BACKPROPAGATION AIDED LOGO GENERATION USING GENERATIVE ADVERSARIAL NETWORKS

Mihai Dogariu[1], Hervé Le Borgne[2], Bogdan Ionescu[3]

Logo detection algorithms rely on comprehensive datasets in order to achieve a high precision. The scarcity of these resources represents the motivation to augment them through artificial logo generation. In this paper, we propose a logo dataset augmentation technique that leverages the generalization power of generative adversarial networks (GANs). We train a GAN on a highly complex dataset formed of single logo instances such that we are able to generate random logos. Then, by applying deep gradient backpropagation, we manage to reconstruct very specific logos with this model. We validate our approach by replacing original logos from in-the-wild images with their reconstructed versions and running logo detection algorithms on the newly created images.

Keywords: Logo generation, Generative Adversarial Networks, Gradient backpropagation

1. Introduction

In recent years, the multimedia field has seen a surge in the amount of media content creation and distribution across all channels. Consumer behaviour has migrated more and more towards visual content, due to increasingly accessible media platforms, backed by larger smart devices outreach. Industries of all sorts have noticed this and have started to address the public via these channels. Consequently, product advertising has gained a fair share of attention. Thus, companies are interested in promoting their products and that is usually done either by directly inserting their products in the media content (if that is a viable approach), e.g., specific car brands in movies, or by inserting the company's logo in the content, if they are focused on less physically tangible products, such as software companies. In return, it became of high interest to be able to analyze automatically a stream of images or videos and detect all logos in that content, in particular on social media. Such a content analysis can feed marketing reports to estimate the visibility of a brand within a particular sector. This strengthens the necessity for marketing research teams to

[1]University "Politehnica" of Bucharest, Romania, e-mail: mihai.dogariu@upb.ro

[2]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France, e-mail: herve.le-borgne@cea.fr

[3]University "Politehnica" of Bucharest, Romania, e-mail: bogdan.ionescu@upb.ro

use logo detection software at a large scale, that is to say for a large number of logos, that are present in a large variety of visual context.

Logo detection is, in fact, an application of object detection, which is a field that reached its maturity. Relevant object detection papers are usually classified in single-stage detectors and two-stage detectors. The first category is composed of popular network architectures such as YOLO [1] (with its variants), SSD [2], and, more recently, RetinaNet [3]. These are composed of a single end-to-end deep network which performs both object detection and recognition. The second category is represented by architectures such as the R-CNN family (R-CNN [4], Fast R-CNN [5], Faster R-CNN [6], Mask R-CNN [7]), where there are two stages: first, a region proposal stage, and second, a recognition stage. Two-stage detectors have traditionally obtained better object detection results, but at the expense of more complex algorithms that require larger computation time, as opposed to their single-stage counterparts.

Another important aspect of logo detection is the dataset that is used to train the detector. Object detectors are usually trained on large enough and thoroughly annotated datasets, e.g., MS-COCO [8], Google Open Images [9], and this gives them enough robustness. Logo detection, however, is poorly represented, with only a few datasets available for detection, such as QMUL-OpenLogo [10], FlickrLogos-27 [11] or FlickrLogos-47 [14]. Additionally, there are WebLogo-2M [15], a weakly labelled (at image level) dataset which does not provide bounding boxes for the logos, and Large Logo Dataset (LLD) [16], a dataset consisting only of digitally represented logos and not in-the-wild instances.

As logo datasets are a scarce resource, our work aims to alleviate this problem by augmenting already existing datasets with synthetically generated logos. We propose to synthesize logos via Generative Adversarial Networks (GANs) [17]. In order to generate specific logos we propose to start from the desired logo, retrieve the latent vector that generates that logo through gradient backpropagation, and slightly alter it such that we obtain different instances of a given sample. Finally, we test our newly generated images on a logo detection task.

The rest of this paper is organized as follows. In Section 2 we discuss the process of logo generation, in Section 3 we give the mathematical insight for retrieving the latent vector that generates particular images through gradient backpropagation, in Section 4 we validate our approach on a logo detection framework, and we conclude and discuss future work in Section 5.

2. **Logo Generation**

The main idea of this paper revolves around the process of logo generation. This is done with the help of a GAN architecture, which has proved to be a successful way of generating images with various other content, such as human faces, landscapes, animals, paintings, etc. For this precise task we

adopted the DCGAN [18] network, as it is one of the most popular architectures for sample generation.

It is also known that GANs require datasets with particular properties for training, in the sense that the training samples must have similar properties (content, lightning, textures, color, structure) if sharp images are desired. If the training dataset is too diverse, then it is likely to obtain blurry or absurd images. Since there are not many logo datasets freely available, we decided to use the LLD, which contains pristine brand logos. The advantage of this dataset is that each logo is clearly represented, undistorted, exactly how it was digitally designed to begin with, on a white background (if any background is visible). This ensures that all images have a frontal pose and are of good quality, as opposed to in-the-wild logos, which usually contain significant noise (occlusions, blur, distortion, skewness, etc.). The disadvantage, however, is that each brand logo appears only once in the dataset, so there is no variety with respect to individual brands.

The LLD dataset consists of roughly 123k logos crawled from the Internet. These logos come in different sizes, therefore the first step was to resize each logo such that their larger side would become 108 pixels. Since DCGAN's discriminator accepts a fixed size input, we padded LLD's logos with white background where it was required, so as to obtain 108×108 pixel logos. This dataset will henceforth be referred to as "LLD" in the remainder of the paper.

Furthermore, we extracted each annotated logo from the Flickr_47 dataset and applied a processing similar to the one on LLD for each sample, i.e., reshaped and padded logos to obtain the 108×108 dimension with the help of the precise mask annotations that the dataset comes with, as seen in Figure 1. This produces around 2k logos, with an average of 42 instances for each of the 47 classes. However, this is not enough for GANs to produce high quality images. Therefore, we extend this dataset by copying its elements until it reaches the same size as LLD and then we mix them together to obtain a dataset that we will name "LLD_Flickr_47", of approximately 246k logos. Additionally, we scraped from the Internet the representative logo for each class in original format (not in-the-wild). We also copied these 47 logos to reach a 123k dataset size named "synth" and added them to the LLD dataset to form a new dataset "LLD_synth".

We trained several DCGAN models on LLD under different parameter setups. Namely, we used different batch sizes (64 or 128), learning rates ($1e-03$, $2e-04$, $1e-04$, $1e-05$), number of Generator passes for each iteration (n_G), number of Discriminator passes for each iteration (n_D), with or without batch normalization. The motivation behind using n_G or n_D greater than 1 is that repeated Generator/Discriminator passes on the same batch should offer higher quality results due to batch normalization layers' better stabilization around the batch mean.

FIG. 1. Logo extraction process. From left to right: original images from Flickr_47 with bounding boxes over logos, resized logos over white background after the annotation mask has been applied, original brand logos composing the "synth" dataset.

A GAN trained with LLD_Flickr_47 and LLD_synth produced low quality images, especially due to the high bias that was introduced when copying the sparse datasets to reach the same number of samples as LLD. Moreover, this bias was clearly reflected in the output images, since they were very often represented as combinations of several images from the training datasets, as depicted in Figure 2.

Since assessing the quality of the generated images is known to be difficult [13], we manually examined the outputs of all our models and decided that the best images were obtained with the model trained on LLD, with batch size of 128, $n_G = 1$, $n_D = 1$, and learning rate of $1e - 04$ for 60 epochs. Some sample logos obtained with this model can be observed in Figure 3. It can be seen that they present a highly complex structure, which includes large variations in shapes, colors, lightning, background, color gradients and text. All further experiments were carried out using this model for logo generation.

3. Backpropagation

GANs transform latent vectors z into images I through their generators, $I = G(z)$. Since we want to generate several logos similar to an already existing ones, we need to know the latent vector that drives the generator towards that

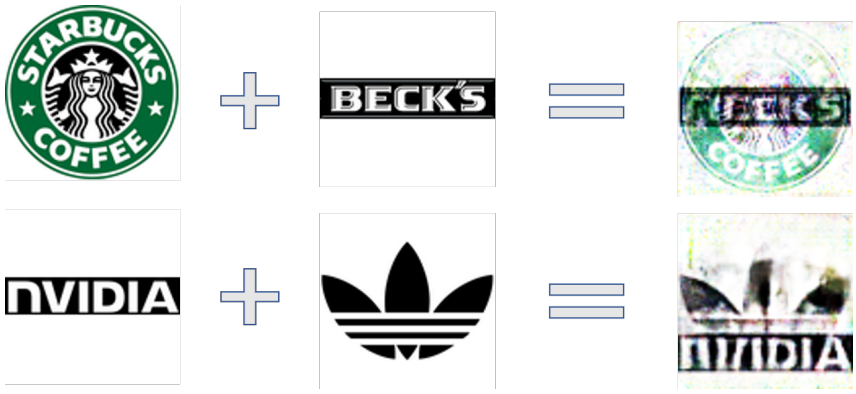

FIG. 2. Output logos (right-side of equal sign) generated as algebraic combinations between training images (left-side of equal sign).

FIG. 3. Logos generated using DCGAN model trained on LLD.

specific image. Therefore, we are interested in the backwards process, namely finding the noise vector that can be used to generate a given logo. This can be formulated as retrieving $z = G^{-1}(I)$.

However, it is difficult to believe that a perfect match can be obtained [13, 12]. Therefore, we seek for an approximation of the latent vector $z' \approx z$. Afterwards, we can add a noise vector ϵ to z' and generate an arbitrary number of new logos, similar to the one that we desire $\mathbf{I}' = \{G(z' + \epsilon), \forall \epsilon \in [-a, a] \mid z' \approx z\}$. Choosing a is just a matter of trial and error, and we observed that $a = 1e - 04$ produced good enough results, depicted in Figure 4.

FIG. 4. Logo variations starting from the leftmost image (with black bounding box) obtained by altering the latent vector with ϵ.

The reverse mapping, from images to vectors, can be done by backpropagating the gradients of the cost functions. We built over the idea of Lipton and Tripathi [19] to retrieve latent code representations corresponding to the desired images. The problem formulation, according to them, goes as follows.

Given a noise vector of size 100, uniformly distributed between -1 and 1, $z \sim U([-1,1]^{100})$, the generator of a GAN will produce an image $G(z)$. We generate another random noise vector z' and its corresponding image $G(z')$. Then, we want to force z' to be as close as possible to z in order to obtain $G(z')$ as close as possible to $G(z)$. This is done by minimizing the L_2 norm:

$$\min_{z'} \|G(z) - G(z')\|_2^2 \tag{1}$$

The optimization over z' is done by gradient descent:

$$z' \leftarrow z' - \eta \nabla_{z'} \|G(z) - G(z')\|_2^2 \tag{2}$$

This optimization is non-convex and it is unknown whether the solution that achieves the global minimum of 0 is unique. Since there is also an additional constraint that the noise vector should fall inside the $[-1,1]^{100}$ hyper-cube, a modified optimization is proposed, where each value that falls outside this interval will be clipped at the interval's closest margin [19]. This method is called *standard clipping*:

$$z' \leftarrow clip(z' - \eta \nabla_{z'} \|G(z) - G(z')\|_2^2) \tag{3}$$

The same authors also propose another variant, where instead of replacing the outliers with -1 or 1, they replace them with a random value sampled uniformly from the $[-1,1]$ interval, which they called *stochastic clipping*.

Additionally, we propose another type of clipping, which we call *tanh clipping*, where each value that falls outside the $[-1,1]$ interval is replaced with its hyperbolic tangent. Having $tanh : \mathbb{R} \to [-1,1]$ ensures that all values will reside in the required interval and will also eliminate the randomness introduced by stochastic clipping.

We implemented these 3 types of clipping and ran the backpropagation algorithm for 200k steps on the same batch of logos. Then, we ran the resulted noise vectors z' through the generator in order to see how near/far from the starting logos each clipping method is. Preliminary results shown in Figure 5 proved that the stochastic clipping is the most effective technique. Consequently, we chose this for our further experiments.

Since the previously presented attempts at recreating the original logos did not achieve satisfactory results, we investigated several other hyperparameter setups, such as different learning rates, number of backpropagation optimization iterations and momentum for stochastic gradient descent. We computed the root mean squared error (RMSE) between the original and the reconstructed logos over an entire batch of logos and illustrate our results in Table 1.

There is no significant difference between the best performing model and the second entry from top to bottom from Table 1, meaning that the extra 100k iterations do not bring a noticeable improvement. We used the boldfaced setup in our next experiments and did not run extensive experiments for setups

FIG. 5. Backpropagated logos obtained with different clipping techniques. Top row: original logos; next 3 rows, in order: standard, stochastic and hyperbolic tangent clipping.

Table 1

RMSE results for different backpropagation setups

learning rate	momentum	number of iterations	RMSE
1e-07	0.75	120k	4090
1e-07	0.9	120k	3640
1e-07	0.9	220k	3976
1e-06	0.75	120k	3860
1e-06	0.9	120k	3750
1e-06	**0.9**	**220k**	**3600**
1e-05	0.9	220k	4519
1e-04	0.9	220k	9981

that did not show promising results. In Figure 6 we present some of the most conclusive examples of how the backpropagation mechanism worked for several logos.

FIG. 6. Logos obtained with the help of the backpropagation mechanism. Top: original logos; bottom: logos obtained after backpropagating original images through the DCGAN.

FIG. 7. Results obtained with AVEA. First row: original logos from LLD; second row: images obtained with the decoder of the AVAE framework; third row: images obtained with the generator of the AVAE framework.

It can be seen that the retrieved logos are generally a blurred version of their original counterparts. This is highly problematic for logos with a complicated texture or logos that contain text. We also examined an alternative way of reconstructing logos with the help of an Adversarial Variational Autoencoder (AVAE) [12]. We present in Figure 7 the results that were obtained with AVAE on LLD. The quality of these images is similar to that we obtained in Figure 3 for the generator, but the decoder results in sharper images. Since our approach could be applied as an add-on to any trained model, we decided to use it further in our experiments.

4. Logo Detection

As it was mentioned before, assessing the quality of the samples generated by a GAN is a complicated subject itself. Subjectively, it is fairly simple to tell the brand that a reconstructed logo represents. Objectively, however, we discovered that the RMSE will cap at some point due to the nature of the optimization function. In order to determine how good the reconstruction is, we setup a logo detection pipeline. Logo detection is, in essence, an object detection problem. Therefore, we applied a consecrated logo detection algorithm, the Faster R-CNN [6] architecture.

We created a dataset composed of logos generated with the algorithm explained in this paper, starting from the Flickr_47 training dataset. The steps are as follows. We extracted each individual logo according to its annotation mask and resized it to 108×108 pixels, as explained in Section 2. We proceeded with reconstructing each individual logo with the backpropagation algorithm explained in Section 3 and obtained their corresponding versions as outputs of the DCGAN's generator. Afterwards, we resized the reconstructed logos back to their original size and applied the annotation masks again, in order to remove the unnecessary background. Finally, we replaced the original logos from the Flickr_47 dataset with their reconstructions. The pipeline for this entire processing is depicted in Figure 8. This dataset was added to the already existing training dataset, thus obtaining two versions for each image in the

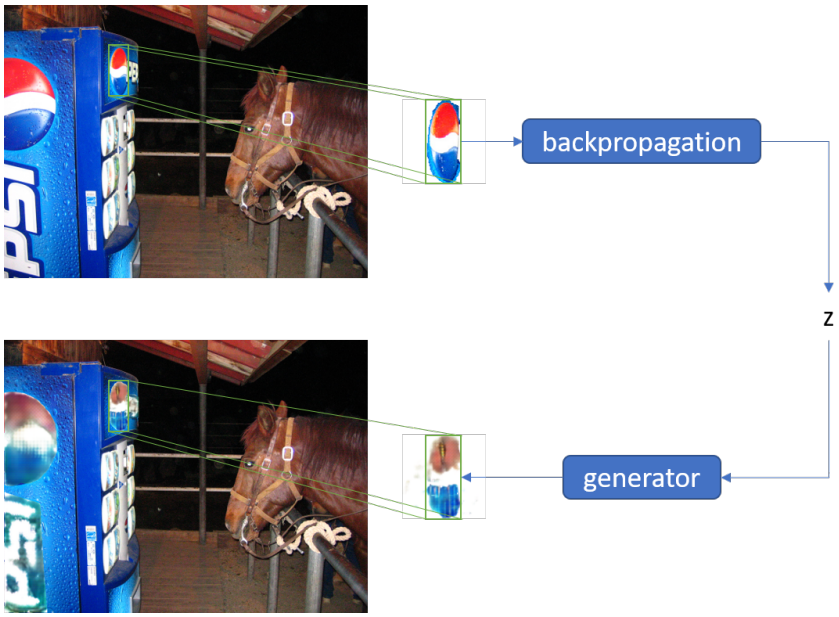

FIG. 8. Pipeline of the backpropagation and generation system, with z the retrieved noise vector.

training dataset: one with the original logos and one with the reconstructed logos.

We ran the logo detection algorithm as follows. First, we pre-trained the model on the MS-COCO dataset, then we fine-tuned it on Flickr_47 train dataset and tested it on the Flickr_47 test dataset. We used the mAP@0.5 IoU metric. This means that a logo is correctly detected if its class label matches the groundtruth and if its detected bounding box overlaps the groundtruth bounding box such that the ratio between the surface of their intersection and the surface of their union exceeds 0.5. We obtained a mAP@0.5 of 0.6019 which is very promising. We also trained another vanilla logo detection model on Flickr_47 training dataset without any reconstructed images and tested it on the Flickr_47 test dataset. We present in Figure 9 two examples of how the training dataset augmented with our algorithm performs better on one example and worse on another example than its vanilla counterpart.

In Figure 9 the top row presents the logo detection results obtained when processing a 'Esso' billboard. The model trained on the vanilla dataset correctly detects the 'Esso' logo. The model trained on the augmented dataset also detects the 'Esso' logo but it also detects two false positives ('pepsi_text' and 'nvidia_text'). On the bottom row, the vanilla dataset leads to a false positive detection ('guiness_text' above the bottle's label) and one wrong prediction ('tsingtao_text' instead of 'chimay_text' on the glass text), while the augmented dataset gives a correct prediction. After examining a significant part of the results, we noticed that errors concentrate mostly on classes which

FIG. 9. Logo detection results obtained by training on Flickr_47 training dataset (left column) and on the augmented dataset (right column)

contain only text, such as 'Pepsi' and 'Google' in Figure 6. This was expected since these classes are very different from what is usually found in the logo dataset that was used to train the DCGAN. Moreover, the text is often tilted, occluded and in poor lightning conditions or barely visible at all.

There are numerous factors that should be taken into consideration when examining the results. First of all, the size of the logos in the original images plays an important role. The DCGAN is limited to generating fixed size logos, which we set to 108×108 pixels, as mentioned in Section 2. This means that logos which are originally larger than this will be downscaled and their reconstructed counterparts will be upscaled through interpolation. As there are no lossless interpolation algorithms, the image's quality will decrease when its size is enlarged. Next, the DCGAN limits the extent to which the backpropagation algorithm can be applied, because it contains several layers (batch normalization, dropout), due to which it is difficult to perfectly recover the latent vector through gradient backpropagation. Moreover, we notice a significant difference between the dataset that was used to train the DCGAN, i.e.,

LLD, and the dataset that we used to reconstruct logos, Flickr_47. We reckon that this factor bears the most responsibility for the inability to correctly reconstruct complex images. By complex we mean images that are affected by poor lightning, occlusion and whose content is highly complex as compared to the training dataset.

5. Conclusions

In this paper we presented a logo generation algorithm based on GANs and gradient backpropagation. We trained a DCGAN model to generate random 108×108 pixels logos starting from the LLD dataset. Then, we extracted in-the-wild logos from the Flickr_47 dataset. These logos were backpropagated through the DCGAN, we extracted their latent code representations and then used these noise vectors to guide the DCGAN's generator to output similar looking logos. These new logos were used to replace their original counterparts and we ran a logo detection pipeline to validate our processing.

This pipeline yielded a mAP@0.5 of 0.6019 which is very promising, especially given that this framework can be applied to any other deep architecture. Its strength comes from the fact that it is not restricted to a particular model. This means that we can use frozen models without having to change their underlying architectures. As a consequence, this provides an alternative dataset augmentation method to already existing classical approaches.

For this algorithm to properly work, several conditions need to be satisfied at once: the training dataset for the GAN's generator should be similar to the one that we wish to reconstruct, the datasets should be diverse enough, so that not only algebraic combinations of training images are encountered (see Figure 2), and the images should be of similar size, with only small variations between them. In conclusion, provided that these conditions are fulfilled, the proposed method represents an appealing solution for dataset augmentation.

Acknowledgment

This work has been funded by the Operational Programme Human Capital of the Ministry of Europe Funds through the Financial Agreement 51675/ 09.07.2019, SMIS code 125125 and by the Ministry of Innovation and Research, UEFISCDI, project SMARTRetail, agreement PN-III-P2-2.1-PTE-2019-0055, 2020-2022.

REFERENCES

[1] *J. Redmon, S. Divvala, R. Girshick and A. Farhadi*, You only look once: Unified, real-time object detection, in Proc. IEEE Conference on Computer Vision and Pattern Recognition 2016 (pp. 779–788).

[2] *W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu and AC. Berg*, Ssd: Single shot multibox detector, in Proc. European Conference on Computer Vision 2016 (pp. 21–37).

[3] *T.Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár*, Focal loss for dense object detection, in Proc. IEEE International Conference on Computer Vision 2017 (pp. 2980–2988).

[4] *R. Girshick, J. Donahue, T. Darrell, J. Malik*, Region-based convolutional networks for accurate object detection and segmentation, IEEE Trans. Pattern Analysis and Machine Intelligence, **38**(2015), No. 1, 142–58.

[5] *R. Girshick* Fast R-CNN, in Proc. IEEE International Conference on Computer Vision 2015 (pp. 1440–1448).

[6] *S. Ren, K. He, R. Girshick and J. Sun*, Faster R-CNN: towards real-time object detection with region proposal networks, IEEE Trans. Pattern Analysis and Machine Intelligence, **39**(2016), No. 6, 1137–49.

[7] *K. He, G. Gkioxari, P. Dollár and R. Girshick*, Mask R-CNN, in Proc. IEEE International Conference on Computer Vision 2017 (pp. 2961–2969).

[8] *T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C.L. Zitnick*, Microsoft coco: common objects in context, in Proc. European Conference on Computer Vision 2014 (pp. 740–755).

[9] *A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov and T. Duerig*, The open images dataset v4: unified image classification, object detection, and visual relationship detection at scale. arXiv preprint arXiv:1811.00982, 2018.

[10] *H. Su, X. Zhu and S. Gong*, Open logo detection challenge, in Proc. British Machine Vision Conference (BMVC), Newcastle, UK, 2018.

[11] *Y. Kalantidis, L.G. Pueyo, M. Trevisiol, R. van Zwol and Y. Avrithis*, Scalable triangulation-based logo recognition, in Proc. 1st ACM International Conference on Multimedia Retrieval 2011 (pp. 1–7).

[12] A. Plumerault, H. Le Borgne, and C. Hudelot, Avae: Adversarial variational auto encoder, In International Conference on Pattern Recognition (ICPR), 2020.

[13] A. Plumerault, H. Le Borgne, and C. Hudelot, Controlling generative models with continuous factors of variations. In International Conference on Learning Representations (ICLR), 2020.

[14] *S. Romberg, L.G. Pueyo, R. Lienhart and R. Van Zwol*, Scalable logo recognition in real-world images, in Proc. 1st ACM International Conference on Multimedia Retrieval 2011 (pp. 1–8).

[15] *H. Su, S. Gong and X. Zhu*, Weblogo-2m: scalable logo detection by deep learning from the web, in Proc. IEEE International Conference Computer Vision Workshops 2017 (pp. 270–279).

[16] *A. Sage, E. Agustsson, R. Timofte and L. Van Gool*, Logo synthesis and manipulation with clustered generative adversarial networks, in Proc. IEEE Conference on Computer Vision and Pattern Recognition 2018 (pp. 5879–5888).

[17] *I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio*, Generative adversarial nets, in Proc. Advances in Neural Information Processing Systems 2014 (pp. 2672–2680).

[18] *A. Radford, L. Metz and S. Chintala*, Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[19] *Z.C. Lipton and S. Tripathi*, Precise recovery of latent vectors from generative adversarial networks. arXiv preprint arXiv:1702.04782, 2017.