

# A Survey on GANs for Anomaly Detection

Federico Di Mattia<sup>1\*</sup> Paolo Galeone<sup>1\*</sup> Michele De Simoni<sup>1\*</sup> Emanuele Ghelfi<sup>1\*</sup>

## Abstract

Anomaly detection is a significant problem faced in several research areas. Detecting and correctly classifying something unseen as anomalous is a challenging problem that has been tackled in many different manners over the years. Generative Adversarial Networks (GANs) and the adversarial training process have been recently employed to face this task yielding remarkable results. In this paper we survey the principal GAN-based anomaly detection methods, highlighting their pros and cons. Our contributions are the empirical validation of the main GAN models for anomaly detection, the increase of the experimental results on different datasets and **the public release of a complete Open Source toolbox for Anomaly Detection using GANs.**

## 1. Introduction

Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior (Chandola et al., 2009). Generative Adversarial Networks (GANs) and the adversarial training framework (Goodfellow et al., 2014) have been successfully applied to model complex and high dimensional distribution of real-world data. This GAN characteristic suggests they can be used successfully for anomaly detection, although their application has been only recently explored. Anomaly detection using GANs is the task of modeling the normal behavior using the adversarial training process and detecting the anomalies measuring an *anomaly score* (Schlegl et al., 2017). To the best of our knowledge, all the GAN-based approaches to anomaly detection build upon on the Adversarial Feature Learning idea (Donahue et al., 2016) in which the BiGAN architecture has been proposed. In their original formulation, the GAN framework learns a generator that maps samples from an arbitrary latent distribution (noise prior) to data as well as a discriminator which tries to distinguish between real and generated samples. **The BiGAN architecture extended the original for-**

**mulation, adding the learning of the inverse mapping which maps the data back to the latent representation.** A learned function that maps input data to its latent representation together with a function that does the opposite (the generator) is the basis of the anomaly detection using GANs.

The paper is organized as follows. In Section 1 we introduce the GANs framework and, briefly, its most innovative extensions, namely conditional GANs and BiGAN, respectively in Section 1.2 and Section 1.3. **Section 2 contains the state of the art architectures for anomaly detection with GANs.** In Section 3 we empirically evaluate all the analyzed architectures. Finally, Section 4 contains the conclusions and future research directions.

### 1.1. GANs

GANs are a framework for the estimation of generative models via an adversarial process in which two models, a discriminator  $D$  and a generator  $G$ , are trained simultaneously. The generator  $G$  aim is to capture the data distribution, while the discriminator  $D$  estimates the probability that a sample came from the training data rather than  $G$ . To learn a generative distribution  $p_g$  over the data  $\mathbf{x}$  the generator builds a mapping from a prior noise distribution  $p_z$  to a data space as  $G(\mathbf{z}; \theta_G)$ , where  $\theta_G$  are the generator parameters. The discriminator outputs a single scalar representing the probability that  $\mathbf{x}$  came from real data rather than from  $p_g$ . The generator function is denoted with  $D(\mathbf{x}; \theta_D)$ , where  $\theta_D$  are discriminator parameters.

The original GAN framework (Goodfellow et al., 2014) poses this problem as a min-max game in which the two players ( $G$  and  $D$ ) compete against each other, playing the following zero-sum min-max game:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))]. \quad (1)$$

### 1.2. Conditional GANs

GANs can be extended to a conditional model (Mirza & Osindero, 2014) **conditioning either  $G$  or  $D$  on some extra information  $y$ .** **The  $y$  condition could be any auxiliary information, such as class labels or data from other modalities.** We can perform the conditioning by feeding  $y$  into

\*Equal contribution <sup>1</sup>Zuru Tech, Modena, Italy. Correspondence to: Federico Di Mattia <federico.d@zuru.tech>.

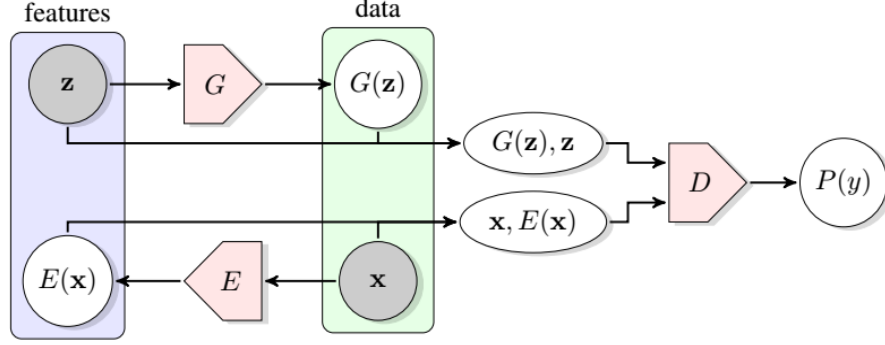


Figure 1. The structure of BiGAN proposed in (Donahue et al., 2016).

both the discriminator and generator as an additional input layer. The generator combines the noise prior  $p_z(\mathbf{z})$  and  $y$  in a joint hidden representation, the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator,  $\mathbf{x}$  and  $y$  are presented as inputs to a discriminative function. The objective function considering the condition becomes:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x}|y)} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|y)))]. \quad (2)$$

### 1.3. BiGAN

Bidirectional GAN (Donahue et al., 2016) extends the GAN framework including an encoder  $E(x; \theta_E)$  that learns the inverse of the generator  $E = G^{-1}$ . The BiGAN training process allows learning a mapping simultaneously from latent space to data and vice versa. The encoder  $E$  is a non-linear parametric function in the same way as  $G$  and  $D$ , and it can be trained using gradient descent. As in the conditional GANs scenario, the discriminator must learn to classify not only real and fake samples, but pairs in the form  $(G(\mathbf{z}), \mathbf{z})$  or  $(\mathbf{x}, E(\mathbf{x}))$ . The BiGAN training objective is:

$$\min_{G, E} \max_D V(D, G, E) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\mathbb{E}_{\mathbf{z} \sim p_{E(\mathbf{x})}} [\log D(\mathbf{x}, \mathbf{z})]] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\mathbb{E}_{\mathbf{x} \sim p_G(\mathbf{x}|\mathbf{z})} [\log(1 - D(\mathbf{x}, \mathbf{z}))]]. \quad (3)$$

Figure 1 depicts a visual structure of the BiGAN architecture.

## 2. GANs for anomaly detection

Anomaly detection using GANs is an emerging research field. Schlegl et al. (2017), here referred to as AnoGAN, were the first to propose such a concept. In order to face the

performance issues of AnoGAN a BiGAN-based approach has been proposed in Zenati et al. (2018), here referred as EGBAD (Efficient GAN Based Anomaly Detection), that outperformed AnoGAN execution time. Recently, Akcay et al. (2018) advanced a GAN + autoencoder based approach that exceeded EGBAD performance from both evaluation metrics and execution speed.

In the following sections, we present an analysis of the considered architecture. The term sample and image are used interchangeably since GANs can be used to detect anomalies on a wide range of domains, but all the analyzed architectures focused mostly on images.

### 2.1. AnoGAN

AnoGAN aim is to use a standard GAN, trained only on positive samples, to learn a mapping from the latent space representation  $\mathbf{z}$  to the realistic sample  $\hat{\mathbf{x}} = G(\mathbf{z})$  and use this learned representation to map new, unseen, samples back to the latent space. Training a GAN on normal samples only, makes the generator learn the manifold  $\mathcal{X}$  of normal samples. Given that the generator learns how to generate normal samples, when an anomalous image is encoded its reconstruction will be non-anomalous; hence the difference between the input and the reconstructed image will highlight the anomalies. The two steps of training and detecting anomalies are summarized in Figure 2.

The authors have defined the mapping of input samples to the latent space as an iterative process. The aim is to find a point  $\mathbf{z}$  in the latent space that corresponds to a generated value  $G(\mathbf{z})$  that is similar to the query value  $\mathbf{x}$  located on the manifold  $\mathcal{X}$  of the positive samples. The research process is defined as the minimization trough  $\gamma = 1, 2, \dots, \Gamma$  backpropagation steps of the loss function defined as the weighted sum of the residual loss  $\mathcal{L}_R$  and discriminator loss  $\mathcal{L}_D$ , in the spirit of Yeh et al. (2016).

The residual loss measures the dissimilarity between the query sample and the generated sample in the input domain

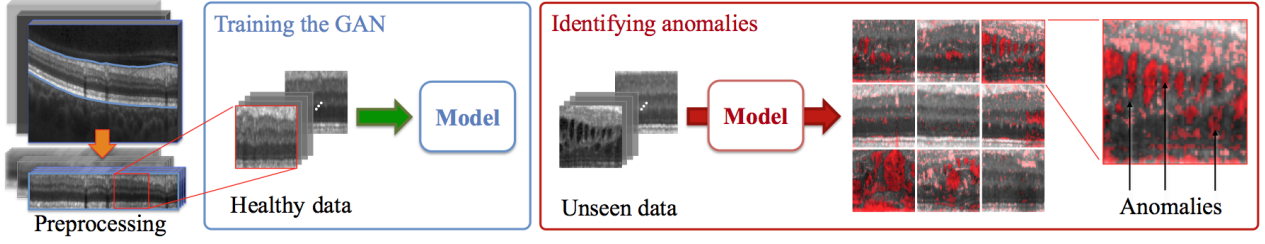


Figure 2. AnoGAN (Schlegl et al., 2017). The GAN is trained on positive samples. At test time, after  $\Gamma$  research iteration the latent vector that maps the test image to its latent representation is found  $\mathbf{z}_\Gamma$ . The reconstructed image  $G(\mathbf{z}_\Gamma)$  is used to localize the anomalous regions.

space:

$$\mathcal{L}_R(\mathbf{z}_\gamma) = \|\mathbf{x} - G(\mathbf{z}_\gamma)\|_1. \quad (4)$$

The *discriminator loss* takes into account the discriminator response. It can be formalized in two different ways. Following the original idea of Yeh et al. (2016), hence feeding the generated image  $G(\mathbf{z}_\gamma)$  into the discriminator and calculating the sigmoid cross-entropy as in the adversarial training phase: this takes into account the discriminator confidence that the input sample is derived by the real data distribution. Alternatively, using the idea introduced by Salimans et al. (2016), and used by the AnoGAN (Schlegl et al., 2017) authors, to compute the feature matching loss, extracting features from a discriminator layer  $\mathbf{f}$  in order to take into account if the generated sample has similar features of the input one, by computing:

$$\mathcal{L}_D(\mathbf{z}_\gamma) = \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(G(\mathbf{z}_\gamma))\|_1, \quad (5)$$

hence the proposed loss function is:

$$\mathcal{L}(\mathbf{z}_\gamma) = (1 - \lambda) \cdot \mathcal{L}_R(\mathbf{z}_\gamma) + \gamma \cdot \mathcal{L}_D(\mathbf{z}_\gamma). \quad (6)$$

Its value at the  $\Gamma$ -th step coincides with the *anomaly score* formulation:

$$A(\mathbf{x}) = \mathcal{L}(\mathbf{z}_\Gamma). \quad (7)$$

$A(\mathbf{x})$  has no upper bound; to high values correspond an high probability of  $\mathbf{x}$  to be anomalous.

It should be noted that the minimization process is required for every single input sample  $\mathbf{x}$ .

#### 2.1.1. PROS AND CONS

##### Pros

- Showed that GANs can be used for anomaly detection.
- Introduced a new mapping scheme from latent space to input data space.

- Used the same mapping scheme to define an anomaly score.

##### Cons

- Requires  $\Gamma$  optimization steps for every new input: bad test-time performance.
- The GAN objective has not been modified to take into account the need for the inverse mapping learning.
- The anomaly score is difficult to interpret, not being in the probability range.

## 2.2. EGBAD

Efficient GAN-Based Anomaly Detection (EGBAD) (Zenati et al., 2018) brings the BiGAN architecture to the anomaly detection domain. In particular, EGBAD tries to solve the AnoGAN disadvantages using Donahue et al. (2016) and Dumoulin et al. (2017) works that allows learning an encoder  $E$  able to map input samples to their latent representation during the adversarial training. The importance of learning  $E$  jointly with  $G$  is strongly emphasized, hence Zenati et al. (2018) adopted a strategy similar to the one indicated in Donahue et al. (2016) and Dumoulin et al. (2017) in order to try to solve, during training, the optimization problem  $\min_{G,E} \max_D V(D, E, G)$  where  $V(D, E, G)$  is defined as in Equation 3. The main contribution of the EGBAD is to allow computing the anomaly score without  $\Gamma$  optimization steps during the inference as it happens in AnoGAN (Schlegl et al., 2017).

## 2.3. GANomaly

Akçay et al. (2018) introduce the GANomaly approach. Inspired by AnoGAN (Schlegl et al., 2017), BiGAN (Donahue et al., 2016) and EGBAD (Zenati et al., 2018) they train a generator network on normal samples to learn their manifold  $\mathcal{X}$  while at the same time an autoencoder is trained to learn how to encode the images in their latent representation efficiently. Their work is intended to improve the ideas of Schlegl et al. (2017), Donahue et al. (2016) and Zenati

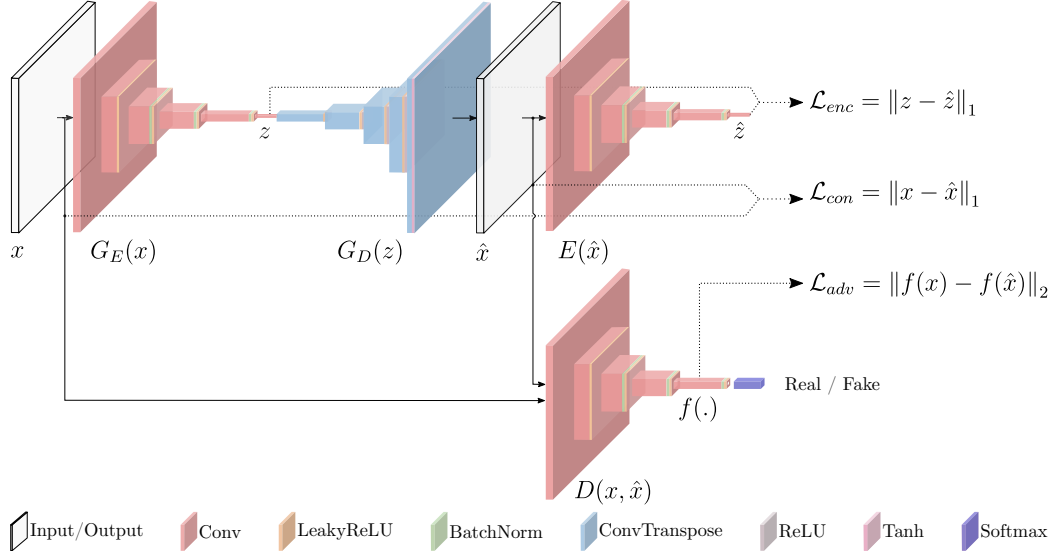


Figure 3. GANomaly architecture and loss functions from (Akçay et al., 2018).

et al. (2018). Their approach only needs a generator and a discriminator as in a standard GAN architecture.

**Generator network** The generator network consists of three elements in series, an encoder  $G_E$ , a decoder  $G_D$  (both assembling an autoencoder structure) and another encoder  $E$ . The architecture of the two encoders is the same.  $G_E$  takes in input an image  $\mathbf{x}$  and outputs an encoded version  $\mathbf{z}$  of it. Hence,  $\mathbf{z}$  is the input of  $G_D$  that outputs  $\hat{\mathbf{x}}$ , the reconstructed version of  $\mathbf{x}$ . Finally,  $\hat{\mathbf{x}}$  is given as an input to the encoder  $E$  that produces  $\hat{\mathbf{z}}$ . There are two main contributions from this architecture. First, the operating principle of the anomaly detection of this work lies in the autoencoder structure. Given that we learn to encode normal (non-anomalous) data (producing  $\mathbf{z}$ ) and given that we learn to generate normal data ( $\hat{\mathbf{x}}$ ) starting from the encoded representation  $\mathbf{z}$ , when the input data  $\mathbf{x}$  is an anomaly its reconstruction will be normal. Because the generator will always produce a non-anomalous image, the visual difference between the input  $\mathbf{x}$  and the produced  $\hat{\mathbf{x}}$  will be high and in particular will spatially highlight where the anomalies are located. Second, the encoder  $E$  at the end of the generator structure helps, during the training phase, to learn to encode the images in order to have the best possible representation of  $\mathbf{x}$  that could lead to its reconstruction  $\hat{\mathbf{x}}$ .

**Discriminator network** The discriminator network  $D$  is the other part of the whole architecture, and it is, with the generator part, the other building block of the standard GAN architecture. The discriminator, in the standard adversarial training, is trained to discern between real and generated data. When it is not able to discern among them, it means that the generator produces realistic images. The generator is continuously updated to fool the discriminator. Refer

to Figure 3 for a visual representation of the architecture underpinning GANomaly.

The GANomaly architecture differs from AnoGAN (Schlegl et al., 2017) and from EGBAD (Zenati et al., 2018). In Figure 4 the three architectures are presented.

Beside these two networks, the other main contribution of GANomaly is the introduction of the generator loss as the sum of three different losses; the discriminator loss is the classical discriminator GAN loss.

**Generator loss** The objective function is formulated by combining three loss functions, each of which optimizes a different part of the whole architecture.

**Adversarial Loss** The adversarial loss it is chosen to be the feature matching loss as introduced in Schlegl et al. (2017) and pursued in Zenati et al. (2018):

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x} \sim p_X} \|f(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim p_X} f(G(\mathbf{x}))\|_2, \quad (8)$$

where  $f$  is a layer of the discriminator  $D$ , used to extract a feature representation of the input. Alternatively, binary cross entropy loss can be used.

**Contextual Loss** Through the use of this loss the generator learns contextual information about the input data. As shown in (Isola et al., 2016) the use of the  $\mathcal{L}_1$  norm helps to obtain better visual results:

$$\mathcal{L}_{con} = \mathbb{E}_{\mathbf{x} \sim p_X} \|\mathbf{x} - G(\mathbf{x})\|_1. \quad (9)$$

**Encoder Loss** This loss is used to let the generator network learn how to best encode a normal (non-anomalous) image:

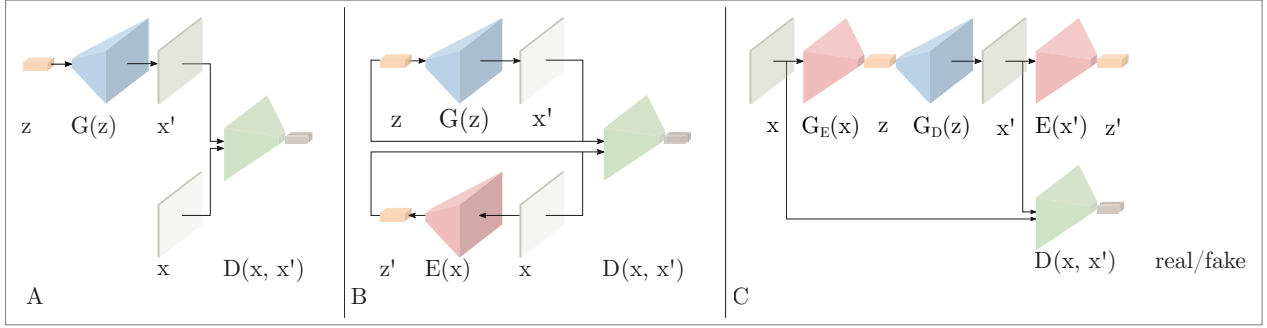


Figure 4. Architectures comparison. A) AnoGAN (Schlegl et al., 2017), B) EGBAD (Zenati et al., 2018), C) GANomaly (Akçay et al., 2018).

$$\mathcal{L}_{enc} = \mathbb{E}_{\mathbf{x} \sim p_X} \|\mathcal{G}_E(\mathbf{x}) - E(\mathcal{G}(\mathbf{x}))\|_2. \quad (10)$$

The resulting generator loss will be the result of the (weighted) sum of the three previously defined losses:

$$\mathcal{L} = w_{adv}\mathcal{L}_{adv} + w_{con}\mathcal{L}_{con} + w_{enc}\mathcal{L}_{enc}, \quad (11)$$

where  $w_{adv}$ ,  $w_{con}$  and  $w_{enc}$  are weighting parameters used to adjust the importance of the three losses.

At test stage, the authors proposed to compute the anomaly score in a different way respect to AnoGAN: only using using  $\mathcal{L}_{enc}$ :

$$\mathcal{A}(\mathbf{x}) = \|\mathcal{G}_E(\mathbf{x}) - E(\mathcal{G}(\mathbf{x}))\|_2. \quad (12)$$

In order to make the anomaly score easier to interpret, they proposed to compute the anomaly score for every sample  $\hat{\mathbf{x}}$  in the test set  $\hat{D}$ , getting a set of individual anomaly scores:  $\mathcal{S} = \{s_i : \mathcal{A}(\hat{x}_i), \hat{x}_i \in \hat{D}\}$  and then apply feature scaling to have the anomaly scores within the probabilistic range of  $[0, 1]$ :

$$s'_i = \frac{s_i + \min(S)}{\max(S) - \min(S)}. \quad (13)$$

### 2.3.1. PROS AND CONS

#### Pros

- An encoder is learned during the training process, hence we don't have the need for a research process as in AnoGAN (Schlegl et al., 2017).
- Using an autoencoder like architecture (no use of noise prior) makes the entire learning process faster.
- The anomaly score is easier to interpret.

- The contextual loss can be used to localize the anomaly.

#### Cons

- It allows to detect anomalies both in the image space and in the latent space, but the results couldn't match: a higher anomaly score, that's computed only in the latent space, can be associated with a generated sample with a low contextual loss value and thus very similar to the input - and vice versa.
- Defines a new anomaly score.

## 3. Experiments

To evaluate the performance of every Anomaly Detection algorithm described we re-implemented all of them using the popular deep learning framework Tensorflow (Abadi et al., 2015) creating a publicly available toolbox for the Anomaly Detection with GANs.

Experiments were made trying to improve, where possible (i.e., where there were known errors communicated directly by the authors), the code. The results shown in the following sections are the best empirically obtained among all the tests carried out and do not necessarily derive from the configurations of the networks that at a theoretical level should be the correct ones. In particular, as described in Appendix A, in version 1 (Appendix A.1) of our tests, with the correct implementation of the BiGAN/EGBAD models, we obtained worse results than those obtained in version 2 (Appendix A.2) of our tests with the architecture already implemented in the reference paper, although this contained known errors.

### 3.1. Experimental setup

The experiments are performed using an Intel® Core™ i7-7820X CPU and NVIDIA® GTX 1080 Ti.



BiGAN/EGBAD and GANomaly performance comparison on MNIST and Fashion-MNIST datasets

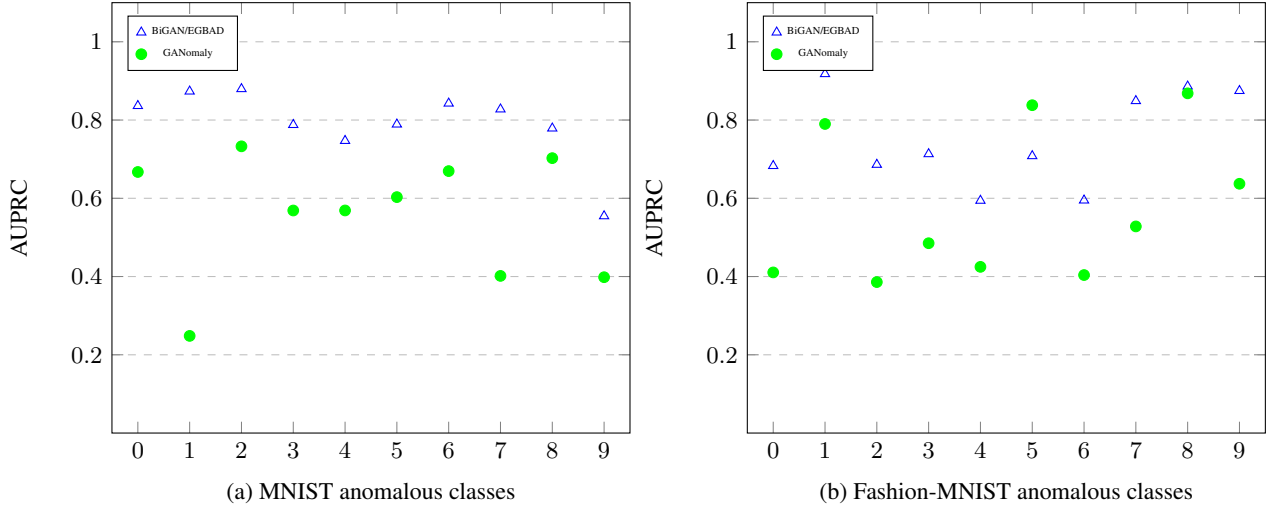


Figure 5. Performance on MNIST (a) and Fashion-MNIST (b) of BiGAN/EGBAD and GANomaly models measured by the area under the precision-recall curve (AUPRC). The results have been selected as the best result between different type of training (with bce/fm and with residual loss). Best viewed in color.

### 3.1.1. DATASETS

We decided to train and test on widely known datasets commonly found in the literature: MNIST (LeCun & Cortes, 2010a), FashionMNIST (Xiao et al., 2017), CIFAR-10 (Krizhevsky et al., 2007), and KDD99 (Lichman, 1999).

**MNIST:** MNIST dataset (LeCun & Cortes, 2010b) is a database of handwritten digits. It consists of  $28 \times 28$  pixels grayscale images split in a training set of 60,000 examples and a test set of 10,000 examples. The output classes are 10 in total and represent the ten digits from 0 to 9. In order to replicate the results of the papers presented in this work, the training process and the subsequent testing phase have been evaluated on each class, i.e., one at a time, only one class has been considered as the anomaly class, and the remaining nine classes have been considered together as the normal data.

**Fashion MNIST:** Fashion-MNIST dataset (Xiao et al., 2017) is a dataset intended to be a more complex drop-in replacement for the traditional MNIST dataset, developed by Zalando® Research Group. It shares with the MNIST dataset the same image size and train-test split; it consists of  $28 \times 28$  pixels images split in a training set of 60,000 examples and a test set of 10,000 examples. Each image is a grayscale image associated, as the MNIST dataset, with a label from 10 classes, from 0 to 9. Each example represents a different item of clothing. As for the MNIST dataset, the training and the following test phase were conducted on each class, one at a time.

**CIFAR-10:** A dataset of tiny images of various subjects,

CIFAR-10 consists of 60000  $32 \times 32$  color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

**KDD:** KDD dataset (Lichman, 1999) consists in a collection of network intrusion detection data, this is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between “bad” connections, called intrusions or attacks, and “good” normal connections. All the examples are string tuples or numbers. Given the considerable amount of data inside the data set, we used as in (Zenati et al., 2018), the KDDCUP99 10 percent version of the data set. Since the anomalies are in a higher percentage than normal data, the normal ones have been considered as anomalies.

### 3.1.2. METHODOLOGY

We follow the same methodology employed by Zenati et al. in the official code accompanying Zenati et al. (2018).

We start by getting all the dataset together (train and test split). Starting from this one big pool of examples, we choose one class as an anomaly and, after shuffling the dataset, we then create a training set by using 80% of the whole data while the remaining 20% is used for the testing set, this process is repeated for all the classes in the dataset. Each model is trained accordingly to its original implementation on the training set and is tested on the whole dataset

	Train			Test	
	BCE	FM	Residual	BCE	FM
Case 1	✓		✓	✓	✓
Case 2		✓	✓	✓	✓
Case 3	✓			✓	✓
Case 4		✓		✓	✓

Table 1. BiGAN - Training and testing configuration combinations.

made up of both regular and anomalous data.

### 3.2. Results

All tests have been made measuring the area under precision and recall curve (AUPRC). For a complete understanding of all the results, please refer to Appendix A. Our fully static-Tensorflow implementation (i.e., only Tensorflow framework has been used, we relied upon neither Keras nor eager execution) produced different results from the one depicted in the original Donahue et al. (2016), Schlegl et al. (2017), and Akcay et al. (2018) papers. The tests have been made by training the GAN networks with different hyper-parameters configurations in order to test a broader range of models configurations. Moreover, following the GANomaly approach, we have added an evaluation process to make model selection and thus select the best model during the training phase. The model selection has been made for BiGAN and GANomaly, since doing it for AnoGAN would be unfeasible, due to the  $\Gamma$  research steps required for every sample at test-time. We intentionally left out the performance evaluation of the AnoGAN model. Due to the inner workings of the architecture that should have required a very long time to be tested because of the necessity to find, every time and for every image taken into consideration, the best latent representation, meaning that for every image we would need 500 training step (500 it is an empirical value found in the original paper (Schlegl et al., 2017)). Following, the training and test methods are described. A summary of the results is present in Figure 5 for the MNIST and Fashion-MNIST dataset, and in Table 3 for the KDD results.

**BiGAN/EGBAD:** With BiGAN/EGBAD (Zenati et al., 2018) architecture we have executed the highest number of training and testing configurations, this means we have performed, for every label the combinations depicted in Table 1. To better understand what the table shows, taking the first row (Case 1) as an example, we are describing the case where, during the training phase, a Binary Cross Entropy loss (BCE) in combination with a Residual loss has been used and, during the testing phase, we computed the anomaly score twice, using the feature matching (FM) and the BCE. All the other cases should be clear. In particular, during the training phase, the choice between using BCE or

	Train		Test	
	BCE	FM	BCE	FM
Case 1	✓		✓	✓
Case 2		✓	✓	✓

Table 2. GANomaly - Training and testing configuration combinations.

	KDD		
	Precision	Recall	F1-Score
BiGAN/EGBAD	<b>0.941174</b>	<b>0.956155</b>	<b>0.948605</b>
GANomaly	0.830256	0.841112	0.835648

Table 3. The performances of BiGAN/EGBAD and GANomaly models on the KDD dataset.

FM influences only the generator loss. The residual loss that it is intended as the difference between the original image and the image reconstructed by the generator starting from a latent representation (please refer to Equation (4)) when used has been added as an additional term to the BCE loss of the encoder. In Figure 5 is possible to see the results on the MNIST and Fashion-MNIST datasets. In Appendix A it is possible to see the complete results. The images used are the original  $28 \times 28$  images. Any additional result from multiple different tests performed with different random seeds has been omitted for the sake of clarity.

**GANomaly:** GANomaly training and testing have been done similarly to the previously described BiGAN/EGBAD architecture. The whole procedure is leaner here, with only the combinations shown in Table 2.

We added a step of model selection during the training phase in order to always save the very best model. For this architecture, the testing phase has been done using an anomaly score equal to the squared difference between the latent representations of the image encoded first with autoencoder part of the network and, after being reconstructed, encoded again with the encoder. To briefly review the working of GANomaly, see Figure 3. The images used are the original ones resized to  $32 \times 32$ . Refer to the plot on the right of Figure 5 to a brief summary of the results on MNIST and Fashion-MNIST datasets and on Table 3 for the results on KDD dataset.

**AnoGAN:** As previously stated, the results of AnoGAN have not been reproduced due to the onerous computational requirements causing the train and test phases to be intensively time-consuming. The reader could refer to the original paper (Schlegl et al., 2017) to an overview of the results.

## 4. Conclusions

We implemented and evaluated the state-of-the-art algorithms for anomaly detection based on GANs. In this work, we unified the three major works under the same framework (Tensorflow) and within the same code-base. The analysis and implementation of the algorithms allowed us to verify the effectiveness of the GANs-based approach to the anomaly detection problem and, at the same time, highlight the differences between the original papers and the publicly available code. Besides, to test the feasibility of the architectures mentioned above, this work intends to provide a modular and ready-to-go solution for everyone desiring an anomaly detection toolkit working out of the box.

## References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Man, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Vigas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- Akcay, S., Abarghouei, A. A., and Breckon, T. P. GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training. abs/1805.06725, 2018. URL <http://arxiv.org/abs/1805.06725>.
- Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. 41(3), 2009.
- Donahue, J., Krhenbhl, P., and Darrell, T. Adversarial Feature Learning. abs/1605.09782, 2016. URL <http://arxiv.org/abs/1605.09782>.
- Dumoulin, V., Belghazi, M. I. D., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. Adversarially learned inference. 2017. URL <http://arXiv.org/abs/1606.00704>.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Nets. pp. 2672–2680, 2014. URL <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.
- Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. abs/1611.07004, 2016. URL <http://arxiv.org/abs/1611.07004>.
- Krizhevsky, A., Nair, V., and Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). 2007. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010a. URL <http://yann.lecun.com/exdb/mnist/>.
- LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010b. URL <http://yann.lecun.com/exdb/mnist/>.
- Lichman, M. KDD Cup 1999 Data (University of California Irvine). 1999. URL <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.
- Mirza, M. and Osindero, S. Conditional Generative Adversarial Nets. abs/1411.1784, 2014. URL <http://arxiv.org/abs/1411.1784>.
- Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved Techniques for Training GANs. abs/1606.03498, 2016. URL <http://arxiv.org/abs/1606.03498>.
- Schlegl, T., Seebeck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. abs/1703.05921, 2017. URL <http://arxiv.org/abs/1703.05921>.
- Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. abs/1708.07747, 2017. URL <http://dblp.uni-trier.de/db/journals/corr/corr1708.html#abs-1708-07747>.
- Yeh, R. A., Chen, C., Lim, T.-Y., Hasegawa-Johnson, M., and Do, M. N. Semantic Image Inpainting with Perceptual and Contextual Losses. abs/1607.07539, 2016. URL <http://arxiv.org/abs/1607.07539>.
- Zenati, H., Foo, C. S., Lecouat, B., Manek, G., and Chandrasekhar, V. R. Efficient GAN-Based Anomaly Detection. abs/1802.06222, 2018. URL <http://arxiv.org/abs/1802.06222>.



## A. Additional Results

In this section, we present more detailed results regarding the different combinations of training-testing pipelines of the BiGAN/EGBAD and GANomaly architectures on MNIST and Fashion-MNIST; results of the CIFAR10 dataset, trained with GANomaly are presented too. Moreover, the results of the addition of a function (namely, a leaky relu activation function in the first layer of the encoder) that was missing from inside the EGBAD published code architecture will be introduced. Surprisingly, the tests have led to the result that without that activation function the network performs better.

### A.1. Version 1

Before getting the results shown in Appendix A.2 we have employed a different test pipeline with a different architecture of the BiGAN network. The main difference was mainly related to:

- **Testing phase** The testing phase is applied twice (for both BiGAN/EGBAD and GANomaly architectures). Once during the model selection during training, and once after the training was finished as a stand alone phase.
- **Activation function** A Leaky Relu function, erroneously missing from the original EGBAD (Zenati et al., 2018) work has been here inserted after the first convolutional layer of the encoder model.

A brief visual understanding of the configuration cases for the BiGAN/EGBAD architecture it is shown in Table 4 below. The configuration is similar to the one shown in Table 1 with the difference that, here, the testing phase has been done separately. The same is true for the GANomaly implemented architecture, shown in Table 5.

	Train			Test	
	BCE	FM	Residual	BCE	FM
Case 1	✓		✓	✓	
Case 2	✓		✓		✓
Case 3		✓	✓	✓	
Case 4		✓	✓		✓
Case 5	✓			✓	
Case 6	✓				✓
Case 7		✓		✓	
Case 8		✓			✓

Table 4. BiGAN - Training and testing configuration combinations.

	Train		Test	
	BCE	FM	BCE	FM
Case 5	✓		✓	
Case 6	✓			✓
Case 7		✓	✓	
Case 8		✓		✓

Table 5. GANomaly - Training and testing configuration combinations.

### A.1.1. MNIST

Here, we present the results on the MNIST dataset. We show in Figure 6 the results of the BiGA/EGBAD architecture and in Figure 7 the results of the GANomaly architecture. In the following, MNIST results are depicted.

**MNIST Tests (w and w/o residual loss)**

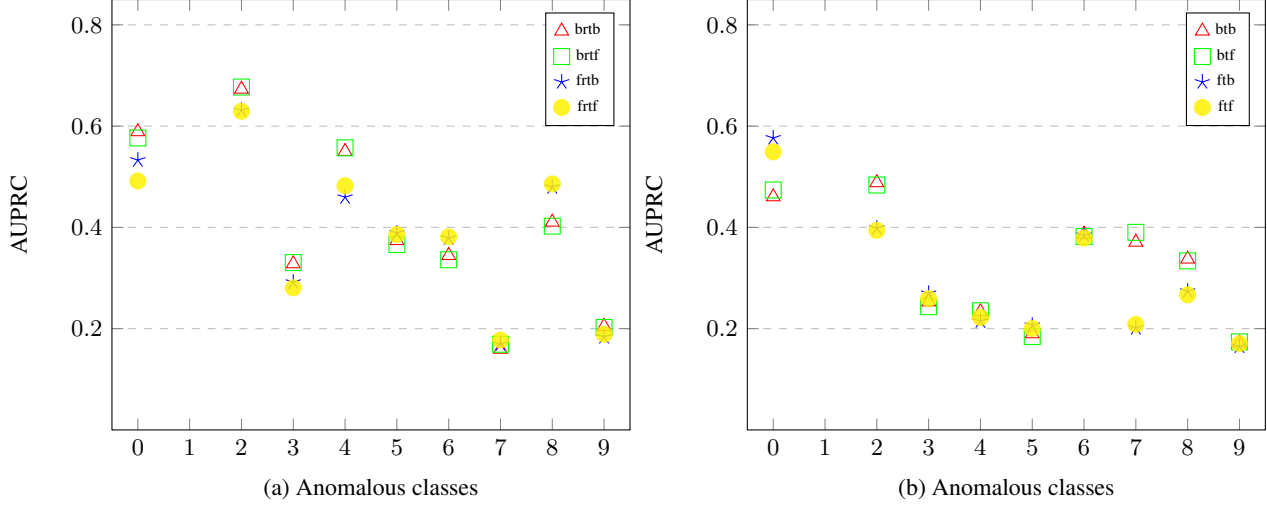


Figure 6. Performance of version 1 of BiGA/EGBAD on MNIST measured by the area under the precision-recall curve (AUPRC). Best viewed in color. Image (a) depicts the results using the residual loss, image (b) it is without the residual loss. "b(r)tb": bce (+ residual loss) trained and bce tested, "b(r)tf": bce (+ residual loss) trained and fm tested, "f(r)tb": fm (+ residual loss) trained and bce tested, "f(r)tf": fm (+ residual loss) trained and fm tested

**GANomaly test results on MNIST dataset**

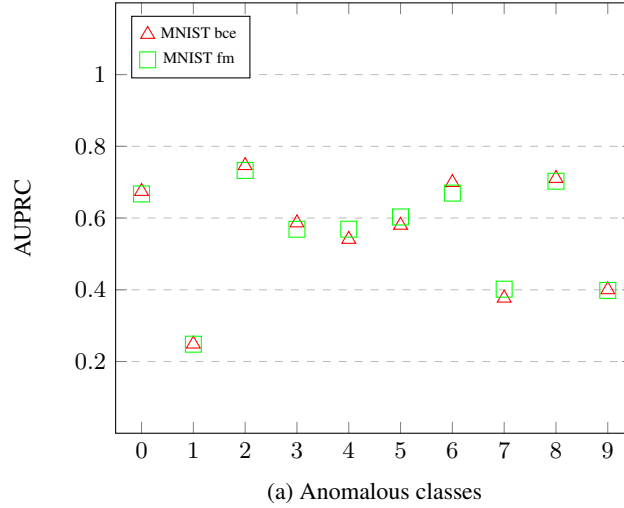


Figure 7. GANomaly results on MNIST. All tests are here performed through a bce metric and measured by the area under the precision-recall curve (AUPRC). Training is done with a bce loss and a fm loss.

### A.1.2. FASHION-MNIST

Here we present the results obtained on the Fashion-MNIST dataset. In Figure 8 we show the results of BiGAN/EGBAD architectures and in Figure 9 the results of the GANomaly architecture.

**Fashion-MNIST Tests (w and w/o residual loss)**

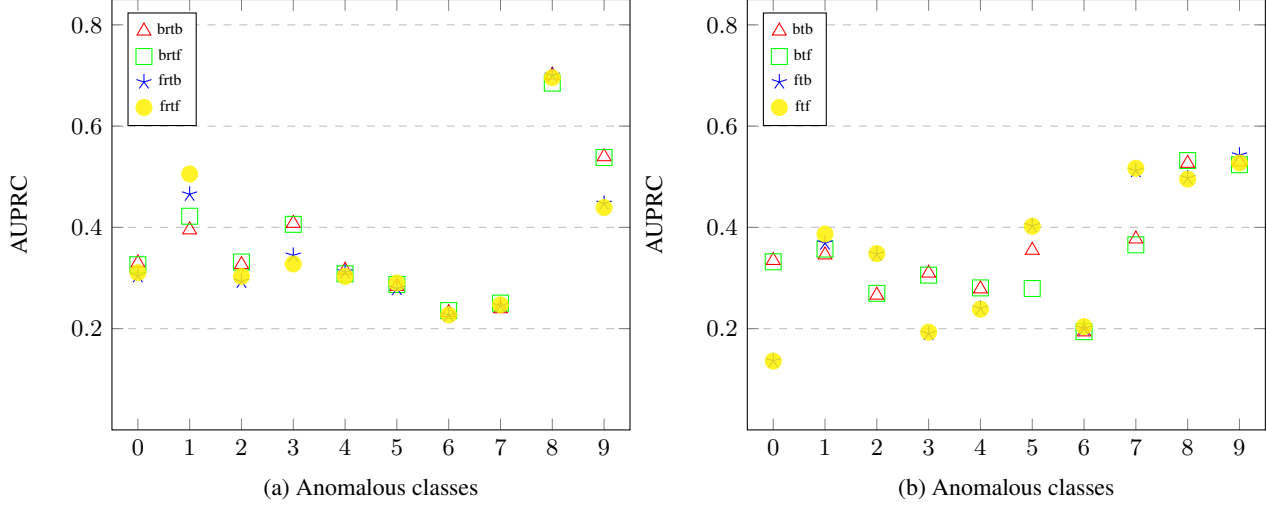


Figure 8. Performance of version 1 of BiGAN/EGBAD on Fashion-MNIST measured by the area under the precision-recall curve (AUPRC). Best viewed in color. Image (a) depicts the results using the residual loss, image (b) it is without the residual loss. "b(r)tb": bce (+ residual loss) trained and bce tested, "b(r)tf": bce (+ residual loss) trained and fm tested, "f(r)tb": fm (+ residual loss) trained and bce tested, "f(r)tf": fm (+ residual loss) trained and fm tested

**GANomaly test results on Fashion-MNIST dataset**

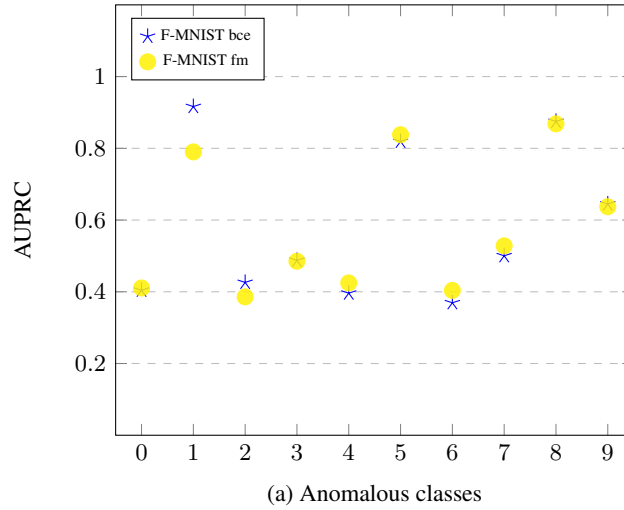


Figure 9. GANomaly results on Fashion-MNIST. All tests are here performed with a bce metric and using the area under the precision-recall curve (AUPRC). Training is done with a bce loss and a fm loss.

### A.1.3. CIFAR10

In the following Figure 10 we present the results on the CIFAR10 dataset.

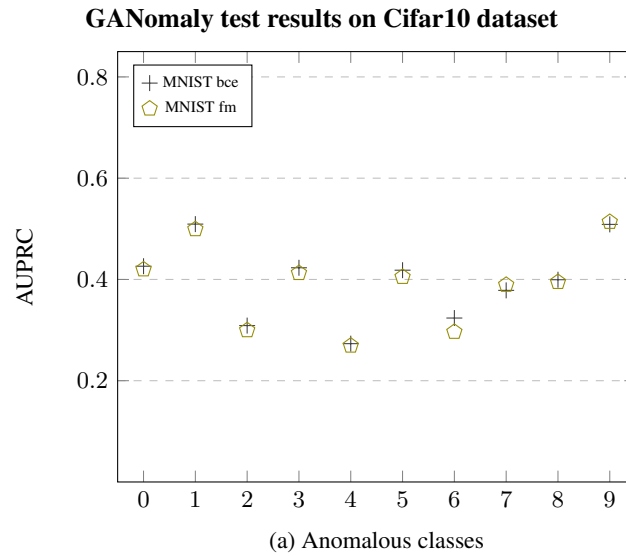


Figure 10. GANomaly results on CIFAR10. All tests are here performed through a bce metric. Training is done with a bce loss and a fm loss.

## A.2. Version 2

In Appendix A.2 we get back to the original version of the EGBAD (Zenati et al., 2018) and find that the results are much better, even if there is no activation function in the first convolutional layer. The main differences from Appendix A.1 are the following:

- **Testing phase** The testing phase is applied only once during the model selection during training.
- **Activation function** The Leaky Relu function has here been left out as for the original EGBAD paper (Zenati et al., 2018).

The training and testing pipelines employed have been already described. Please refer to section 1.3 and section 2.3 for more details. Every following section will introduce the results for a specific dataset.

### A.2.1. MNIST

Here we present the results for the MNIST dataset. In Figure 11 we show the results of BiGAN/EGBAD. In Figure 12 we show the results of GANomaly architecture.

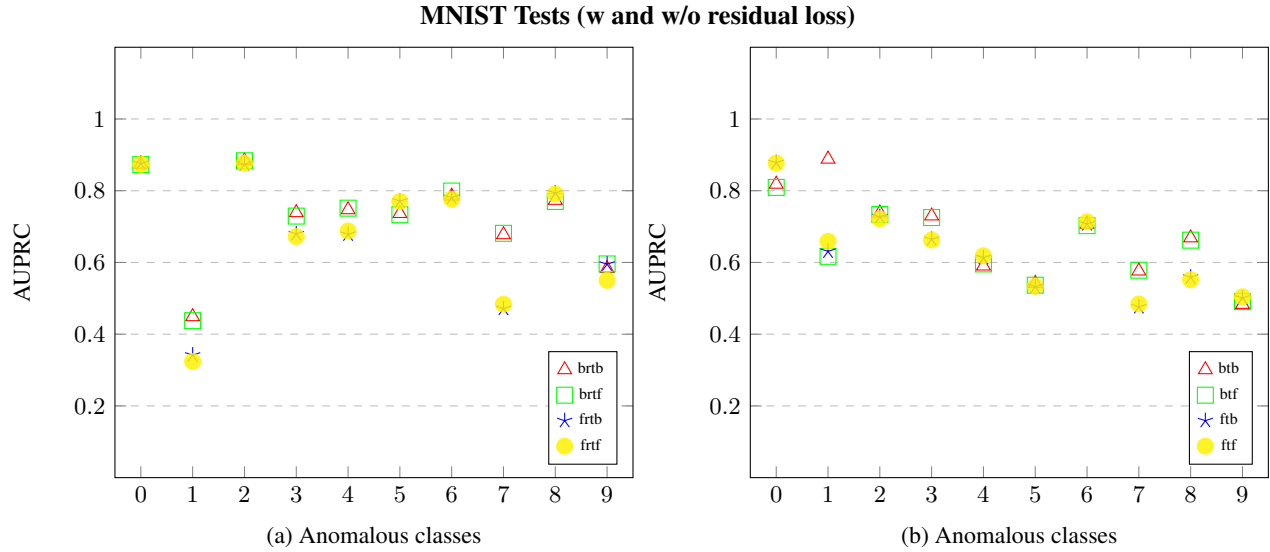


Figure 11. Performance of version 2 of BiGAN/EGBAD on MNIST measured by the area under the precision-recall curve (AUPRC). Best viewed in color. Image (a) depicts the results using the residual loss, image (b) it is without the residual loss. "b(r)tb": bce (+ residual loss) trained and bce tested, "b(r)tf": bce (+ residual loss) trained and fm tested, "f(r)tb": fm (+ residual loss) trained and bce tested, "f(r)tf": fm (+ residual loss) trained and fm tested



## GANomaly test results on MNIST dataset

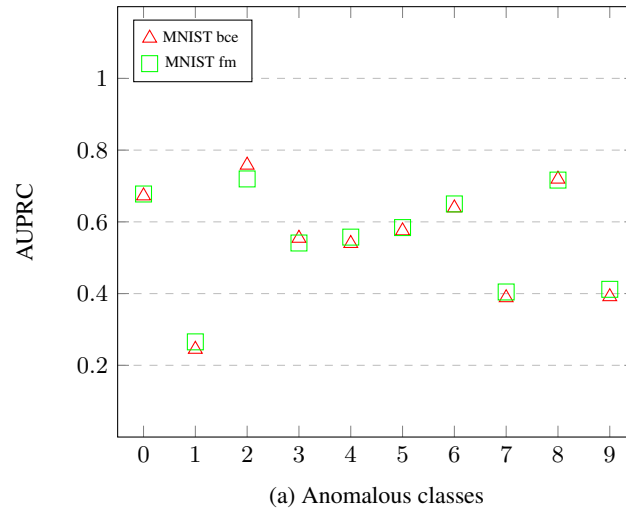


Figure 12. GANomaly results on MNIST. All tests are here performed through a bce metric and using the area under the precision-recall curve (AUPRC). Training is done with a bce loss and a fm loss.

### A.2.2. FASHION-MNIST

In the following, we present the plots regarding the performances on Fashion-MNIST dataset. Figure 13 shows the performance of BiGAN/EGBAD model and Figure 14 the performance of GANomaly model.

**Fashion-MNIST Tests (w and w/o residual loss)**

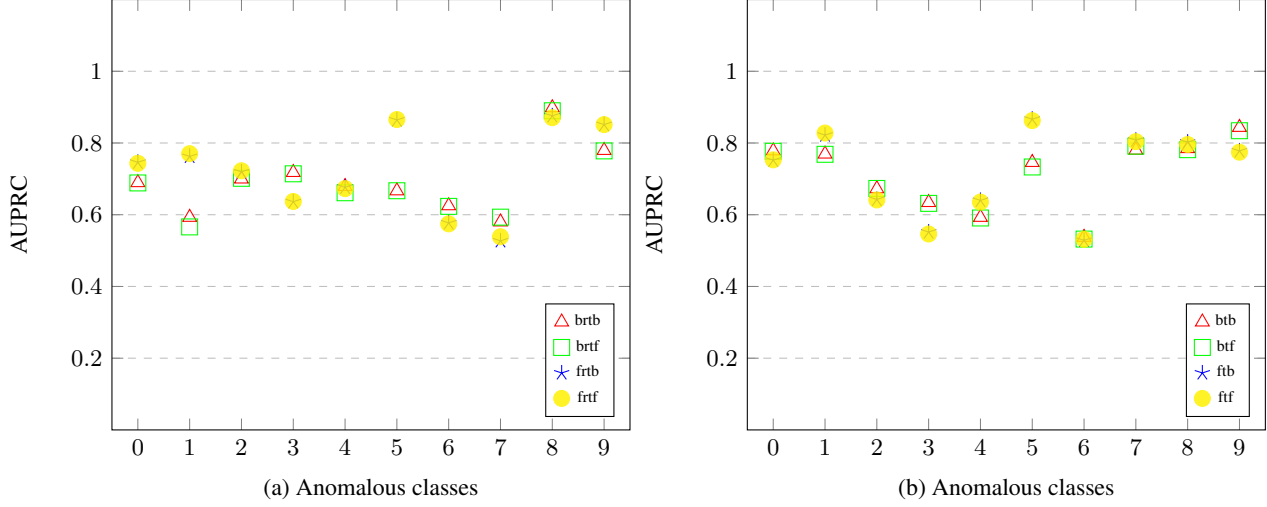


Figure 13. Performance of version 2 of BiGAN/EGBAD on Fashion-MNIST measured by the area under the precision-recall curve. Best viewed in color. Image (a) depicts the results using the residual loss, image (b) it is without the residual loss. "b(r)tb": bce (+ residual loss) trained and bce tested, "b(r)tf": bce (+ residual loss) trained and fm tested, "f(r)tb": fm (+ residual loss) trained and bce tested, "f(r)tf": fm (+ residual loss) trained and fm tested

**GANomaly test results on Fashion-MNIST dataset**

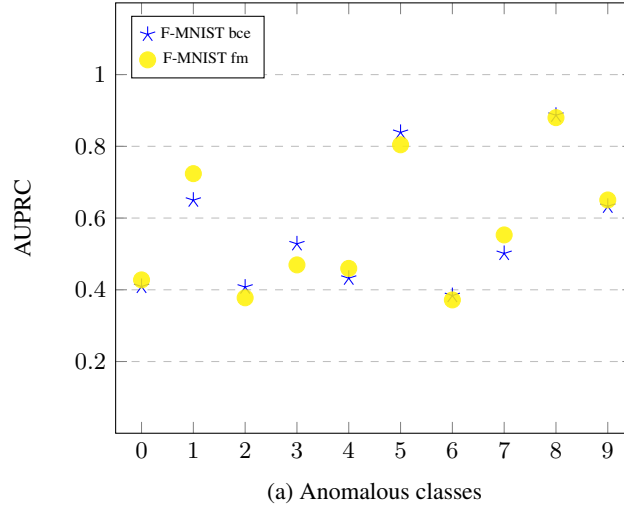


Figure 14. GANomaly results on Fashion-MNIST. All tests are here performed through a bce metric and using the area under the precision-recall curve (AUPRC). Training is done with a bce loss and a fm loss.

## A.2.3. CIFAR10

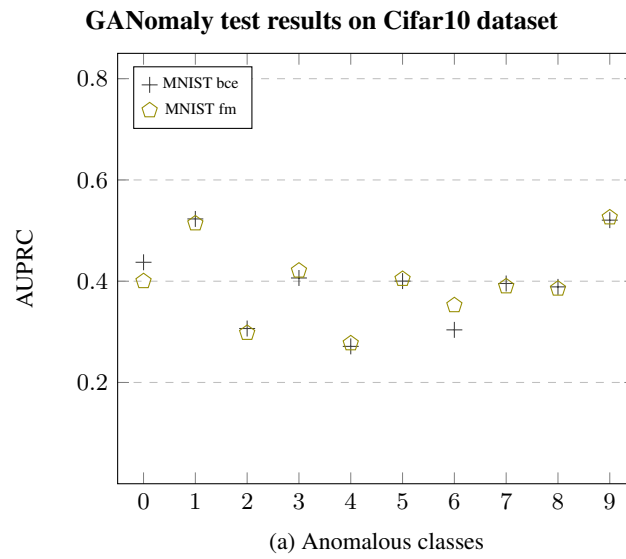


Figure 15. GANomaly results on CIFAR10. All tests are here performed through a bce metric and using the area under the precision-recall curve (AUPRC). Training is done with a bce loss and a fm loss.