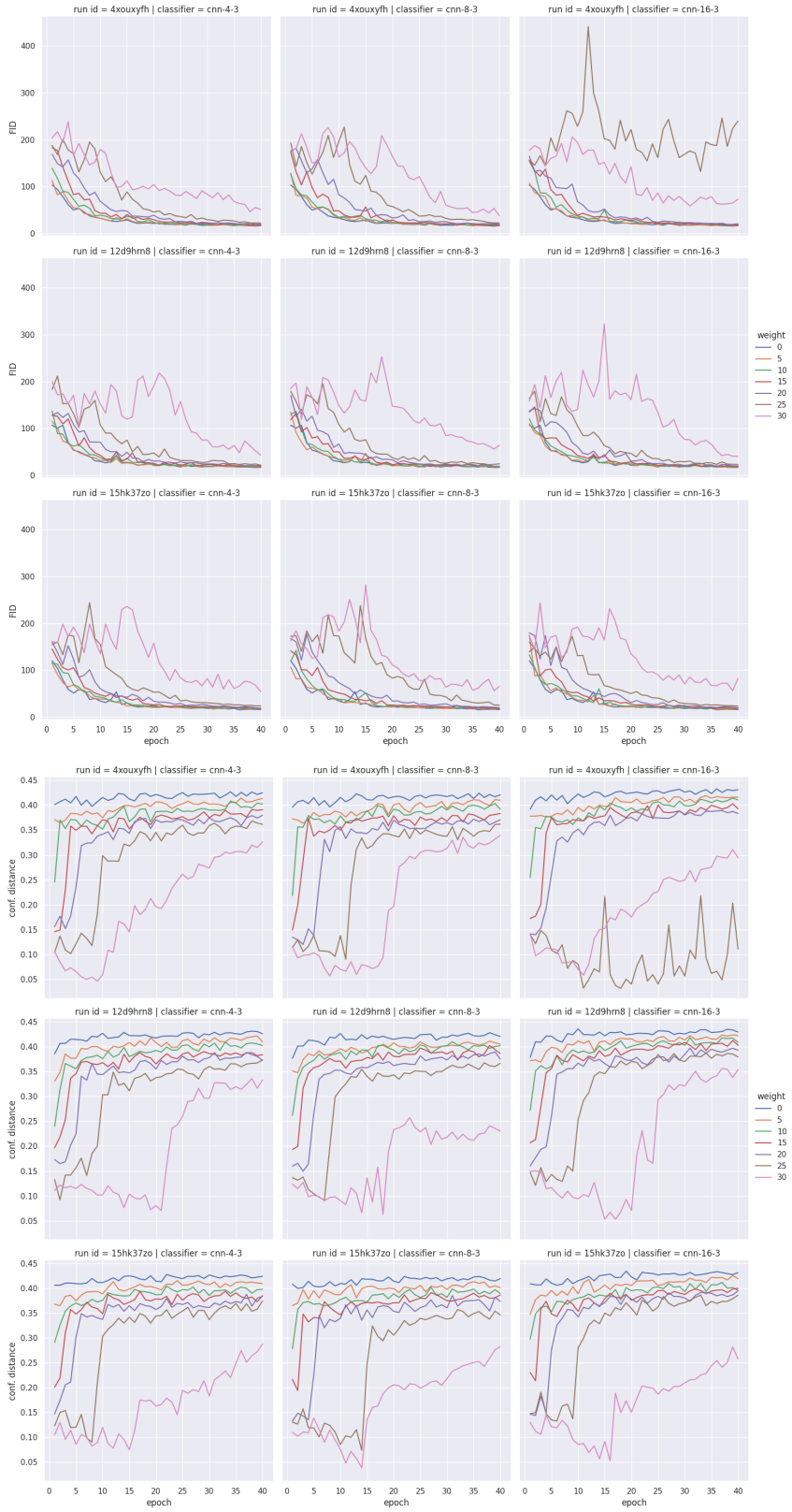


Figure B.38: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST using a number of pre-train epochs of 2.

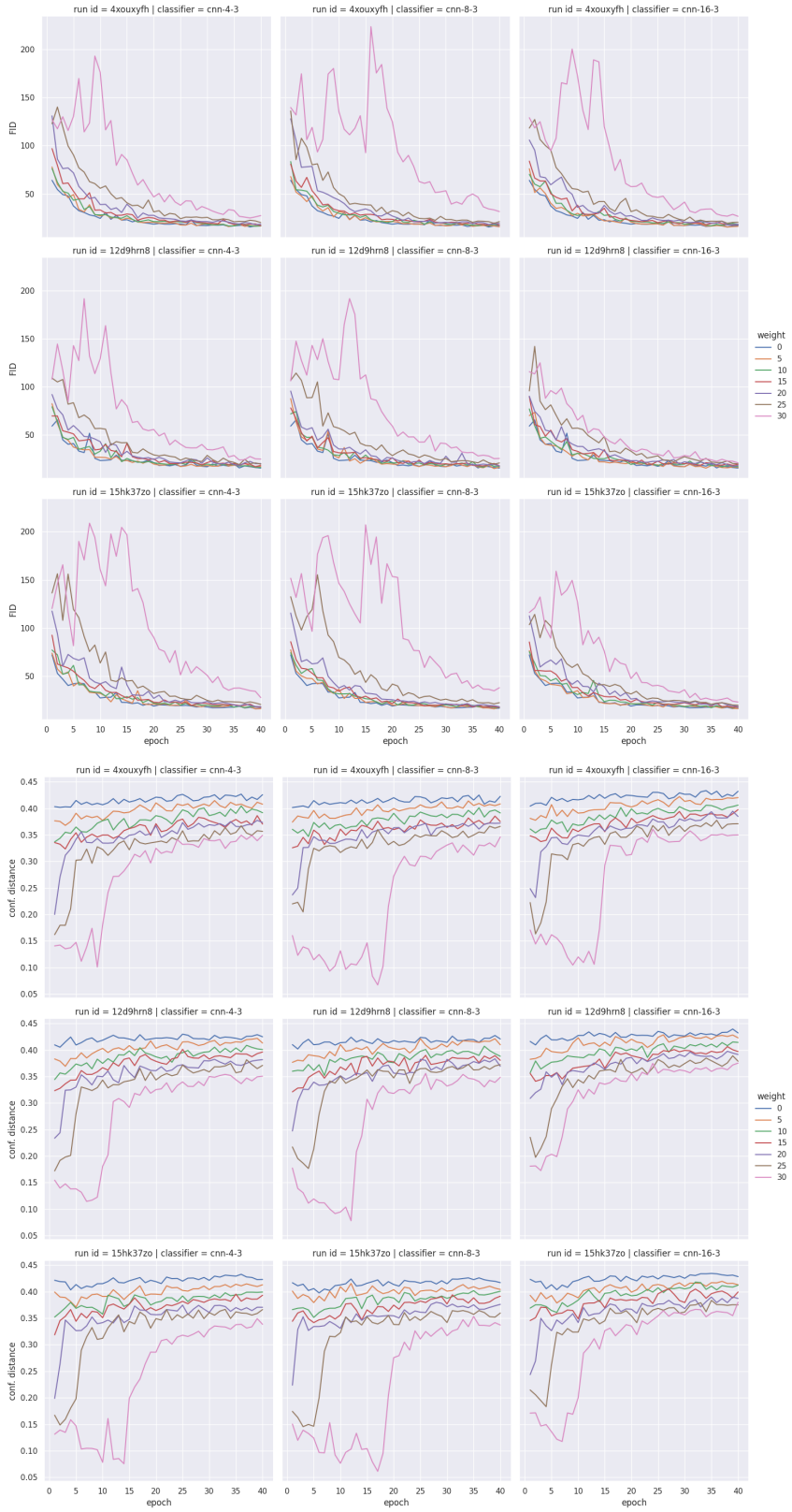


Figure B.39: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST a number of pre-train epochs of 5.

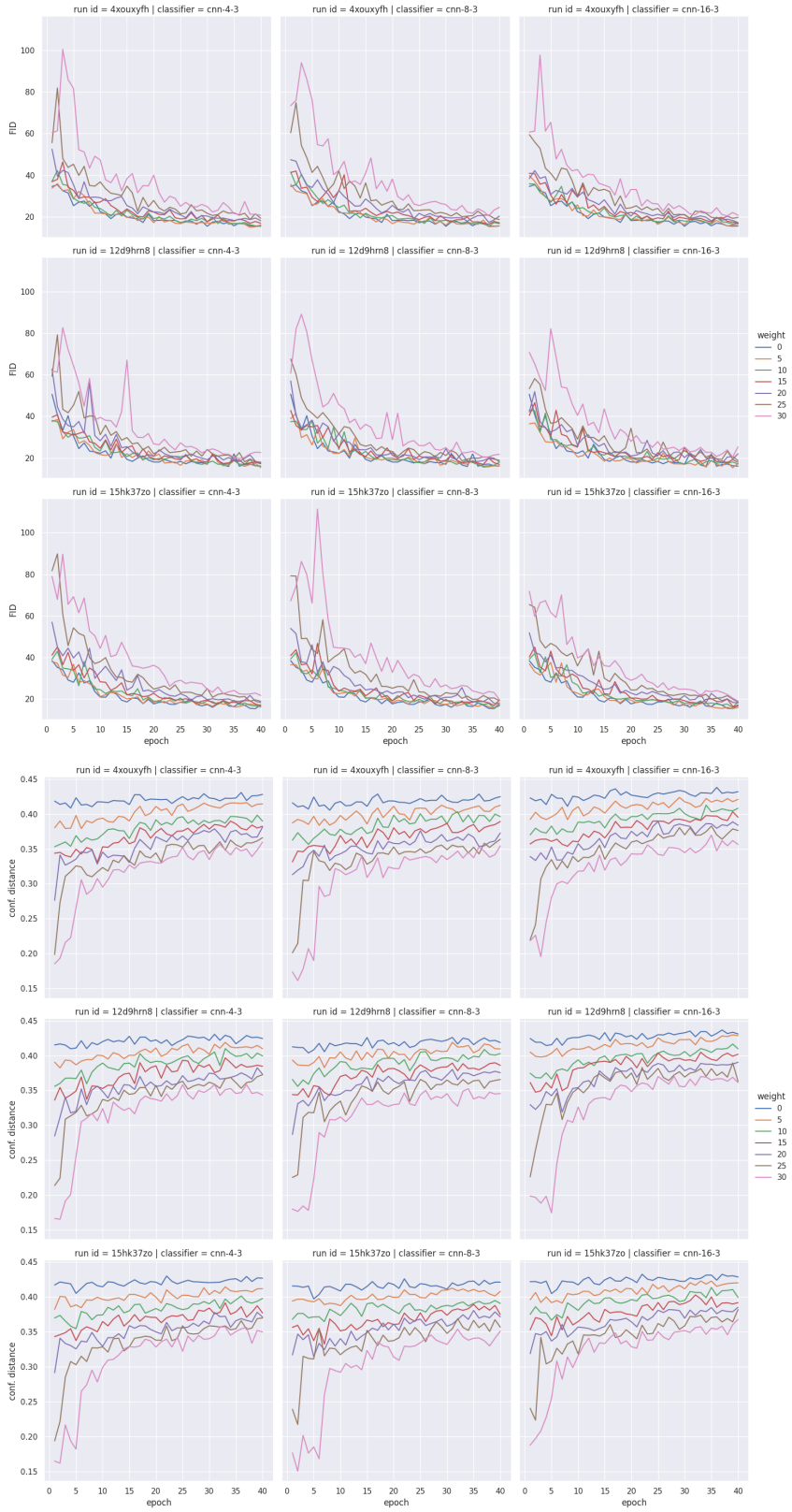


Figure B.40: FID and average confusion distance evolution for classes "Dress" and "T-Shirt/top" of FMNIST using a number of pre-train epochs of 10.

B.2.2 "Sneaker" vs. "Sandal"

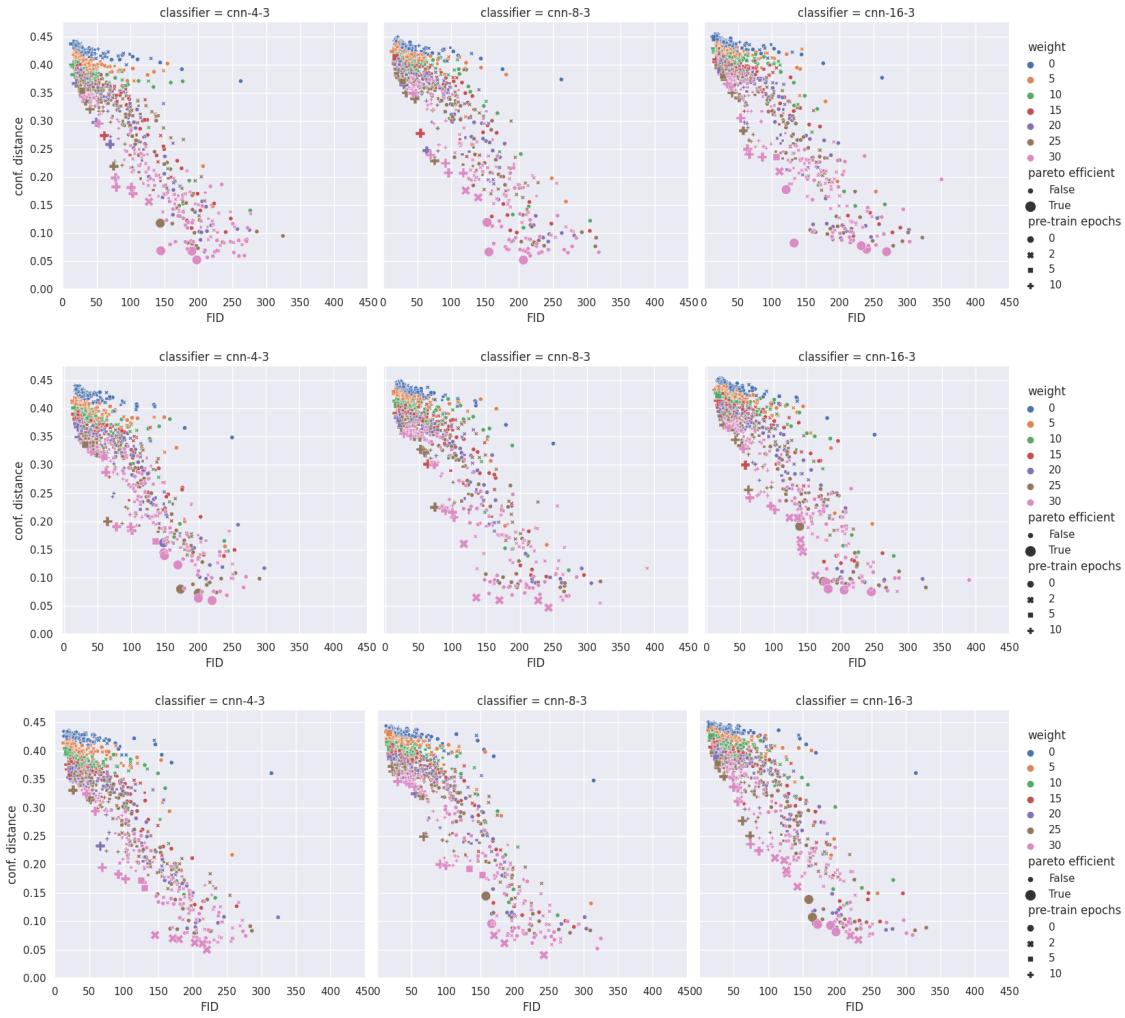


Figure B.41: All metrics gathered using the dataset consisting of examples of classes "Sneaker" and "Sandal" of the Fashion MNIST dataset. Each row corresponds to a run using a different RNG seed.

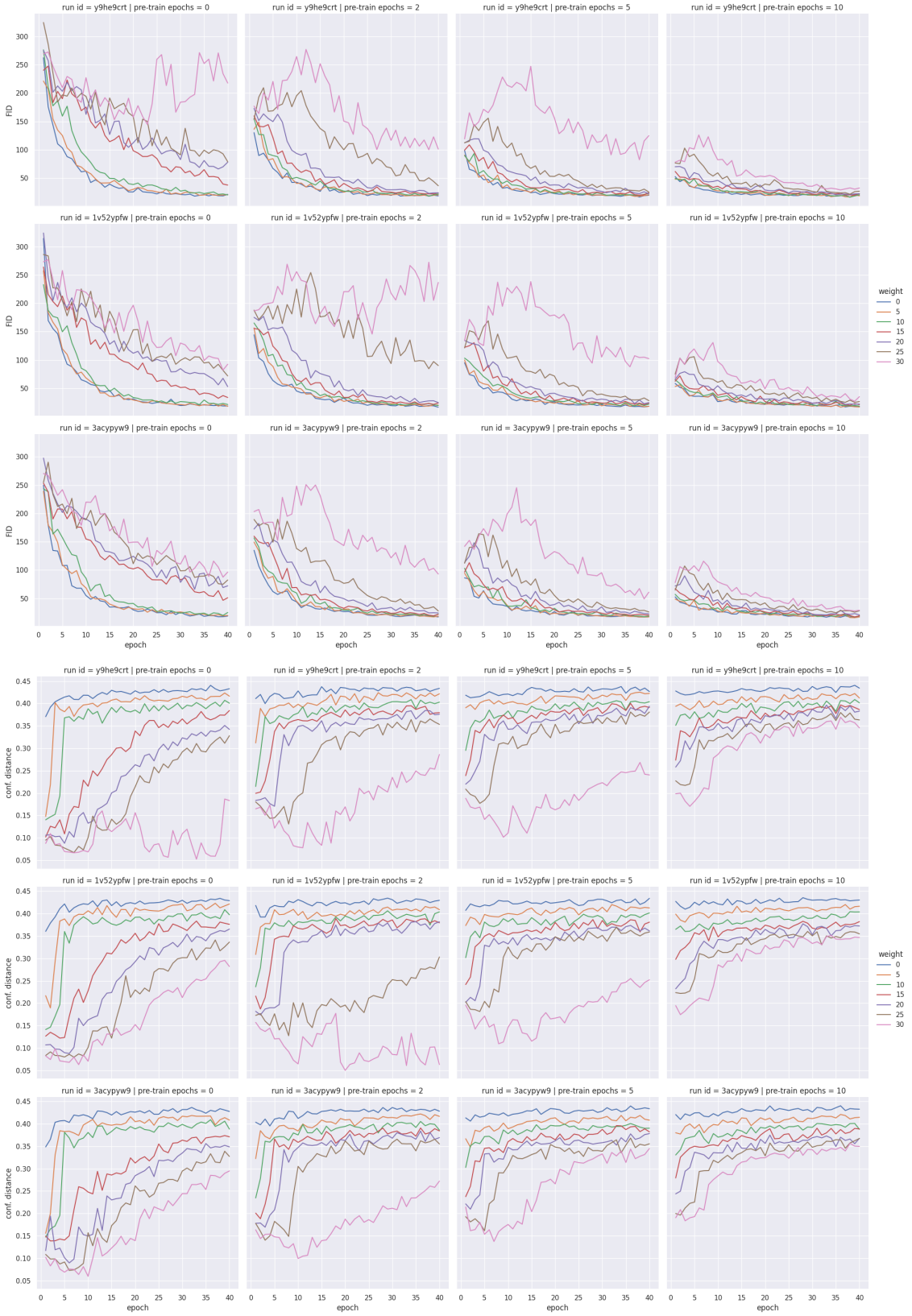


Figure B.42: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST using a classifier with $nf = 4$.

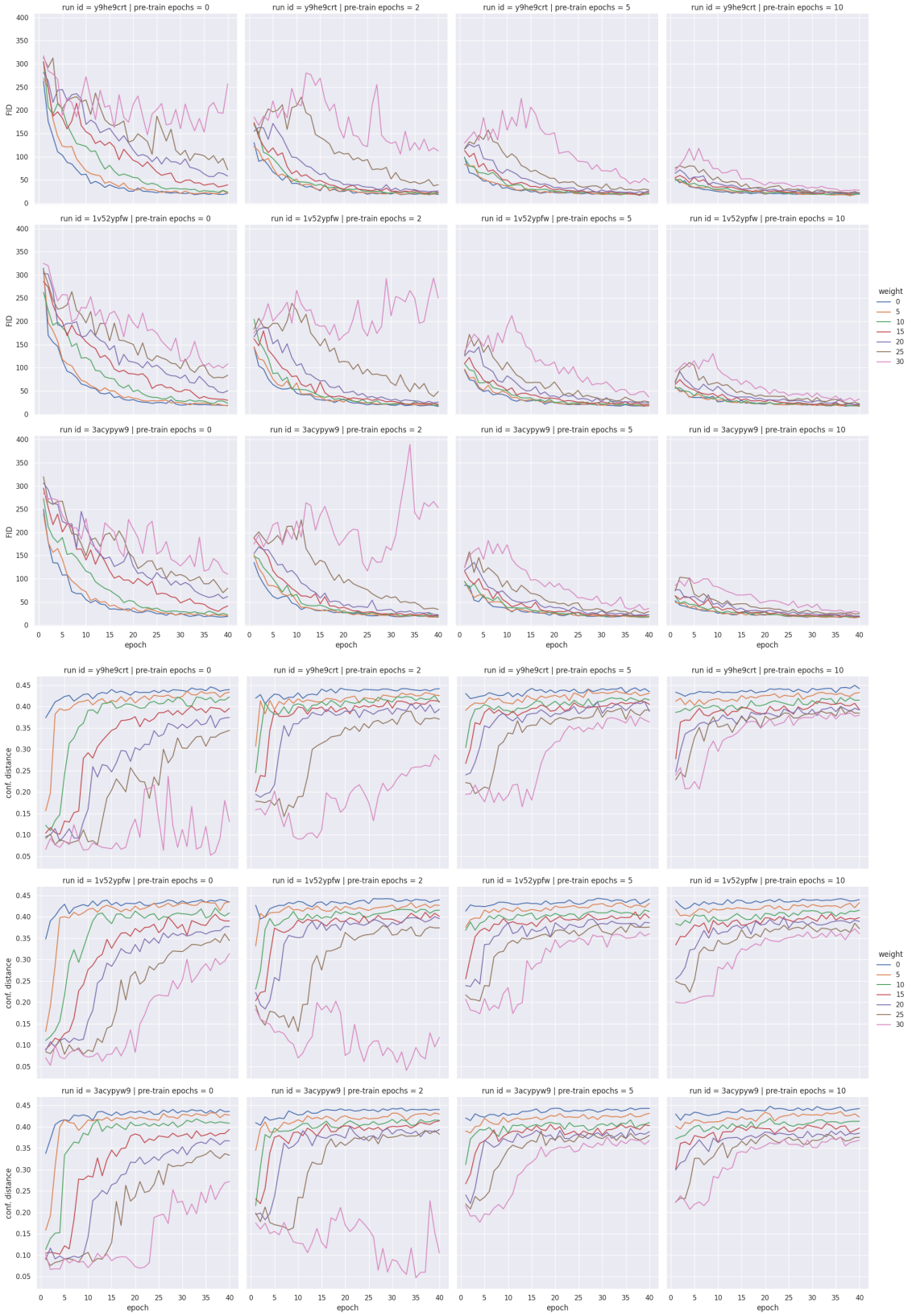


Figure B.43: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST using a classifier with $nf = 2$.

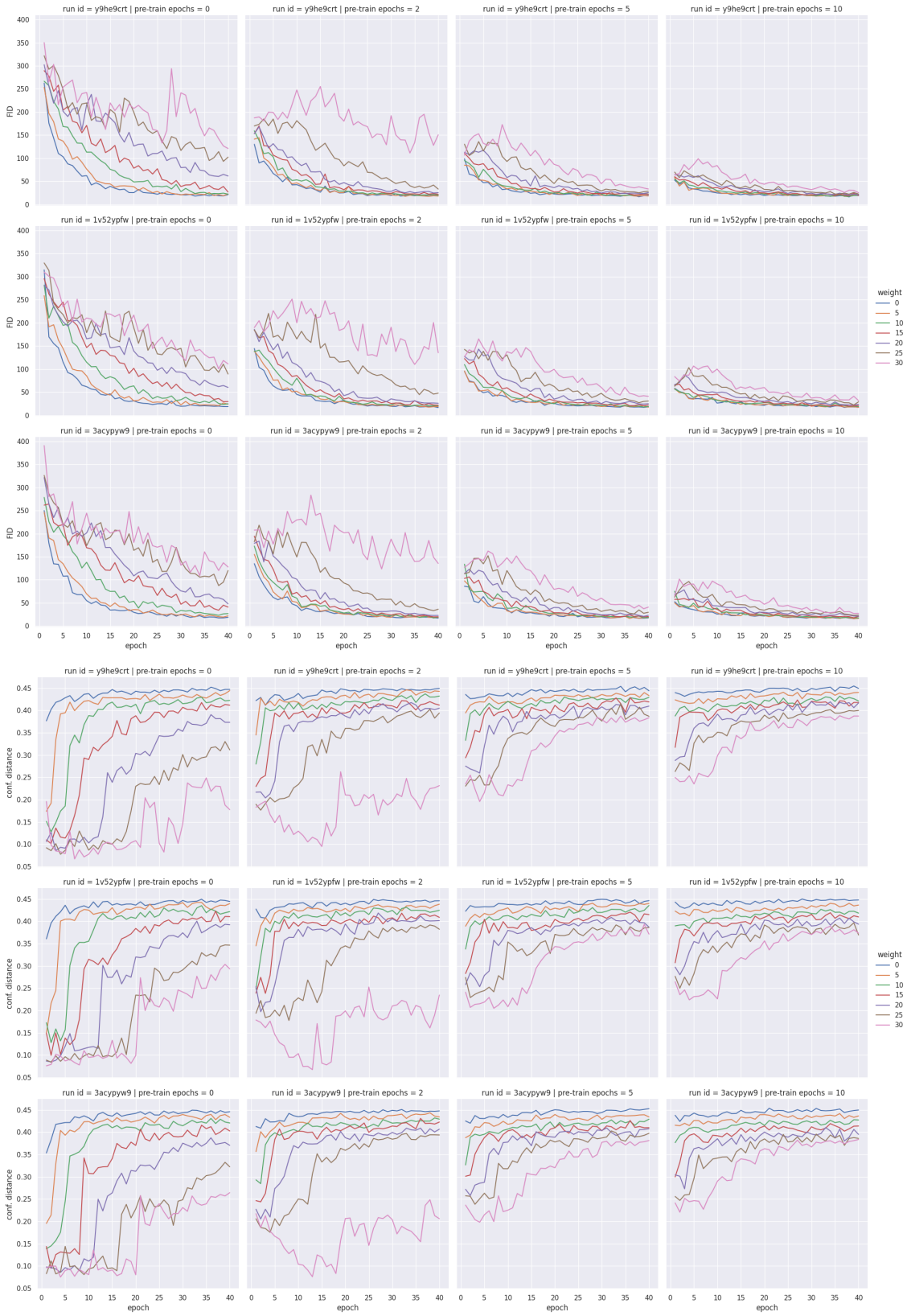


Figure B.44: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST using a classifier with $nf = 16$.

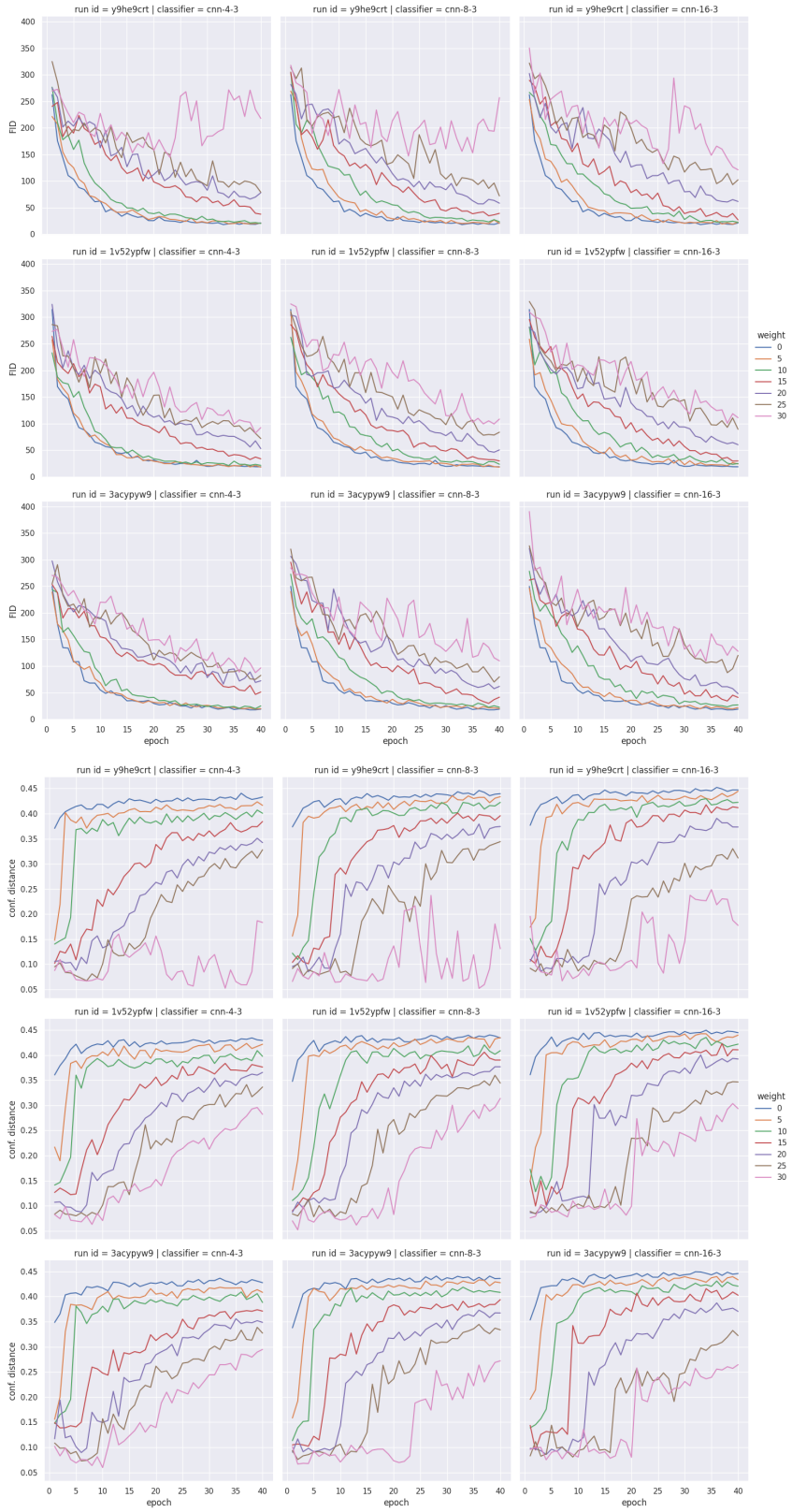


Figure B.45: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST a number of pre-train epochs of 0.

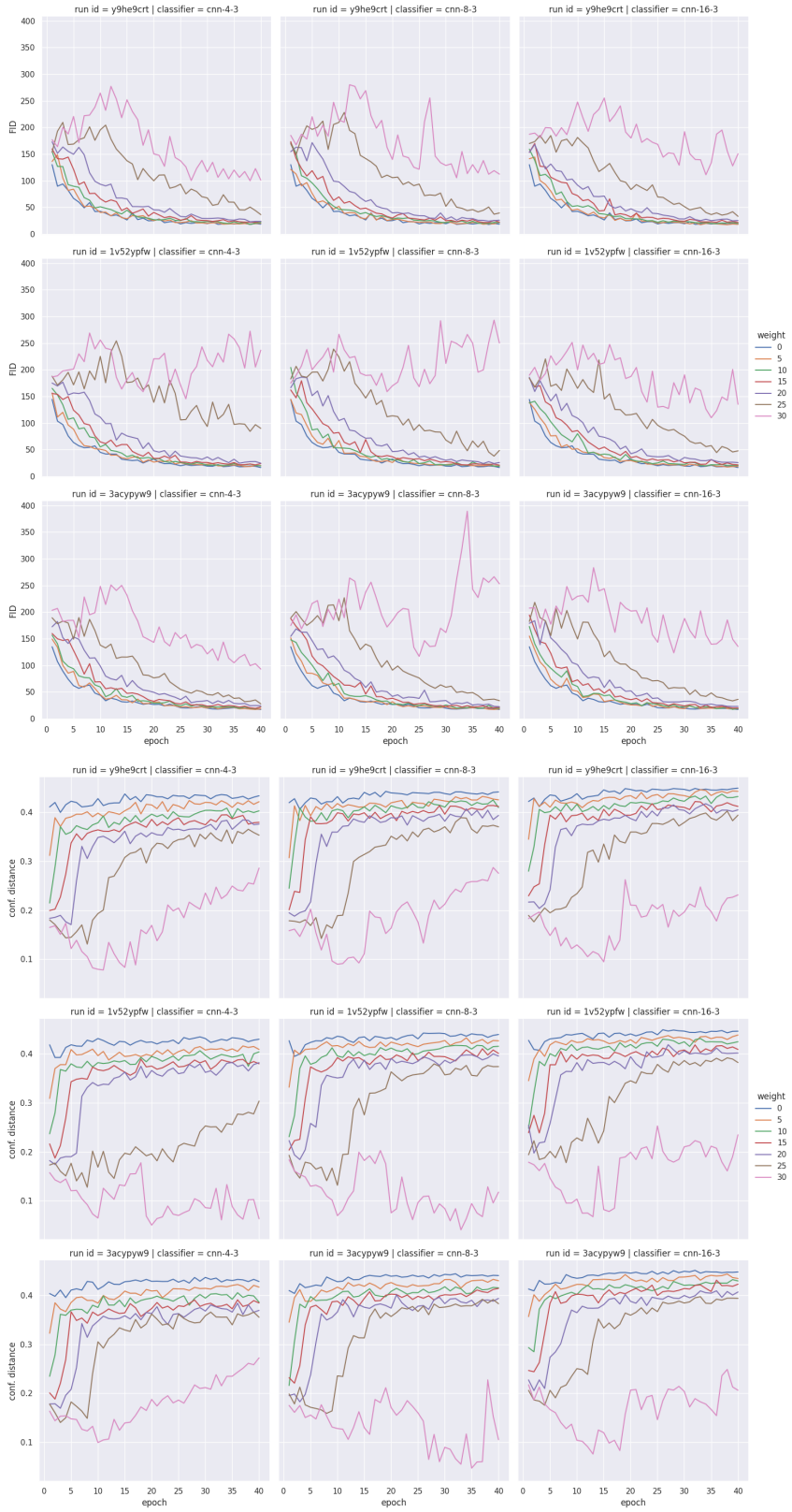


Figure B.46: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST using a number of pre-train epochs of 2.

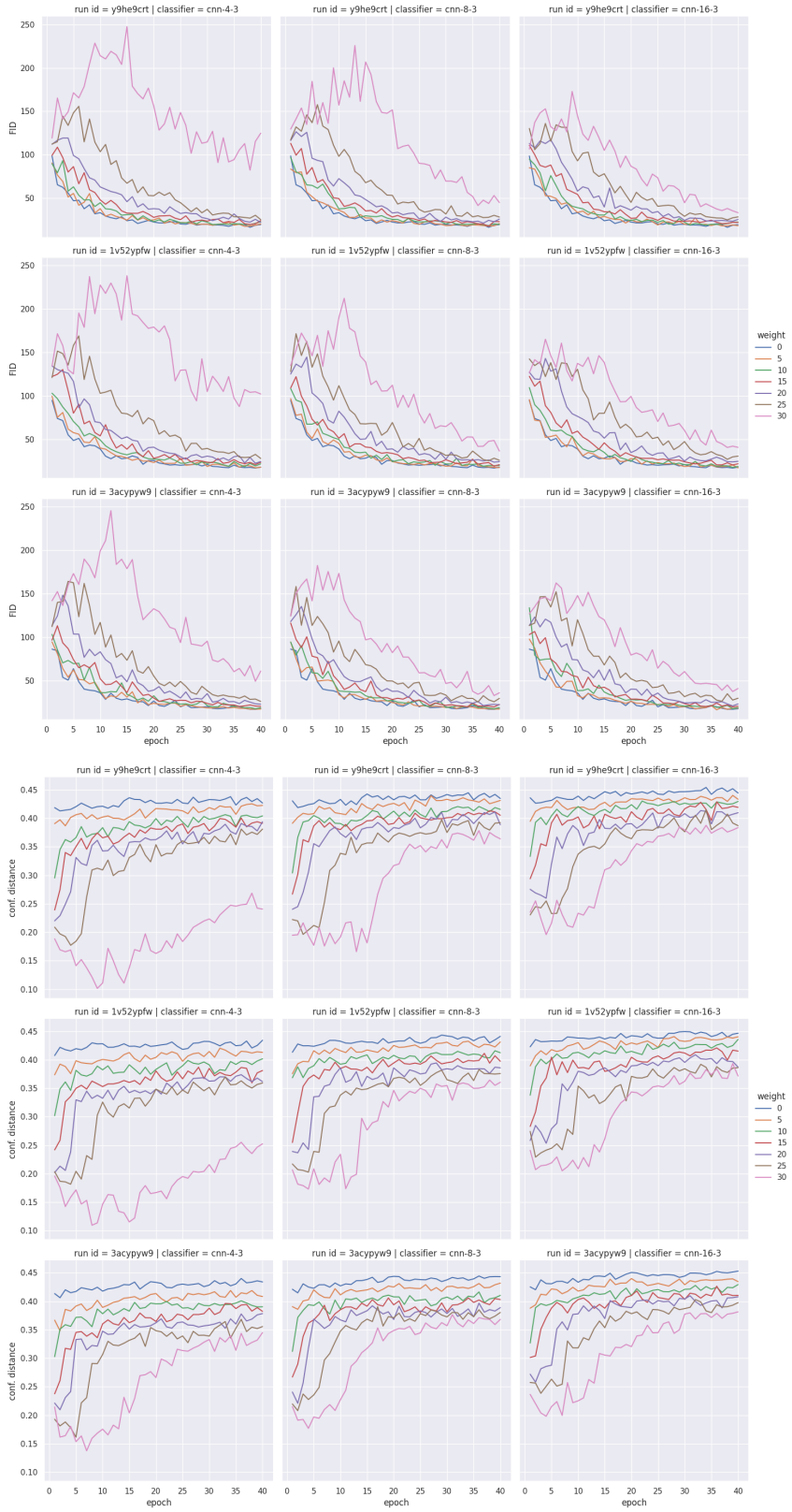


Figure B.47: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST a number of pre-train epochs of 5.

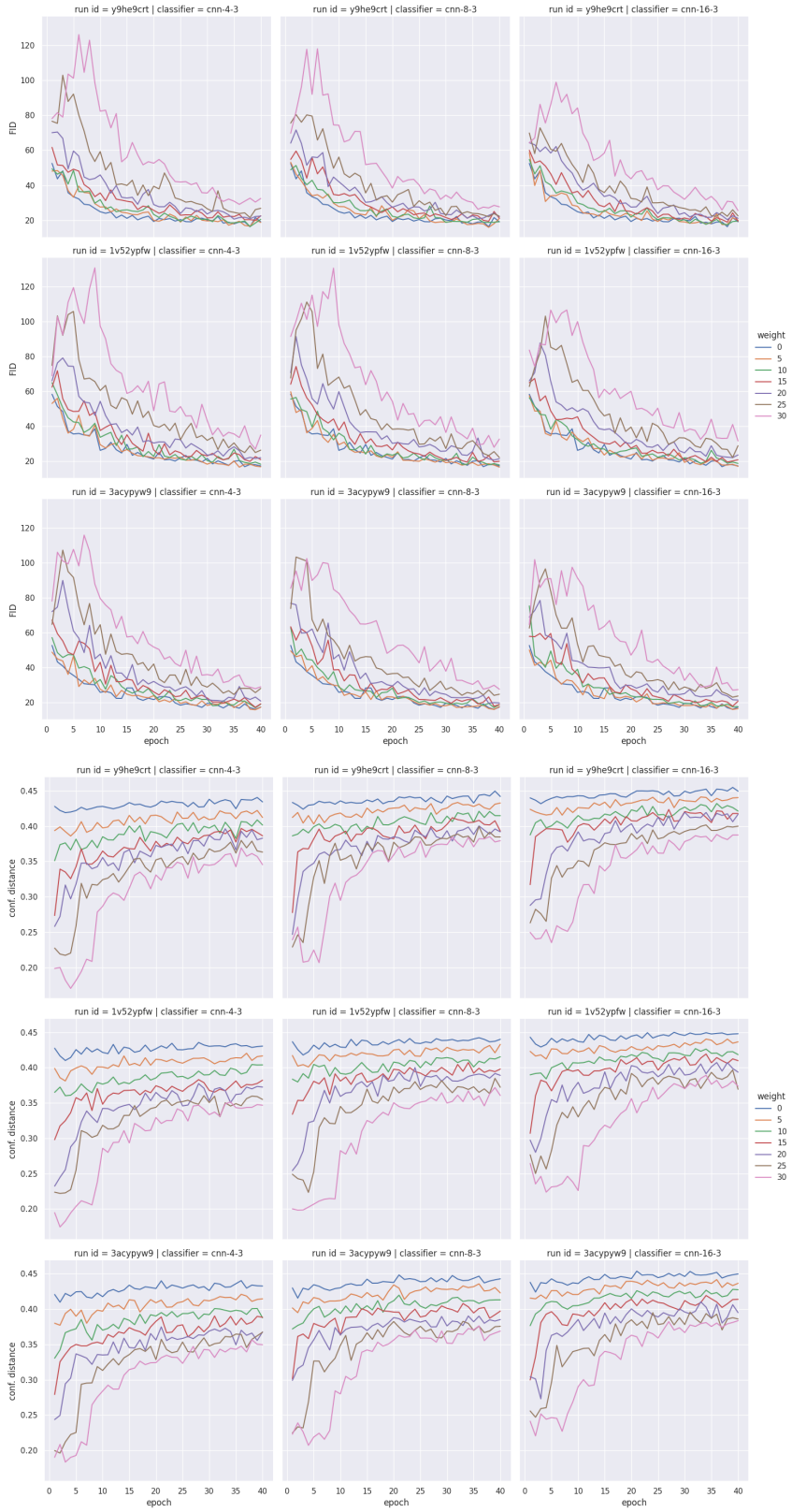


Figure B.48: FID and average confusion distance evolution for classes "Sneaker" and "Sandal" of FMNIST using a number of pre-train epochs of 10.