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 \( F \) 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).

<table>
<thead>
<tr>
<th>Method</th>
<th>Gaussian High-Pass</th>
<th>DnCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marra18</td>
<td>52.4</td>
<td>62.6</td>
</tr>
<tr>
<td>Joslin20</td>
<td>51.0</td>
<td>51.7</td>
</tr>
<tr>
<td>DIF</td>
<td>73.3</td>
<td>92.3</td>
</tr>
</tbody>
</table>

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 \( \times \) 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 (\( F_{DnCNN} \)). During the inference of the model, this fingerprint was subtracted from the output according to:

\[ R_i = f_D(X_i) - F_{DnCNN} \]  

(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 \( 2 \times 2 \) pixels for down-sampling and a deconvolution layer with a kernel size of \( 2 \times 2 \) pixels and stride of 2 pixels for up-sampling. The latter is suspected to be the
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. 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 2.0R2, 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
Table C. Detection accuracy (%) of CNN architectures on images from LTMs. D-Net is a up-sampling network and C-Net convolutional network. U-Net demonstrates best results.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>SD 1.4</th>
<th>SD 2.1</th>
<th>MJ</th>
<th>DALL-E-Mini</th>
<th>GLIDE</th>
<th>DALL-E-2</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-Net</td>
<td>99.3</td>
<td>89.5</td>
<td>99.0</td>
<td>99.0</td>
<td>90.3</td>
<td>79.5</td>
<td>92.8</td>
</tr>
<tr>
<td>U1-Net</td>
<td>99.4</td>
<td>88.8</td>
<td>98.9</td>
<td>98.3</td>
<td>85.2</td>
<td>80.2</td>
<td>91.8</td>
</tr>
<tr>
<td>D-Net</td>
<td>99.4</td>
<td>89.3</td>
<td>98.9</td>
<td>99.0</td>
<td>89.0</td>
<td>78.0</td>
<td>91.9</td>
</tr>
<tr>
<td>C-Net</td>
<td>60.7</td>
<td>53.3</td>
<td>89.9</td>
<td>98.0</td>
<td>88.2</td>
<td>69.6</td>
<td>76.6</td>
</tr>
</tbody>
</table>

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

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

C.3. Cross-Detection for GAN Models

We report the complete map of cross-detection for datasets corresponding to GAN and ProGAN$_i$ 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 ($128 \times 128$ 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.
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 ($F_E$) exhibit a random pattern (Fig. J), while the fingerprints obtained through residual averaging ($F_A$) display a trend similar to cross-detection (Fig. I). However, the cross-detection of $F_A$ decreases more rapidly with epochs, lacks normalization, and is always symmetric. As a result, the certainty of
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 checkpoint epochs. Starting from epoch 40, images retain high-quality.

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


