Inpainting Detection

Paul Adler Hakim Guenther Damaris Gann Hannes Ehringfeld

Abstract

This report presents a series of experiments conducted to develop a robust image inpainting detector, focusing particularly on the noise patterns present in the images. We tested various training data creation methods and model architectures. Our findings suggest that the noise patterns characteristic of modern inpainting pipelines can be effectively learned and detected without the need for specialized model architectures. Further, we determined that the noise pattern is predominantly a product of the Variational Autoencoder used in the inpainting pipelines. Consequently, the actual creation of inpaintings may be unnecessary for generating training data. The exclusive use of the Autoencoder considerably simplifies the scalability of the training data generation process. The code of this project is available on GitHub [1] and the custom dataset can be found on Huggingface[2].

1. Introduction

With the continual improvement of image inpainting tools, the ability to detect manipulated images is becoming increasingly important. This is especially true for preventing the spread of misinformation and preserving the integrity of visual data. One distinguishing feature between authentic and inpainted image areas is the noise pattern present in the image. Although noise patterns vary between different cameras and image editing tools, AI-based tools, such as stable diffusion or midjourney, may exhibit common characteristics. This project explored this premise by examining two model architectures and various training data generation methods.

We tested different methods of training data creation, establishing that the use of the Variational Autoencoder (VAE) in an inpainting pipeline can efficiently generate training data. We also demonstrated that standard models could effectively learn and detect noise patterns characteristic of modern inpainting pipelines, even outperforming specialized architectures on our dataset.

[1] https://github.com/Hanneseh/MPDL_Project_2

[2] https://huggingface.co/datasets/hakimgunther/inpainting

The report is structured as follows: We initially outline the model architectures in the Approach section. Our experimental procedures, including the dataset generation process and the training cycles implemented for each model, are described in the Experiments section. In the Results section we organize our testing results in tables and make some key observations before we interpret them in the final Discussion section.

2. Approach

To develop the inpainting detector, we employed a relatively recent model architecture known as NIX-Net, as proposed by Li et al. (2021). For comparative analysis, we trained a baseline model from the Fully Convolutional Network (FCN) class, as proposed by Shelhamer et al. (2016). Our datasets were custom made and based on a 10k LAION and 15k COCO subset. Its creation process is described in Datasets.

2.1. Noise-Image Cross-fusion Network (NIX-Net)

The NIX-Net is an architecture that enhances the detection performance of inpainted images by leveraging both the image and its noise residual. It stands apart from conventional models by valuing the importance of noise residual patterns for mask detection. The architecture is composed of three key components: feature extraction, multi-scale cross fusion, and mask detection. At its heart is the feature extraction component, which defines the noise residual of an inpainted image and applies an SRM (Spectral Residual Magnitude) filter inspired by recent studies in image forensics. This process includes two parallel streams that learn from multi-scale feature maps of the input image and its noise residual, thus ensuring a comprehensive understanding of the features (Li et al., 2021). Since there was no implementation provided along with the paper, we had to create one based on the information given.
While the NIX-Net's structure is well-explained, a notable gap arises in explaining dimensions, especially channel specifics, within Convolutional Layers. This gap complicates network size estimation, with dimensions and channels influencing the network's parameter count significantly. Depending on choices, the network's parameter count can vary from approximately 28 million to approximately 140

million, impacting computational and performance considerations.

2.2. Fully Convolutional Network (FCN)

The Fully Convolutional Network (FCN) is a classic semantic segmentation architecture. Trained end-to-end on a pixel-to-pixel basis, FCNs introduced the capability to accept inputs of any size and generate correspondingly sized outputs, significantly improving upon previous methods in semantic segmentation. The model's design adapted pre-existing classification networks such as AlexNet, VGGNet, and GoogLeNet into fully convolutional formats and fine-tuned their learned representations to the segmentation task (Shelhamer et al., 2016). We chose to use a FCN with a pre-trained ResNet 50 backbone since it is available in pytorch and is therefore easy to employ with short training cycles (PyTorch, 2023).

2.3. Datasets

The custom dataset that is based on the 10k LAION subset was provided to us at the beginning of the project, so we call this Dataset 1. Later in the project, after we conducted some experiments, we decided to create Dataset 2 based on the 15k COCO subset to increase the complexity for the models. The way both datasets were created is similar but not equal. To understand the differences, we first need to explain our naming scheme. We have images with the following names: original, mask, inpainted, realfake and fakefake.

The original image is just the image that was drawn from one of the base datasets. We only had those available for Dataset 2, which we created ourselves.

The mask image is a binary mask with Gaussian blurring applied to its edges. This blurring helps to smooth the transition between the inpainted and original areas in later steps. In Dataset 1, these masks were randomly generated rectangles. For Dataset 2, we created one third of the masks by applying the YOLOv8 object detector (Jocher et al., 2023) to the image first, using the detected bounding box of a random class as the mask. This way, the mask is linked to a specific object in the image, which could be entirely replaced by the inpainting pipeline. Another third of the masks were generated by applying the MaskFormer (Cheng et al., 2021), available through Hugging Face, to the images. Again, the segmentation mask related to a random class is used as the mask for the inpainting pipeline. The last third of the masks was not generated. Instead, we used the "QD-IMD: Quick Draw Irregular Mask Dataset" (Iskakov, 2018) to further diversify the masks in our dataset. We saved the masks with the Gaussian blur as RGB images, where all color channels contained the same values ranging from 0 to 255.

The inpainted image is the output of the Stable Diffusion 1.5 inpainting pipeline (Rombach et al., 2022). The pipeline takes an image, a mask, and a text prompt to generate the inpainting. We used datasets with textual descriptions of the content in each image as our base dataset. This allowed us to use the textual descriptions as prompts for our model. For Dataset 1, the inpainted images were not available, but we saved them as we created Dataset 2.

The realfake images are compositions of the inpainted and original versions. The inpainted version is present in the regions corresponding to the mask. The realfake images were created since we suspected the inpainting pipeline to evenly distribute the noise across the entire image, even if only specific parts were actually inpainted.

The fakefake images are compositions of the original and reconstructed versions of the same image. The reconstructed version is the output of the VAE available in the stable diffusion 1.5 inpainting pipeline. The reconstructed version is present in the regions corresponding to the mask. The fakefake images were created to check whether the actual inpainting step in the pipeline is responsible for creating specific noise patterns, or if that is a result of the autoencoder step.

The exact sizes and splits of the datasets we used for training are listed in Table 1. The splits remain the same for all models and training cycles. Bear in mind that Dataset 2 includes Dataset 1. Thus, when we trained with Dataset 2 on inpainted images directly, we had fewer examples than when we trained on realfake or fakefake images, as there are no inpainted images available from Dataset 1.

Table 1. Dataset Split Sizes

Dataset	Train	Validation	Test
Dataset 1	9000	500	500
Dataset 2	20878	1986	1984
Dataset 2 (inpainted)	11878	1486	1484

3. Experiments

Each model was trained multiple times, varying the number of parameters, hyperparameters, input data, and mask handling. In this section, we provide an overview of the experimental setups.

3.1. FCN Baseline Experiments

We performed seven training cycles for the FCN model, with the same number of parameters and hyperparameter configuration. We modified the last layer of the model to output binary segmentation masks, as the original implementation outputs 20 distinct classes. The FCN model has about 35 million trainable parameters. In all training cycles, we used

an early stopper to halt training after 5 consecutive epochs without improvement in the validation loss. We evaluated the models based on the Intersection over Union (IoU) metric. The following hyperparameters were consistently used in training: the number of epochs was set to 100, the batch size was 8, and the learning rate was 0.0001. All training was done on a NVIDIA RTX 2080TI. The loss function utilized was the Sigmoid focal loss, which is a common choice for semantic segmentation tasks, as it helps to address the problem of imbalanced classes. The optimizer used was Adam.

In the first three training cycles, all on realfake images of Dataset 1, we tested different ways of handling the ground truth segmentation masks. As mentioned above, the edges of the segmentation masks have a Gaussian blurring applied to them, which begs the question: at what gray level should we consider the mask to start?

In the first training run, we did not apply any specialized handling of the mask, since we were not aware of the problem. This led to an IoU on the test set of 0.827.

In the second cycle, we explicitly turned the masks into binary in the dataloader, such that all pixels with a gray value higher than 0 would become 1. This kind of mask handling led to an IoU of 0.824.

In the third training cycle, we tested the other extreme and only turned gray values equal to 255 to 1, the others to 0. This approach yielded the best result of a test IoU of 0.944. Hence, we used this kind of mask handling in all subsequent training iterations.

In total, we created 5 versions of the FCN baseline, all trained on different data. For a complete overview, refer to Table 3 and 4 in the Results section.

3.2. NIX-Model Experiments

During the training on the different datasets we tested many configurations of the NIX-Net with a different amount of trainable parameters. As already stated the Network was not fully explained. Different NIX-Nets with slightly changes in the channel and dimension declaration has therefore been tested. In the following the different NIX-Net will be called followed by their amount of trainable parameters in order of keeping track of them.
In all training cycles, we used an early stopper to halt training after 5 consecutive epochs without improvement in the validation loss. We evaluated the models based on the Intersection over Union (IoU) metric. The following hyperparameters were consistently used in training: the number of epochs was set to 100 with varying batch sizes depending on the model, and the learning rate was 0.0001. All training was done on a NVIDIA A40. The loss function utilized was the Sigmoid focal loss. The optimizer used was Adam.

In contrast to the different training cycles for the FCN the NIX-Net has been trained with masks shown in the third training cycle of the FCN.
The choice of models and their parameters was determined based on the datasets used. For Dataset 1, the initial NIX-Net implementation with around 40 million parameters was selected, serving as the foundational baseline. Subsequently, for experiments involving Dataset 2 and its inpainted variant, a NIX-Net configuration with 60 million parameters was adopted.
Preliminary assessments of NIX-Net performance were conducted using RealFake images, where the version with 60 million parameters exhibited the best results (see table 2). To ensure a fair comparison across results, the NIX-Net models trained on FakeFake or inpainted images for Dataset 2 were also configured with 60 million parameters.

Table 2. IoU of different NIX-Nets on Dataset 2

Parameters	**Trained & Tested on**	**IoU**
$\sim$ 40 million	realfake	0.769
$\sim$ 60 million	realfake	0.869
$\sim$ 85 million	realfake	0.683

4. Results

In this section we list the results from our experiments as well as some example images with ground truth and segmentation masks. We first report the results of the baseline, than the ones of the NIX-Model.

4.1. FCN Baseline Results

In Table 3, we list the training results from Dataset 1. We trained one model on realfake images and another on fakefake images, then evaluated each on both test sets. What's interesting is that the model trained on realfake images also performs very well on fakefake images. The reverse is not as successful, yet it still achieves a fairly high Intersection over Union score.

Table 3. FCN Training Results on Dataset 1

Trained on	**Tested on**	**IoU**
realfake	realfake	0.944
realfake	fakefake	0.934
fakefake	fakefake	0.950
fakefake	realfake	0.783

In Table 4, we present the results of three different models trained on the larger and more complex Dataset 2. Each of these models was trained on either realfake, fakefake, or inpainted images. The observations we made from the results in Table 3 regarding the performance of a realfake

model on fakefake images and vice versa still apply. If we do not care about precise region detection, we could simply train on fakefake images and would still detect the inpainting, but not in its entirety.

Table 4. FCN Training Results on Dataset 2

Trained on	Tested on	IoU
realfake	realfake	0.909
realfake	fakefake	0.886
realfake	inpainted	0.213
fakefake	realfake	0.762
fakefake	fakefake	0.909
fakefake	inpainted	0.211
inpainted	realfake	0.141
inpainted	fakefake	0.027
inpainted	inpainted	0.172

We also see in Table 4 that the model trained on inpainted data directly is unable to reliably detect any inpainting kind in any of the three image classes. The reason for this becomes apparent if we take a look at the test we did on the inpainted images with the model that was trained on realfake or fakefake images. The model simply predicts the entire image to be inpainted which hints at the fact that the noise pattern of the inpainting pipeline is equally distributed throughout the inpainted images. Refer to the Appendix for example images with predictions. Therefore, if we train on inpainted images directly, the model is not able to learn from the masks we provide because it does not see any distinguishable pattern.

4.2. NIX-Net Results

We conducted the same training process on two instances of the NIX-Net with 40 million paramaters, one on the realfake and another on the fakefake images of Dataset 1, mirroring the approach we took for the baseline. The results of this exercise are presented in Table 5. Upon close examination of the IoU scores across all tests, we noticed a pattern mirroring that in the results of the baseline, as presented in Table 3. The model trained on realfake images demonstrated superior precision in detecting inpainted regions on fakefake images, compared to the fakefake-trained model's ability to identify inpaintings in realfake. Surprisingly, although the absolute values of the IoU scores are comparable to those of the baseline, they are marginally worse for each test. This was unexpected, given that we anticipated the specialized model architecture to offer an advantage over the baseline.

In subsequent experiments, we continued to replicate the approach used for the baseline, training three separate instances of NIX-Net with 60 million parameters. Each instance was trained on realfake, fakefake, and inpainted images respectively. The results, presented in table 6, once again reflect the pattern observed in the baseline results in table 4.

Table 5. Nix-Net Training Results on Dataset 1

Trained on	Tested on	IoU
realfake	realfake	0.883
realfake	fakefake	0.916
fakefake	fakefake	0.905
fakefake	realfake	0.759

Noteworthy in this case is the inability to generalize to the inpainted dataset. Even though the loss did decrease during training, the model failed to predict a mask for the inpainted area. At some point during training, the Intersection over Union (IoU) between the real and predicted masks converged to 0.0 and did not recover to values above 0.0. This behavior repeated multiple times while training on Dataset 2 for all three methods. While we have achieved useful results for the realfake and fakefake images by starting a new training iteration, training on inpainted areas has been more unstable.

The poorer performance of NIX-Net compared to FCN, besides the lack of information about the model structure, could be explained by the data used to train and evaluate the models. Initially, NIX-Net was trained and tested on data generated using GAN approaches, yielding better results. These approaches introduced specific noise patterns, resembling digital fingerprints, in the inpainted areas of the images. NIX-Net was fine-tuned to identify and exploit these noise patterns.

However, in our specific dataset, we adopted different approaches, resulting in images with different and less noise. Consequently, the noise residuals of the images did not provide any new information. Furthermore, the attempted feature extraction and multi-scale cross-fusion of noise residuals within NIX-Net did not yield the intended results. The disparity in noise characteristics between our data and the GAN-generated data could have hindered the successful integration of these techniques.

Table 6. Nix-Net Training Results on Dataset 2

Trained on	Tested on	IoU
realfake	realfake	0.869
realfake	fakefake	0.868
realfake	inpainted	0.214
fakefake	realfake	0.525
fakefake	fakefake	0.785
fakefake	inpainted	0.185
inpainted	realfake	0.0
inpainted	fakefake	0.0
inpainted	inpainted	0.0

5. Discussion

The contemplation of the results leads to the conclusion that, in a real-world scenario where someone uses the inpainting pipeline to manipulate an image, we would not be able to predict the exact region of the inpainting. The person would have to go through the additional step of copying the inpainted region on top of the original image so that our models can distinguish between the noise patterns. However, the models trained on realfake and fakefake images can learn the noise patterns and, from our qualitative observation, they seem to reliably detect inpainted images as being fully inpainted. This information can be used as a baseline in a warning system for manipulated images.

Since we only trained on the noise patterns of a single image inpainting pipeline, we wanted to test if our models were able to extrapolate to the noise patterns of other pipelines. Therefore, we generated 100 training examples with the Stable Diffusion Pipeline Version 2.1 (Previously 1.5) (Rombach et al., 2022) and tested our Dataset 2-trained models on them. The results are listed in 7. Unfortunately, there seems to be no communality between the noise patterns of these two versions, so both models are unable to make good predictions.

Table 7. FCN Extrapolation Results

Trained on	**Tested on**	**IoU**
realfake	realfake sd 2.1	0.070
realfake	fakefake sd 2.1	0.031
fakefake	realfake sd 2.1	0.011
fakefake	fakefake sd 2.1	0.046

Therefore, we suggest that future projects should incorporate noise patterns from different inpainting pipelines so that the model might find commonalities between their noise patterns. Since we showed that training purely on fakefake images is enough to achieve good results, the training data generation can be simplified by using any image and the VAE of the inpainting pipeline (if available). This omits the requirement of having text prompts for each image.

When comparing NIX-Net and FCN models, our findings indicate no requirement for specialized model architectures. Surprisingly, a pretrained FCN surpassed the NIX-Net in performance, despite having fewer trainable parameters. It is worth noting, however, that some ambiguity in the implementation details provided by Li et al. (2021) potentially hindered our ability to accurately replicate their results.

References

Cheng, B., Schwing, A. G., and Kirillov, A. Per-pixel classification is not all you need for semantic segmentation, 2021. URL https://arxiv.org/abs/2107.06278.

Iskakov, K. Quick draw irregular mask dataset, 2018. URL https://github.com/karfly/qd-imd.

Jocher, G., Chaurasia, A., and Qiu, J. Ultralytics yolov8, 2023. URL https://github.com/ultralytics/ultralytics.

Li, A., Ke, Q., Ma, X., Weng, H., Zong, Z., Xue, F., and Zhang, R. Noise doesn't lie: Towards universal detection of deep inpainting. *CoRR*, abs/2106.01532, 2021. URL https://arxiv.org/abs/2106.01532.

PyTorch. FCN-Resnet50 Model. https://pytorch.org/vision/main/models/generated/torchvision.models.segmentation.fcn_resnet50.html, 2023. Accessed: July 15, 2023.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, June 2022.

Shelhamer, E., Long, J., and Darrell, T. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016. URL http://arxiv.org/abs/1605.06211.

6. Appendix

6.1. Pictures of FCN

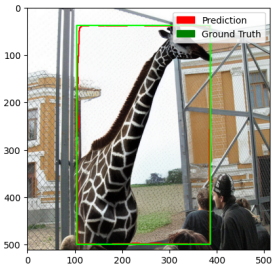

Figure 1. Result of applying the FCN model trained on realfake images to a realfake test image

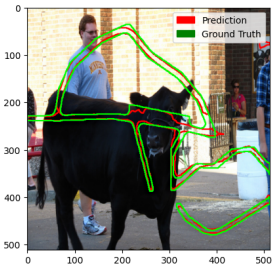

Figure 2. Result of applying the FCN model trained on fakefake images to a fakefake test image

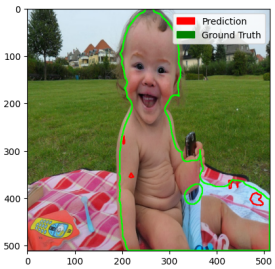

Figure 3. Result of applying the FCN model trained on realfake images to a realfake test image generated by the Stable Diffusion inpaining pipeline Version 2

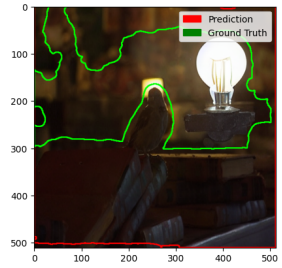

Figure 4. Result of applying the FCN model trained on realfake images to an inpainted image. The model misclassifies the entire image as inpainted due to the evenly distributed noise pattern.

6.2. Breakdown of individual contributions

All additional data has been created by Hakim Guenter. All the tests for the FCN were done by Hannes Ehringfeld. The implementation and testing of the NIX-Net were done by Damaris Gann and Paul Adler.