Deep Image Fingerprint: Towards Low Budget Synthetic Image Detection and Model Lineage Analysis — Supplemental Material

Sergey Sinitsa, Ohad Fried
Reichman University

This document is structured as follows: In Appendix A we provide expanded implementation details. In Appendix B we explain the GAN datasets and the data preparation process using Stable Diffusion model variants. In Appendix C we present additional experimental results.

A. Implementation Details

This section includes the implementation details of our work. The DIF models were trained and tested on an RTX 3060 GPU with 12 GB of VRAM. We used PyTorch [12] as our deep learning framework.

A.1. Selection of Denoising Filter

Here we explain why DnCNN [17] was chosen as the denoising filter (f_D) for all the fingerprint methods. The objective of the f_D is to extract only noise and artifacts while filtering out the semantic information of the image. The pattern of $\mathcal{F}$ is primarily present in the high-mid frequencies [9], making a high-pass filter a straightforward choice as a reference f_D . However, some semantic content, such as edges, also exists in these frequencies. Therefore, a more sophisticated denoising filter is required. DnCNN is such a filter. It is trained to extract Gaussian noise while preserving edges, and it has demonstrated good performance with other types of noise as well [17]. To validate our assumption, we compare the performance of Marra18, Joslin20, and DIF using both a Gaussian high-pass filter and DnCNN (Tab. A).

Method	Gaussin High-Pass	DnCNN
Marra18	52.4	**62.6**
Joslin20	51.0	**51.7**
DIF	73.3	**92.3**

Table A. Mean accuracy (%) of detection for LTIMs. 1024 images were used in train set for each.

As expected, all of the methods perform worse with the Gaussian high-pass filter. The high-frequency content of the image influences the averaged and extracted fingerprint pattern, leading to low correlation values between unseen residuals (test set) and fingerprints.

A.2. Usage of DnCNN

The DnCNN-S [17] model was trained separately from the residual extraction procedure. The training was performed according to the original work with minor changes. We trained the DnCNN-S model for 2,000 epochs with a learning rate of 10^{-4} and the Adam optimizer [8]. Only real images were used during training. Random crop is set to size (48×48) pixels and a sigma range is set to [5, 15]. The number of training images is 1024.

During the inference of the DnCNN, we applied post-processing. First, the input to the DnCNN was padded with 10 pixels on each dimension and then reduced as post-processing according to the recommendations of the authors [17]. Additionally, we observed a bias within the residuals, so we performed an additional post-processing step. We took the training set of the DnCNN and calculated the average of its residuals, thus estimating the fingerprint of the DnCNN ($\mathcal{F}_{DnCNN}$). During the inference of the model, this fingerprint was subtracted from the output according to:

$$R_i = f_D(X_i) - \mathcal{F}_{DnCNN} \tag{1}$$

A.3. U-Net Architecture

In Tab. B, we summarize the architecture of the U-Net model, denoted in the paper as g_θ. Each row represents a convolution block, comprised of two convolutional layers and either an up-sampling or a down-sampling layer. The convolutional layers have a kernel size of 3, stride of 1, and padding of 1 pixels for each spatial dimension to achieve boundary artifacts. Each convolutional layer is accompanied by Batch-Normalization [4] and an activation function, which are specified in Tab. B. We use max-pooling with a kernel size of 2×2 pixels for down-sampling and a deconvolution layer with a kernel size of 2×2 pixels and stride of 2 pixels for up-sampling. The latter is suspected to be the

Model	C_{in}	C_{out}	f_{act}
Encoder	16	32	Leaky-ReLU
	32	64	Leaky-ReLU
	64	128	Leaky-ReLU
	128	256	Leaky-ReLU
Decoder	256+256	128	Leaky-ReLU
	128+128	64	Leaky-ReLU
	64+64	32	Leaky-ReLU
	32+32	32	Leaky-ReLU
	32	3	TanH

Table B. U-Net model architecture. Each row represents a convolutional block, with C_{in} and C_{out} indicating the input and output channels, respectively. f_{act} is the activation function used in the block. The symbol "+" in C_{in} indicates that the layer input includes skip connections.

main causes of grid-like artifacts [10]. The last block consists only of a single convolution layer where we do not use Batch-Normalization and rely on hyperbolic tangent (TanH) as the activation function.

A.4. Selection of Model Architecture

We selected the U-Net architecture through hyperparameter tuning and by incorporating concepts mentioned in Section 3.1. A Convolutional Network (C-Net) consists of convolutional blocks without down-sampling and up-sampling operations. An Up-Sampling Network (D-Net) serves as the decoder in the U-Net model. U-Net with 1x1 kernels within convolutional layers denoted as U1-Net, where we test the importance of only up-sampling artifacts. U-Net refers to the aforementioned architecture.

In Tab. C, U-Net outperforms other architectures, while C-Net shows the worst performance due to the lack of up-sampling, which is crucial (Section 3). Interestingly, C-Net shows minimal decline when used with the GLIDE model, indicating a dominant presence of boundary artifacts (Figure 2). Additionally, U1-Net, designed to reduce boundary artifacts, experiences approximately a 5% decline compared to other architectures when used with GLIDE.

D-Net performs similarly to U-Net as both models generate both up-sampling and boundary artifacts. However, U-Net was chosen due to its superior performance.

B. Datasets

B.1. GAN Datasets

The GAN datasets that we use in this work are from the supplementary materials of Wang *et al.* [15]. Train set consists of 360k real images corresponding to 20 LSUN classes [16] and 360k images generated by 20 ProGAN [5] models. Models are trained per LSUN class. For the ProGAN dataset we randomly select 2,000 images per real and fake class, for each LSUN image class, resulting in 4,000 images per LSUN class overall. Test set of [15] contains images generated by a number of GANs, specifically by StyleGAN [6], StyleGAN2 [7], BigGAN [1], StarGAN [2], GauGAN [11] and ProGAN [5].

B.2. Fined-Tuned Stable Diffusion Models

We will outline the process of constructing datasets for the fine-tuned Stable Diffusion models discussed in **??**. In all cases, fine-tuning involves the DreamBooth method [13], specifically LTIM is trained to reproduce a target object or style from a pre-defined keyword. There are three models: SD 1.4S, SD 1.5A, and SD 2.0R. The authors customly fine-tuned SD 1.4S with 10 photos. SD 1.5A[1] is a publicly available model that was fine-tuned in two stages: first, SD 1.4 was fine-tuned with 680k anime images, and then, after replacing the image decoder with one from SD 1.5, it was additionally fine-tuned with [13] on a different set of anime images. The last model, SD 2.0R[2], is fine-tuned with an unspecified amount of robot images. Following above, keywords are known for each case, as SD 1.4S is trained by us, and for others, keyword is specified in the instructions for each model. To produce images with a new models a keyword is added to captions from the Laion-5B dataset [14] in the following format: "keyword caption". This forces fine-tuned model to generate images with new information previously unknown to the source models. Fig. E shows an example of outputs resulting from the same caption.

C. Additional Results

C.1. The Effect of Train Set Size

In Section 4 we present an evaluation of the proposed method as a detector of generated images. To evaluate the method under various amounts of training samples, we measure performance on images produced by both GAN and LTIM models, as shown in Figs. B and C. We observe that the accuracy remains stable across most of the models and starts to degrade drastically at $N_S = 128$, where we observe a 5% drop for the majority of the models.

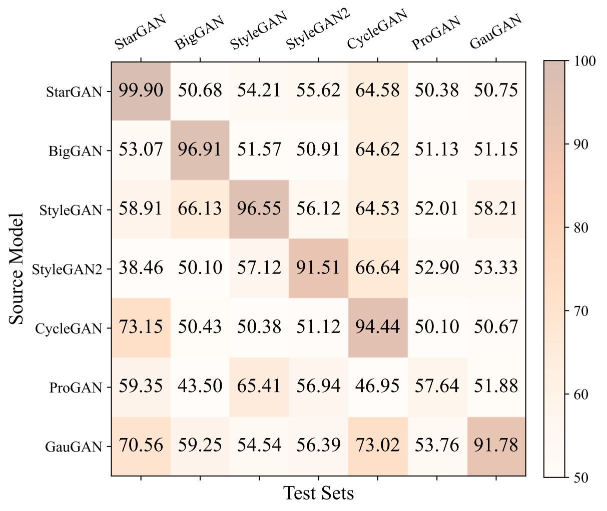
C.2. Consistency for a Low Sample Amount

We report the consistency of accuracy with a low number of samples. In this setting, each training sample has a greater impact on the results. To test this effect, we trained four models (Resnet-50, Grag21, Resnet-M, and DIF) ten times with 128 samples according to Section 4.2. Each time we randomly sampled the training set. As shown in Fig. D, despite the weaker consistency of the results for DIF, statistics support the relation observed in Table 2.

[1]https://huggingface.co/DGSpitzer/Cyberpunk-Anime-Diffusion
[2]https://huggingface.co/nousr/robo-diffusion-2-base

Architecture	SD 1.4	SD 2.1	MJ	DALL·E-Mini	GLIDE	DALL·E-2	Mean
U-Net	99.3	**89.5**	**99.0**	**99.0**	**90.3**	**79.5**	**92.8**
U1-Net	99.4	88.8	98.9	98.3	85.2	80.2	91.8
D-Net	**99.4**	89.3	98.9	99.0	89.0	78.0	91.9
C-Net	60.7	53.3	89.9	98.0	88.2	69.6	76.6

Table C. Detection accuracy (%) of CNN architectures on images from LTIMs. D-Net is a up-sampling network and C-Net convolutional network. U-Net demonstrates best results.



Source Model	StarGAN	BigGAN	StyleGAN	StyleGAN2	CycleGAN	ProGAN	GauGAN
StarGAN	99.90	50.68	54.21	55.62	64.58	50.38	50.75
BigGAN	53.07	96.91	51.57	50.91	64.62	51.13	51.15
StyleGAN	58.91	66.13	96.55	56.12	64.53	52.01	58.21
StyleGAN2	38.46	50.10	57.12	91.51	66.64	52.90	53.33
CycleGAN	73.15	50.43	50.38	51.12	94.44	50.10	50.67
ProGAN	59.35	43.50	65.41	56.94	46.95	57.64	51.88
GauGAN	70.56	59.25	54.54	56.39	73.02	53.76	91.78

Figure A. Cross-detection accuracy (%) for GAN models.

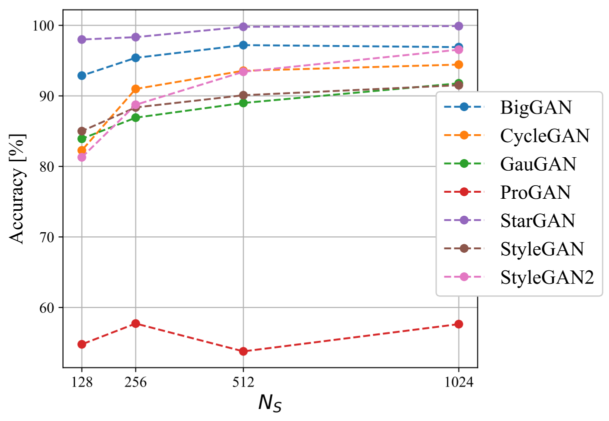


Figure B. Accuracy (%) as a function of train samples for GAN models.

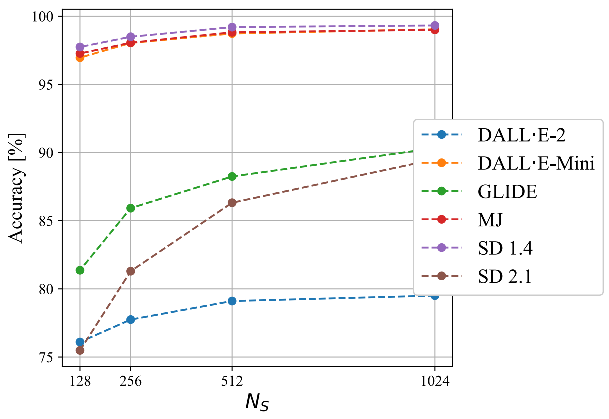


Figure C. Accuracy (%) as a function of train samples for LTIM models.

C.3. Cross-Detection for GAN Models

We report the complete map of cross-detection for datasets corresponding to GAN and ProGAN$_t$ models in Figs. A and F respectively. As shown in the figures, the cross-correlation is low for all the models, thus models were trained separately.

C.4. Cross-Detection for Custom Trained ProGANs

Models P_A, P_B, $\hat{P}_A$ and $\hat{P}_B$ were trained on centrally cropped images (128 × 128 px) from AFHQ [3]. We report the full cross-detection matrix for the four models in Fig. H. The cross-detection across all the models is similar, namely no symmetry between different models and high-symmetry within converged models. In addition we present the examples of images generated by P_A ordered according to check point epochs at Fig. G. Image quality corresponds to the convergence state of the model. Starting from epoch 40 the model produces visually appealing results, that correspond to symmetric and high cross-detection accuracy observed in Fig. H. Cross-detection for $\hat{P}_A$ is symmetric, but with lower cross-detection accuracy values. We suspect that in this specific case, the model did not converge properly.

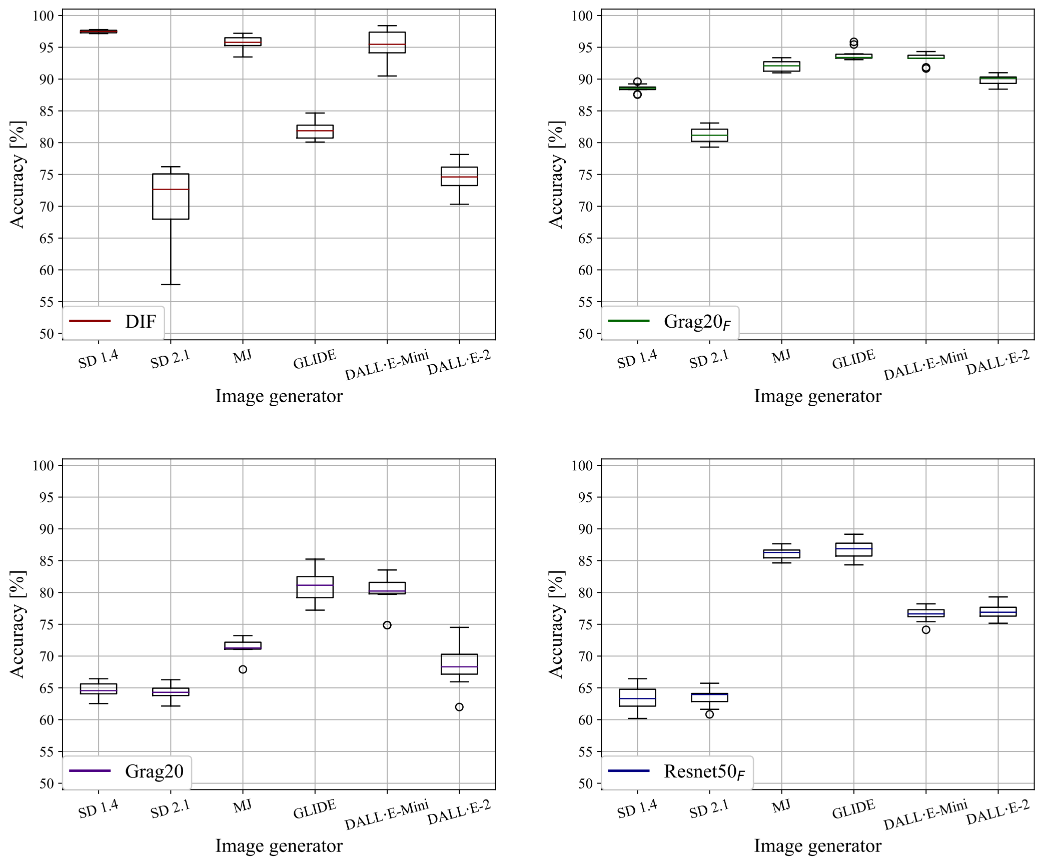


Figure D. Box plots of accuracy per image generator dataset. DIF demonstrates less consistent results, especially with SD 2.1., but preserves relation from Table 2.

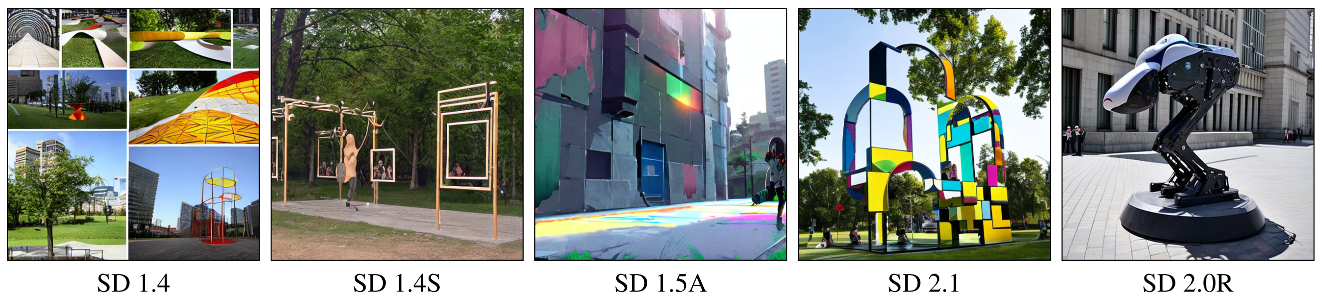
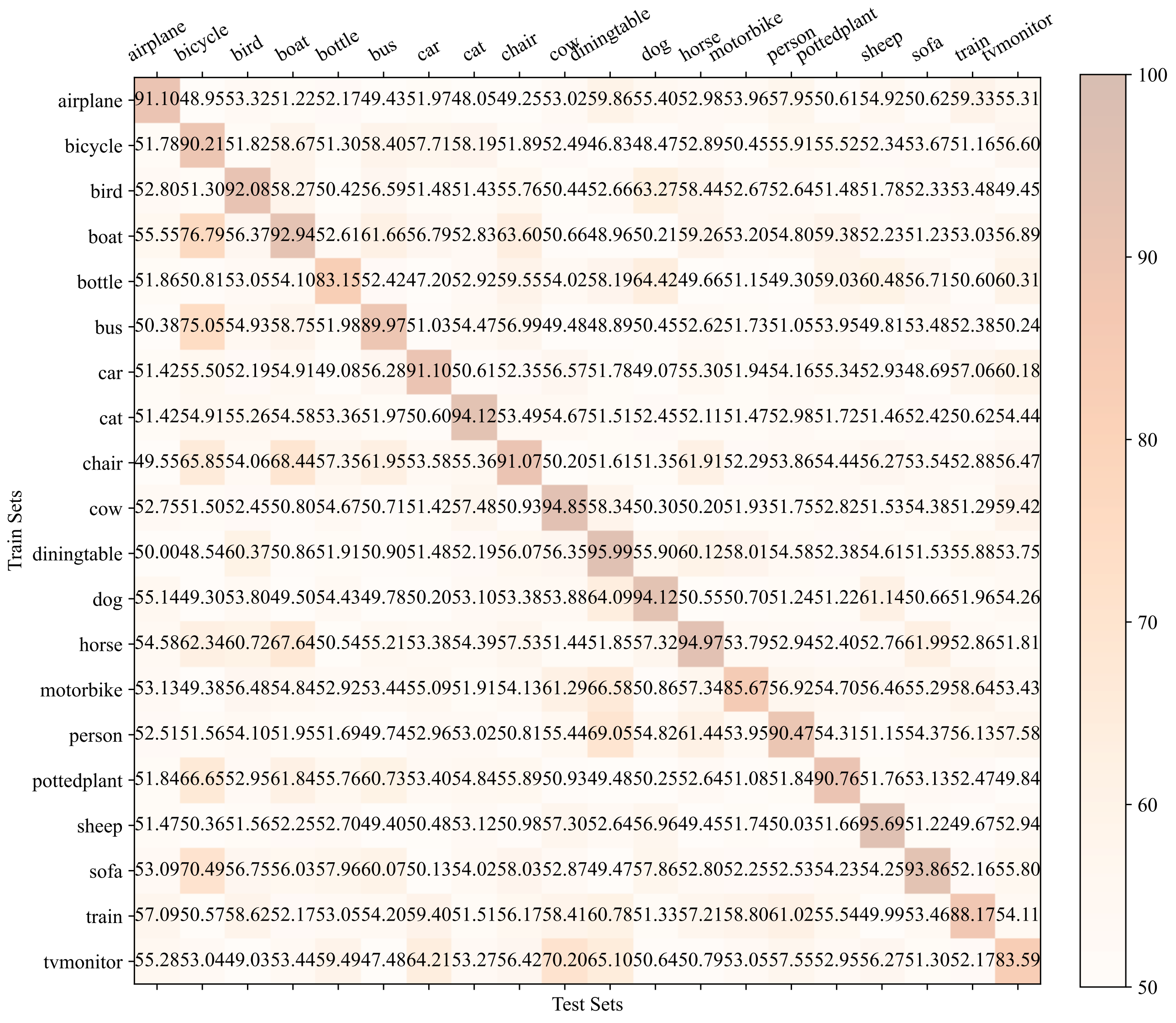


Figure E. Examples of images produced by Stable Diffusion models and their variants. Images correspond to the caption "Best Public Arts Installations park 2015" and addition of model specific keywords.

C.5. Cross-Detection vs. Cross-Correlation

In this section, we clarify why model lineage estimation is performed using image cross-detection rather than fingerprint cross-correlation. Cross-correlation (Section 3) is an intuitive process where we expect a correlation value of 1 for a fingerprint matched with itself, high absolute values for similar fingerprints, and low values for unrelated fingerprints. However, in our experiments (Section 4.3), we found that these values are not informative. The absolute cross-correlation values of the extracted fingerprints ($\mathcal{F}_E$) exhibit a random pattern (Fig. J), while the fingerprints obtained through residual averaging ($\mathcal{F}_A$) display a trend similar to cross-detection (Fig. I). However, the cross-detection of $\mathcal{F}_A$ decreases more rapidly with epochs, lacks normalization, and is always symmetric. As a result, the certainty of



	airplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	person	pottedplant	sheep	sofa	train	tvmonitor
airplane	91.10	48.95	53.32	51.22	52.17	49.43	51.97	48.05	49.25	53.02	59.86	55.40	52.98	53.96	57.95	50.61	54.92	50.62	59.33	55.31
bicycle	51.78	90.21	51.82	58.67	51.30	58.40	57.71	58.19	51.89	52.49	46.83	48.47	52.89	50.45	55.91	55.52	52.34	53.67	51.16	56.60
bird	52.80	51.30	92.08	58.27	50.42	56.59	51.48	51.43	55.76	50.44	52.66	63.27	58.44	52.67	52.64	51.48	51.78	52.33	53.48	49.45
boat	55.55	76.79	56.37	92.94	52.61	61.66	56.79	52.83	63.60	50.66	48.96	50.21	59.26	53.20	54.80	59.38	52.23	51.23	53.03	56.89
bottle	51.86	50.81	53.05	54.10	83.15	52.42	47.20	52.92	59.55	54.02	58.19	64.42	49.66	51.15	49.30	59.03	60.48	56.71	50.60	60.31
bus	50.38	75.05	54.93	58.75	51.98	89.97	51.03	54.47	56.99	49.48	48.89	50.45	52.62	51.73	51.05	53.95	49.81	53.48	52.38	50.24
car	51.42	55.50	52.19	54.91	49.08	56.28	91.10	50.61	52.35	56.57	51.78	49.07	55.30	51.94	54.16	55.34	52.93	48.69	57.06	60.18
cat	51.42	54.91	55.26	54.58	53.36	51.97	50.60	94.12	53.49	54.67	51.51	52.45	52.11	51.47	52.98	51.72	51.46	52.42	50.62	54.44
chair	49.55	65.85	54.06	68.44	57.35	61.95	53.58	55.36	91.07	50.20	51.61	51.35	61.91	52.29	53.86	54.44	56.27	53.54	52.88	56.47
cow	52.75	51.50	52.45	50.80	54.67	50.71	51.42	57.48	50.93	94.85	58.34	50.30	50.20	51.93	51.75	52.82	51.53	54.38	51.29	59.42
diningtable	50.00	48.54	60.37	50.86	51.91	50.90	51.48	52.19	56.07	56.35	95.99	55.90	60.12	58.01	54.58	52.38	54.61	51.53	55.88	53.75
dog	55.14	49.30	53.80	49.50	54.43	49.78	50.20	53.10	53.38	53.88	64.09	94.12	50.55	50.70	51.24	51.22	61.14	50.66	51.96	54.26
horse	54.58	62.34	60.72	67.64	50.54	55.21	53.38	54.39	57.53	51.44	51.85	57.32	94.97	53.79	52.94	52.40	52.76	61.99	52.86	51.81
motorbike	53.13	49.38	56.48	54.84	52.92	53.44	55.09	51.91	54.13	61.29	66.58	50.86	57.34	85.67	56.92	54.70	56.46	55.29	58.64	53.43
person	52.51	51.56	54.10	51.95	51.69	49.74	52.96	53.02	50.81	55.44	69.05	54.82	61.44	53.95	90.47	54.31	51.15	54.37	56.13	57.58
pottedplant	51.84	66.65	52.95	61.84	55.76	60.73	53.40	54.84	55.89	50.93	49.48	50.25	52.64	51.08	51.84	90.76	51.76	53.13	52.47	49.84
sheep	51.47	50.36	51.56	52.25	52.70	49.40	50.48	53.12	50.98	57.30	52.64	56.96	49.45	51.74	50.03	51.66	95.69	51.22	49.67	52.94
sofa	53.09	70.49	56.75	56.03	57.96	60.07	50.13	54.02	58.03	52.87	49.47	57.86	52.80	52.25	52.53	54.23	54.25	93.86	52.16	55.80
train	57.09	50.57	58.62	52.17	53.05	54.20	59.40	51.51	56.17	58.41	60.78	51.33	57.21	58.80	61.02	55.54	49.99	53.46	88.17	54.11
tvmonitor	55.28	53.04	49.03	53.44	59.49	47.48	64.21	53.27	56.42	70.20	65.10	50.64	50.79	53.05	57.55	52.95	56.27	51.30	52.17	83.59

Figure F. Accuracy (%) cross-detection for ProGAN models.

model relationships is diminished due to: a) increased sensitivity to changes during training/fine-tuning, b) absence of a symmetry parameter, and c) insufficient information from the correlation value alone.

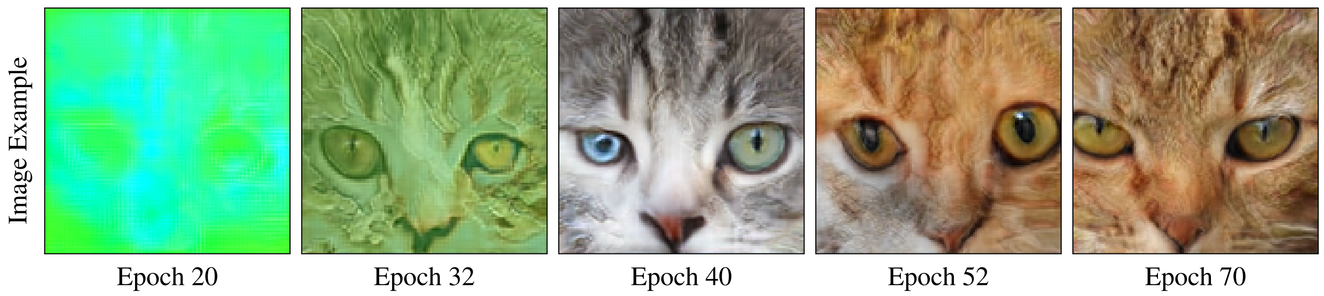
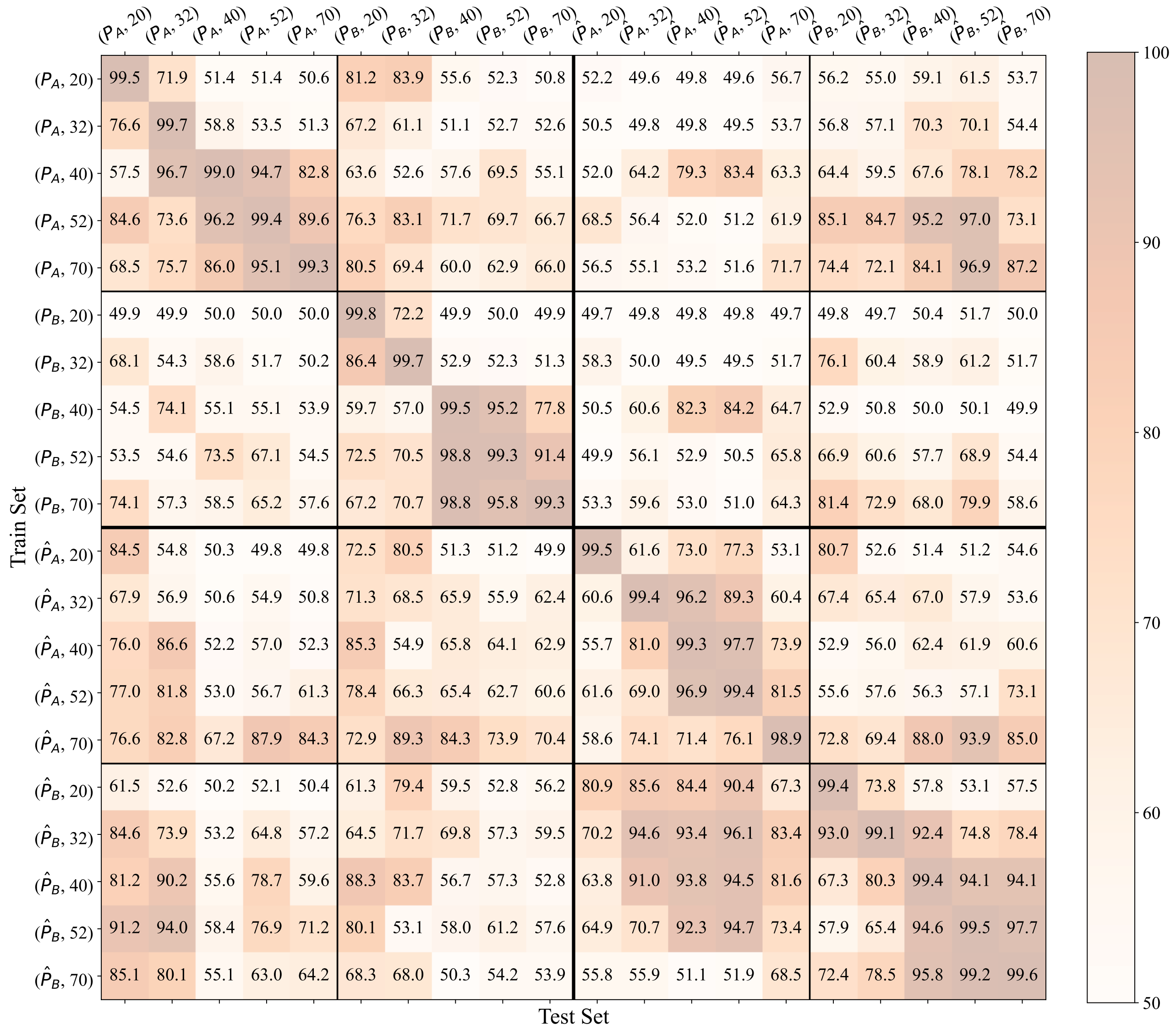
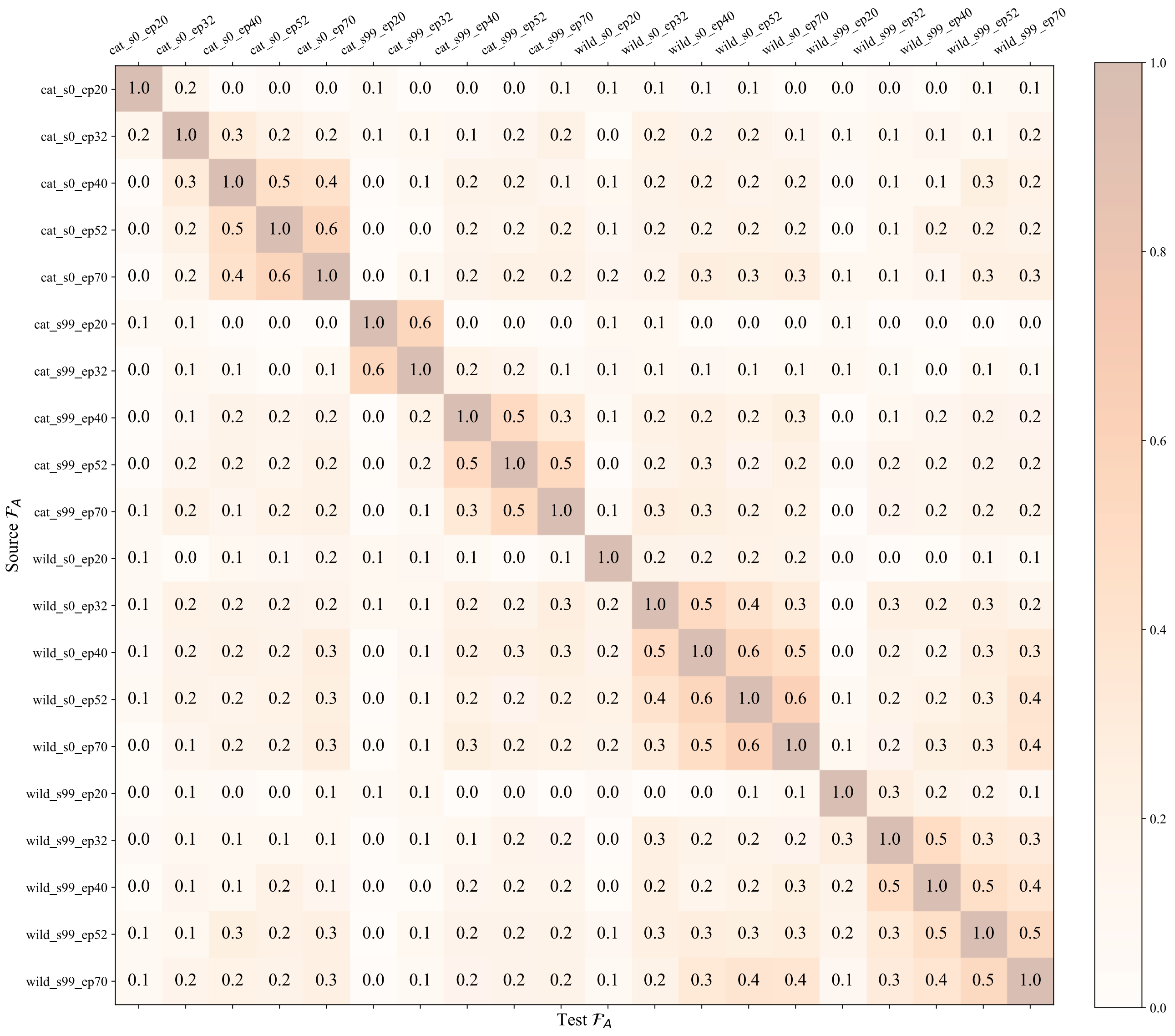


Figure G. Examples of images produced by P_A ordered according to check point epochs. Starting from epoch 40 images retain high-quality.



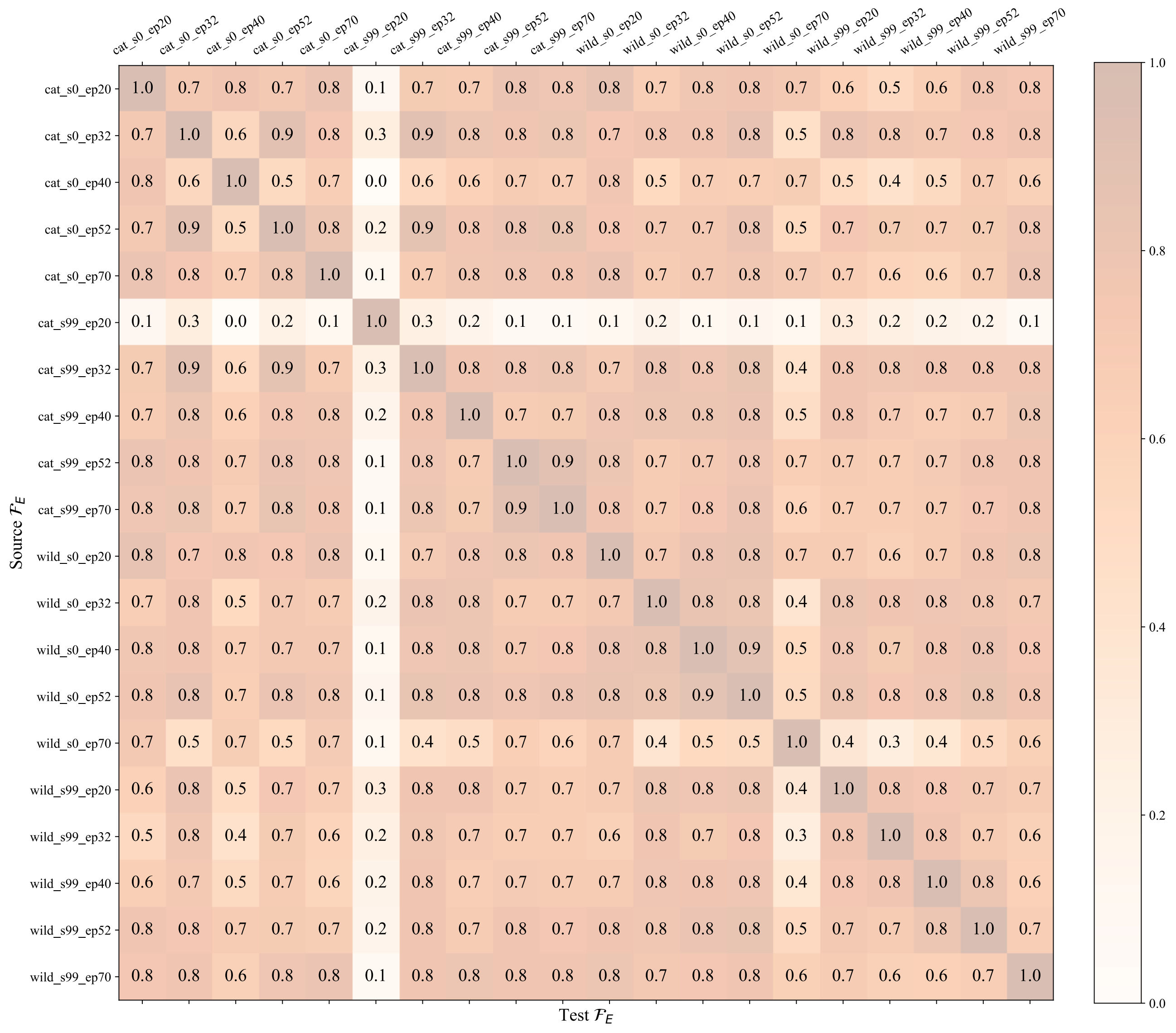
	$(P_A, 20)$	$(P_A, 32)$	$(P_A, 40)$	$(P_A, 52)$	$(P_A, 70)$	$(P_B, 20)$	$(P_B, 32)$	$(P_B, 40)$	$(P_B, 52)$	$(P_B, 70)$	$(\hat{P}_A, 20)$	$(\hat{P}_A, 32)$	$(\hat{P}_A, 40)$	$(\hat{P}_A, 52)$	$(\hat{P}_A, 70)$	$(\hat{P}_B, 20)$	$(\hat{P}_B, 32)$	$(\hat{P}_B, 40)$	$(\hat{P}_B, 52)$	$(\hat{P}_B, 70)$
$(P_A, 20)$	99.5	71.9	51.4	51.4	50.6	81.2	83.9	55.6	52.3	50.8	52.2	49.6	49.8	49.6	56.7	56.2	55.0	59.1	61.5	53.7
$(P_A, 32)$	76.6	99.7	58.8	53.5	51.3	67.2	61.1	51.1	52.7	52.6	50.5	49.8	49.8	49.5	53.7	56.8	57.1	70.3	70.1	54.4
$(P_A, 40)$	57.5	96.7	99.0	94.7	82.8	63.6	52.6	57.6	69.5	55.1	52.0	64.2	79.3	83.4	63.3	64.4	59.5	67.6	78.1	78.2
$(P_A, 52)$	84.6	73.6	96.2	99.4	89.6	76.3	83.1	71.7	69.7	66.7	68.5	56.4	52.0	51.2	61.9	85.1	84.7	95.2	97.0	73.1
$(P_A, 70)$	68.5	75.7	86.0	95.1	99.3	80.5	69.4	60.0	62.9	66.0	56.5	55.1	53.2	51.6	71.7	74.4	72.1	84.1	96.9	87.2
$(P_B, 20)$	49.9	49.9	50.0	50.0	50.0	99.8	72.2	49.9	50.0	49.9	49.7	49.8	49.8	49.8	49.7	49.8	49.7	50.4	51.7	50.0
$(P_B, 32)$	68.1	54.3	58.6	51.7	50.2	86.4	99.7	52.9	52.3	51.3	58.3	50.0	49.5	49.5	51.7	76.1	60.4	58.9	61.2	51.7
$(P_B, 40)$	54.5	74.1	55.1	55.1	53.9	59.7	57.0	99.5	95.2	77.8	50.5	60.6	82.3	84.2	64.7	52.9	50.8	50.0	50.1	49.9
$(P_B, 52)$	53.5	54.6	73.5	67.1	54.5	72.5	70.5	98.8	99.3	91.4	49.9	56.1	52.9	50.5	65.8	66.9	60.6	57.7	68.9	54.4
$(P_B, 70)$	74.1	57.3	58.5	65.2	57.6	67.2	70.7	98.8	95.8	99.3	53.3	59.6	53.0	51.0	64.3	81.4	72.9	68.0	79.9	58.6
$(\hat{P}_A, 20)$	84.5	54.8	50.3	49.8	49.8	72.5	80.5	51.3	51.2	49.9	99.5	61.6	73.0	77.3	53.1	80.7	52.6	51.4	51.2	54.6
$(\hat{P}_A, 32)$	67.9	56.9	50.6	54.9	50.8	71.3	68.5	65.9	55.9	62.4	60.6	99.4	96.2	89.3	60.4	67.4	65.4	67.0	57.9	53.6
$(\hat{P}_A, 40)$	76.0	86.6	52.2	57.0	52.3	85.3	54.9	65.8	64.1	62.9	55.7	81.0	99.3	97.7	73.9	52.9	56.0	62.4	61.9	60.6
$(\hat{P}_A, 52)$	77.0	81.8	53.0	56.7	61.3	78.4	66.3	65.4	62.7	60.6	61.6	69.0	96.9	99.4	81.5	55.6	57.6	56.3	57.1	73.1
$(\hat{P}_A, 70)$	76.6	82.8	67.2	87.9	84.3	72.9	89.3	84.3	73.9	70.4	58.6	74.1	71.4	76.1	98.9	72.8	69.4	88.0	93.9	85.0
$(\hat{P}_B, 20)$	61.5	52.6	50.2	52.1	50.4	61.3	79.4	59.5	52.8	56.2	80.9	85.6	84.4	90.4	67.3	99.4	73.8	57.8	53.1	57.5
$(\hat{P}_B, 32)$	84.6	73.9	53.2	64.8	57.2	64.5	71.7	69.8	57.3	59.5	70.2	94.6	93.4	96.1	83.4	93.0	99.1	92.4	74.8	78.4
$(\hat{P}_B, 40)$	81.2	90.2	55.6	78.7	59.6	88.3	83.7	56.7	57.3	52.8	63.8	91.0	93.8	94.5	81.6	67.3	80.3	99.4	94.1	94.1
$(\hat{P}_B, 52)$	91.2	94.0	58.4	76.9	71.2	80.1	53.1	58.0	61.2	57.6	64.9	70.7	92.3	94.7	73.4	57.9	65.4	94.6	99.5	97.7
$(\hat{P}_B, 70)$	85.1	80.1	55.1	63.0	64.2	68.3	68.0	50.3	54.2	53.9	55.8	55.9	51.1	51.9	68.5	72.4	78.5	95.8	99.2	99.6

Figure H. Cross-detection accuracy (%) for ProGAN models P_A, P_B, $\hat{P}_A$ and $\hat{P}_B$. Clusters of high cross-detection accuracy re-appear for each model at epoch 40,52 and 70.



	cat_s0_ep20	cat_s0_ep32	cat_s0_ep40	cat_s0_ep52	cat_s0_ep70	cat_s99_ep20	cat_s99_ep32	cat_s99_ep40	cat_s99_ep52	cat_s99_ep70	wild_s0_ep20	wild_s0_ep32	wild_s0_ep40	wild_s0_ep52	wild_s0_ep70	wild_s99_ep20	wild_s99_ep32	wild_s99_ep40	wild_s99_ep52	wild_s99_ep70
cat_s0_ep20	1.0	0.2	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.1
cat_s0_ep32	0.2	1.0	0.3	0.2	0.2	0.1	0.1	0.1	0.2	0.2	0.0	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.2
cat_s0_ep40	0.0	0.3	1.0	0.5	0.4	0.0	0.1	0.2	0.2	0.1	0.1	0.2	0.2	0.2	0.2	0.0	0.1	0.1	0.3	0.2
cat_s0_ep52	0.0	0.2	0.5	1.0	0.6	0.0	0.0	0.2	0.2	0.2	0.1	0.2	0.2	0.2	0.2	0.0	0.1	0.2	0.2	0.2
cat_s0_ep70	0.0	0.2	0.4	0.6	1.0	0.0	0.1	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.1	0.1	0.1	0.3	0.3
cat_s99_ep20	0.1	0.1	0.0	0.0	0.0	1.0	0.6	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0
cat_s99_ep32	0.0	0.1	0.1	0.0	0.1	0.6	1.0	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.1	0.1
cat_s99_ep40	0.0	0.1	0.2	0.2	0.2	0.0	0.2	1.0	0.5	0.3	0.1	0.2	0.2	0.2	0.3	0.0	0.1	0.2	0.2	0.2
cat_s99_ep52	0.0	0.2	0.2	0.2	0.2	0.0	0.2	0.5	1.0	0.5	0.0	0.2	0.3	0.2	0.2	0.0	0.2	0.2	0.2	0.2
cat_s99_ep70	0.1	0.2	0.1	0.2	0.2	0.0	0.1	0.3	0.5	1.0	0.1	0.3	0.3	0.2	0.2	0.0	0.2	0.2	0.2	0.2
wild_s0_ep20	0.1	0.0	0.1	0.1	0.2	0.1	0.1	0.1	0.0	0.1	1.0	0.2	0.2	0.2	0.2	0.0	0.0	0.0	0.1	0.1
wild_s0_ep32	0.1	0.2	0.2	0.2	0.2	0.1	0.1	0.2	0.2	0.3	0.2	1.0	0.5	0.4	0.3	0.0	0.3	0.2	0.3	0.2
wild_s0_ep40	0.1	0.2	0.2	0.2	0.3	0.0	0.1	0.2	0.3	0.3	0.2	0.5	1.0	0.6	0.5	0.0	0.2	0.2	0.3	0.3
wild_s0_ep52	0.1	0.2	0.2	0.2	0.3	0.0	0.1	0.2	0.2	0.2	0.2	0.4	0.6	1.0	0.6	0.1	0.2	0.2	0.3	0.4
wild_s0_ep70	0.0	0.1	0.2	0.2	0.3	0.0	0.1	0.3	0.2	0.2	0.2	0.3	0.5	0.6	1.0	0.1	0.2	0.3	0.3	0.4
wild_s99_ep20	0.0	0.1	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	1.0	0.3	0.2	0.2	0.1
wild_s99_ep32	0.0	0.1	0.1	0.1	0.1	0.0	0.1	0.1	0.2	0.2	0.0	0.3	0.2	0.2	0.2	0.3	1.0	0.5	0.3	0.3
wild_s99_ep40	0.0	0.1	0.1	0.2	0.1	0.0	0.0	0.2	0.2	0.2	0.0	0.2	0.2	0.2	0.3	0.2	0.5	1.0	0.5	0.4
wild_s99_ep52	0.1	0.1	0.3	0.2	0.3	0.0	0.1	0.2	0.2	0.2	0.1	0.3	0.3	0.3	0.3	0.2	0.3	0.5	1.0	0.5
wild_s99_ep70	0.1	0.2	0.2	0.2	0.3	0.0	0.1	0.2	0.2	0.2	0.1	0.2	0.3	0.4	0.4	0.1	0.3	0.4	0.5	1.0

Figure I. Cross-correlation of $\mathcal{F}_A$ for ProGAN models $P_A, P_B, \hat{P}_A$ and $\hat{P}_B$. Values are symmetric for all the models and cluster of high values (above 0.5) are smaller then in Fig. H.



	cat_s0_ep20	cat_s0_ep32	cat_s0_ep40	cat_s0_ep52	cat_s0_ep70	cat_s99_ep20	cat_s99_ep32	cat_s99_ep40	cat_s99_ep52	cat_s99_ep70	wild_s0_ep20	wild_s0_ep32	wild_s0_ep40	wild_s0_ep52	wild_s0_ep70	wild_s99_ep20	wild_s99_ep32	wild_s99_ep40	wild_s99_ep52	wild_s99_ep70
cat_s0_ep20	1.0	0.7	0.8	0.7	0.8	0.1	0.7	0.7	0.8	0.8	0.8	0.7	0.8	0.8	0.7	0.6	0.5	0.6	0.8	0.8
cat_s0_ep32	0.7	1.0	0.6	0.9	0.8	0.3	0.9	0.8	0.8	0.8	0.7	0.8	0.8	0.8	0.5	0.8	0.8	0.7	0.8	0.8
cat_s0_ep40	0.8	0.6	1.0	0.5	0.7	0.0	0.6	0.6	0.7	0.7	0.8	0.5	0.7	0.7	0.7	0.5	0.4	0.5	0.7	0.6
cat_s0_ep52	0.7	0.9	0.5	1.0	0.8	0.2	0.9	0.8	0.8	0.8	0.8	0.7	0.7	0.8	0.5	0.7	0.7	0.7	0.7	0.8
cat_s0_ep70	0.8	0.8	0.7	0.8	1.0	0.1	0.7	0.8	0.8	0.8	0.8	0.7	0.7	0.8	0.7	0.7	0.6	0.6	0.7	0.8
cat_s99_ep20	0.1	0.3	0.0	0.2	0.1	1.0	0.3	0.2	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.3	0.2	0.2	0.2	0.1
cat_s99_ep32	0.7	0.9	0.6	0.9	0.7	0.3	1.0	0.8	0.8	0.8	0.7	0.8	0.8	0.8	0.4	0.8	0.8	0.8	0.8	0.8
cat_s99_ep40	0.7	0.8	0.6	0.8	0.8	0.2	0.8	1.0	0.7	0.7	0.8	0.8	0.8	0.8	0.5	0.8	0.7	0.7	0.7	0.8
cat_s99_ep52	0.8	0.8	0.7	0.8	0.8	0.1	0.8	0.7	1.0	0.9	0.8	0.7	0.7	0.8	0.7	0.7	0.7	0.7	0.8	0.8
cat_s99_ep70	0.8	0.8	0.7	0.8	0.8	0.1	0.8	0.7	0.9	1.0	0.8	0.7	0.8	0.8	0.6	0.7	0.7	0.7	0.7	0.8
wild_s0_ep20	0.8	0.7	0.8	0.8	0.8	0.1	0.7	0.8	0.8	0.8	1.0	0.7	0.8	0.8	0.7	0.7	0.6	0.7	0.8	0.8
wild_s0_ep32	0.7	0.8	0.5	0.7	0.7	0.2	0.8	0.8	0.7	0.7	0.7	1.0	0.8	0.8	0.4	0.8	0.8	0.8	0.8	0.7
wild_s0_ep40	0.8	0.8	0.7	0.7	0.7	0.1	0.8	0.8	0.7	0.8	0.8	0.8	1.0	0.9	0.5	0.8	0.7	0.8	0.8	0.8
wild_s0_ep52	0.8	0.8	0.7	0.8	0.8	0.1	0.8	0.8	0.8	0.8	0.8	0.8	0.9	1.0	0.5	0.8	0.8	0.8	0.8	0.8
wild_s0_ep70	0.7	0.5	0.7	0.5	0.7	0.1	0.4	0.5	0.7	0.6	0.7	0.4	0.5	0.5	1.0	0.4	0.3	0.4	0.5	0.6
wild_s99_ep20	0.6	0.8	0.5	0.7	0.7	0.3	0.8	0.8	0.7	0.7	0.7	0.8	0.8	0.8	0.4	1.0	0.8	0.8	0.7	0.7
wild_s99_ep32	0.5	0.8	0.4	0.7	0.6	0.2	0.8	0.7	0.7	0.7	0.6	0.8	0.7	0.8	0.3	0.8	1.0	0.8	0.7	0.6
wild_s99_ep40	0.6	0.7	0.5	0.7	0.6	0.2	0.8	0.7	0.7	0.7	0.7	0.8	0.8	0.8	0.4	0.8	0.8	1.0	0.8	0.6
wild_s99_ep52	0.8	0.8	0.7	0.7	0.7	0.2	0.8	0.7	0.8	0.7	0.8	0.8	0.8	0.8	0.5	0.7	0.7	0.8	1.0	0.7
wild_s99_ep70	0.8	0.8	0.6	0.8	0.8	0.1	0.8	0.8	0.8	0.8	0.8	0.7	0.8	0.8	0.6	0.7	0.6	0.6	0.7	1.0

Figure J. Cross-correlation of $\mathcal{F}_E$ for ProGAN models $P_A, P_B, \hat{P}_A$ and $\hat{P}_B$. Relation appears to be random.

References

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv*, 2019. 2

[2] 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. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018. 2

[3] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. *CoRR*, abs/1912.01865, 2019. 3

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 1

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv*, 2018. 2

[6] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4217–4228, 2021. 2

[7] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020. 2

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 1

[9] Francesco Marra, Diego Gragnaniello, Luisa Verdoliva, and Giovanni Poggi. Do gans leave artificial fingerprints? *arXiv*, 2018. 1

[10] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. 2

[11] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. *arXiv*, 2019. 2

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017. 1

[13] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv*, abs/2208.12242, 2022. 2

[14] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models, 2022. 2

[15] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A. Efros. Cnn-generated images are surprisingly easy to spot... for now. *arXiv*, 2019. 2

[16] 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. *CoRR*, abs/1506.03365, 2015. 2

[17] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, Jul 2017. 1