Abstract

We present a model for differentiating between images that are authentic copies of ones published by photographers, and images that have been manipulated by cropping, splicing or downsampling after publication. The model comprises an encoder that resides with the photographer and a matching decoder that is available to observers. The encoder learns to embed imperceptible positional signatures into image values prior to publication. The decoder learns to use these steganographic positional (“stegapos”) signatures to determine, for each small image patch, the 2D positional coordinates that were held by the patch in its originally-published image. Crop, splice and downsample edits become detectable by the inconsistencies they cause in the hidden positional signatures. We find that training the encoder and decoder together produces a model that imperceptibly encodes position, and that enables superior performance on established benchmarks for splice detection and high accuracy on a new benchmark for crop detection.

1 Introduction

Images are a prime currency of modern communication, rivaled only by text. This trend is growing as social media platforms continue to adopt images as their main tokens of information. In this context, it is increasingly easy for misinformation to be spread by unauthentic images that contain misleading edits. Detecting image tampering has become critically important.

We present a model that detects two types of tampering. One is splicing, where content from one image is mosaiced with content from another image, often with post-processing to increase the perceived authenticity of the result. The other is cropping, where an exterior portion of an image is removed. Cropping is common and harmless when done by a photographer to improve framing or isolate subject matter before the image is shared, but it can be harmful when done later to intentionally shift the image’s context or meaning without the photographer’s consent [11,2].

Most existing forensics methods that detect and localize these types of tampering exploit inconsistencies in image content, both at the semantic level, such as inconsistencies in lighting or object positions, and at the pixel level, such as inconsistencies in camera noise patterns. These methods can be effective for detecting splices (e.g., [31,25,10]) and sometimes even certain crops [26,20]. However, any content-based method is fundamentally limited by the sophistication of the tamperer, and this leads to an “arms race” between forensic methods that use inconsistencies to detect tampering, and counter-forensic methods that use processing or other tricks to suppress inconsistencies [5].

We introduce an alternative strategy based on learned steganography. The idea is to hide a unique positional signature within each small image patch before an image is published, so that subsequent cropping, splicing or downsampling of the image become detectable by the distortions they cause in
the field of hidden positional signatures. For this we introduce a two part, encoder-decoder model that is trained to achieve “steganographic positioning”: maximizing the decoder’s ability to recover each pixel’s absolute position while minimizing the encoder’s perceptual distortion. Experimentally we find that our model learns to successfully encode steganographic positional signatures, and that these stegovpos signatures are quite useful for detecting crops, splices and decimations.

Our contributions are three-fold: (1) we introduce a new image encoding model, called stegovpositional encoding, that inserts decodable absolute position information into images using learned steganographic noise, (2) we show that our model provides competitive performance across a range of splice detection benchmarks without fine-tuning, and (3) we create and publish a new benchmark for crop detection called called SmartCrop21 and report our model’s performance on it.

2 Related Work

Digital steganography. Digital image steganography refers to the act of hiding and recovering data in digital images while preserving visual quality. It is most commonly used to embed external information such as messages or secondary images. There is a trade-off between visual quality and information capacity. When embedding digital signals such as text and binary strings, steganographic systems tend to sacrifice visual quality for perfect reconstruction as these signals typically require exact recovery to be operational. When embedding sampled continuous signals such as images and audio, exact recovery can often be relaxed because they typically do not require perfect recovery to be useful. Our method falls into the second category because it also relaxes away from exact recovery. A key difference, though, is that it requires certain information (position) to be embedded locally in specific regions of the image, as opposed to being distributed globally across the image as a whole.

Content authentication. One way steganography has been used in forensics is for content authentication, where the task is to classify an image as being either authentic or manipulated. Traditional authentication is done by storing hashes of authentic image content in an external database that potentially-tampered images are compared against (e.g., [30]); but recent work has shown that learned steganography can be used to effectively hide the “hash” of an image within it’s own image values, allowing authentication decisions to be made without an external database. Our work also uses learned steganography to detect tampering without an external database or reference image, but it goes beyond authentication by also detecting crop-sizes and splices at the pixel level.

Splice detection. One can often detect splices passively, without pro-actively hiding signatures like we do, by exploiting anomalies in the semantic layout of an image and by exploiting inconsistencies in lighting, shadows, tone mapping, camera noise signatures, and so on. The most effective methods in this class train deep neural networks on large datasets of example splices and produce either splice bounding boxes or pixel-level splice masks. Our model also produces pixel-level splice masks, but because it uses pro-active encoding, it improves performance and is less likely to be fooled in the future as tamperers and their counter-forensic tools continue advancing.

Crop detection. Passively detecting crops is much harder because there are fewer anomalies and inconsistencies to exploit. Semantic anomalies are weak because it is hard to distinguish between (i) a region that was cropped from a published image and (b) a similar authentic image that was framed and published that way by a photographer. As a result, techniques in this category produce relatively coarse crop localizations, and they rely on the presence of radial optical effects like chromatic aberration, radial distortion and vignetting. In contrast, our steganographic approach is more precise and succeeds whether these effects are present or not. For example, it can be used to protect the published assets of digital content creators in addition to photographers’ images.

3 Steganographic Positional Encoding and Decoding

3.1 Encoder

The input to the encoder is a captured image $I$ of size $H \times W \times C$ with $C$ channels and values $I(x, y, c) \in [0, 1]$. The output is an encoded image $\hat{I}$ of the same size and range. The encoder is
Figure 1: The encoder concatenates the input image $I$ with a pre-set field of sinusoidal positional codes $\Psi$ and then maps these to a stega-positional residual $\gamma$ that is added to $I$ (with clamping) to produce stegapos image $\hat{I}$.

depicted in Figure 1 and operates by generating a residual that is added to $I$ with clamping that bounds the residual and restricts the output values to $[0, 1]$.

The residual is meant to include, for every small image patch, sufficient but subtle information about the integer position $(x, y) \in [1, \ldots, W] \times [1, \ldots, H]$ of the patch center. The residual is generated by a fully convolutional U-Net [14, 16] whose input is a channel-wise concatenation of the image $I$ and an $H \times W \times D$ array $\Psi(x, y, \cdot) \in [-1, 1]^D$ of per-pixel positional codes. The codes are adapted from the “positional embedding” used in sequence-to-sequence translation [21] and are tuples of the form

$$\Psi(x, y, \cdot) = \{(\cos(\omega_k x), \sin(\omega_k x), \cos(\omega_k y), \sin(\omega_k y)) \mid \omega_k = \omega_o^{4k/D}, k = 1, \ldots, D/4\} \quad (1)$$

with frequencies $\omega_k$ forming a geometric progression from some sufficiently small base frequency $\omega_o$. These codes have several nice properties. They assign a unique $D$-dimensional embedding to each position $(x, y)$ (provided that $\omega_o$ is sufficiently small); they vary smoothly with position; and they have values in $[-1, 1]$. They also have a linear and shift-invariant relational property that is helpful to many CNNs that process them: codes $\Psi(x, y, \cdot)$ and $\Psi(x + \Delta x, y + \Delta y, \cdot)$ are related by a linear transformation that depends only on $(\Delta x, \Delta y)$, the spatial separation between them (Theorem 3.1, see Supplementary Material C for proof). In our experiments we use dimension $D = 8$, and we follow [21] by using base frequency $\omega_o = 10^{-4}$.

The U-Net has the effect of mixing the per-pixel positional codes and image colors to create a residual $\gamma$ that is imperceptible yet informative about position. The encoder’s full specification is thus

$$\gamma = \text{clamp}_{[-0.2, 0.2]}(\tilde{f}_\theta([I; \Psi])) \quad (2)$$
$$\hat{I} = \text{clamp}_{[0, 1]}(I + \gamma), \quad (3)$$

with $\theta$ being the trainable parameters of the U-Net $\tilde{f}_\theta(\cdot)$. In the sequel we use the notation $\hat{I} = f_\theta(I)$ to denote the full mapping from captured to encoded image, and we refer to $\hat{I}$ as a stega-positional image, or stegapos image for short.

3.2 Decoder

The decoder comprises a standard CNN and a custom linear regression layer. The CNN $p = g_\psi(\hat{I})$ with trainable parameters $\psi$ accepts an $N \times M$ image $\hat{I}$ and uses undecimated (i.e., stride-1) convolutions to produce an output value $p(i, j) \in \mathbb{R}^2$ for each input pixel $i, j$. Each output element $p(i, j)$ is affected by a corresponding $R \times R$ receptive field of input image values, with receptive field size $R$ determined by the filter sizes and number of layers. The CNN output $p$ is of size $(N - R + 1) \times (M - R + 1)$ where each $p(i, j)$ maps the pixel coordinates $(i, j)$ in the stegapos
We train our system end-to-end for accurate patch localization using the loss function

\[ L = \lambda_p ||p - \mathcal{P}_{x,y}^{[0,0],s}||^2_2 + \lambda_I ||\hat{I}_{YUV} - I_{YUV}||_1 + \lambda_{LPIPS} L_{LPIPS} + \lambda_\gamma ||\gamma||_1 + \lambda_{\text{critic}} L_{\text{critic}} \]

3.3 Training

We train our system end-to-end for accurate patch localization using the loss function

\[ L = \lambda_p ||p - \mathcal{P}_{x,y}^{[0,0],s}||^2_2 + \lambda_I ||\hat{I}_{YUV} - I_{YUV}||_1 + \lambda_{LPIPS} L_{LPIPS} + \lambda_\gamma ||\gamma||_1 + \lambda_{\text{critic}} L_{\text{critic}} \]
where \( I_{YUV} \) is an image converted to YUV color space, \( \mathcal{L}_{LPIPS}(I, \hat{I}) \) is the perceptual similarity loss \([29]\) between the encoded image and the original image, and \( \mathcal{L}_{\text{critic}}(I, \hat{I}) \) is an adversarial critic loss adapted from Tancik et al. \([18]\) (see Supplementary Material E.1 for details).

We use a two-phase training regime similar to \([18]\), first training for positional accuracy by setting \( \lambda_1 = \lambda_{\text{critic}} = \lambda_{LPIPS} = \lambda_7 = 0 \), and then training for both position and visual quality with all loss terms until convergence. We find the first phase converges quickly with high positional accuracy but low visual quality, and that visually quality restores when the image regularization coefficients \( \{\lambda_1, \lambda_{\text{critic}}, \lambda_7, \lambda_{LPIPS}\} \) are gradually increased during the second phase.

## 4 Experiments

We train our encoder-decoder architecture on 100,000 images of size \( 400 \times 400 \) from the MIRFLICKR 1M dataset. We use RGB images \((C = 3)\) and dimension \( D = 8 \) with base frequency \( \omega_0 = 10^{-4} \) for the input positional codes. Our decoder receptive field size is \( R = 50 \). During training, we use a batch size of 8 and Adam optimizer with learning rate \( 10^{-7} \) and \((\beta_1, \beta_2) = (0.9, 0.999)\).

To achieve decimation-equivariance in the decoder, we find it sufficient to use a simple augmentation approach during training: We randomly scale each encoded image by \( s \in [0.2, 1.0] \) before feeding it to the decoder and then evaluate the positional loss between the outputs \( p \) and the expected decimated output \( \mathcal{P}([0,0], s) \).

Figure 3 visualizes an example of the encoder’s residual and of the decoder’s output positional fields, and Figure 4 shows some representative results for encoding and decoding. Overall we find that the model is capable of preserving high visual quality while also achieving high positioning accuracy.

### Stegapos or not?

We begin with the authentication task of classifying an image as being stegapos encoded or not. This is important for an observer who wants to verify the source of the image, or who wants to avoid spurious crop or splice detections that could occur in images that are not encoded.

The visualizations in Figure 3(a) suggest that stegapos-encoded images \( \hat{I} \) can be easily distinguished by the positional estimates \( p = g_\psi(\hat{I}) \) they induce. To verify this, we use a classifier which consists of our decoder architecture and an additional layer with a sigmoid activation that accepts \( p \) and outputs the probability that \( \hat{I} \) is stegapos encoded. We train and validate the classifier on a 25,000/10,000-split of the original MIRFLICKR 1M dataset, applying stegapos encoding to half of the images and leaving the rest unencoded. We freeze the encoder and decoder parameters and use back-propagation to optimize the classifier weights using the \( \ell_2 \)-loss between the estimated and ground-truth binary (encoded/unencoded) labels. We observe that our network quickly converges to 100\% training and 99.4\% validation binary classification accuracy (thresholded at \( y = 0.5 \)).
Detecting crops and downsampling. Next we consider the task of estimating the scale and crop-offset of a stegapos image by using the offset and scale estimates \((\mu, s)\) from Equation 6. Figure 4 shows typical results, where the model is able to accurately estimate the cropped regions without sacrificing image quality.

To quantify accuracy, we build and release a new dataset, \textsc{SmartCrop25K}, comprising 25,000 images of size 400 \times 400 from the MIRFLICKR 25K dataset which are cropped at various sizes from full size (no crop) down to 1/16-size (50 \times 50 crop). The cropping was done automatically by a saliency-aware system that preserves salient content. Figure 8 visualizes our model’s quantitative results for offset and scale on this dataset.

Crop localization is harder when an image is downsampled \((s < 1)\) and when the crop retains a smaller fraction of the image. We find that when the input images are not downsampled (first column, \(s = 1\)) the model’s crop offset error is less than 25 pixels for all but the smallest crop sizes. When the images are downsampled (columns 2-5) the error degrades gracefully, providing useful results for larger crops in moderately-scaled images \((s = 0.8, 0.5)\) and then dropping off for smaller crops in severely-scaled images \((s = 0.3)\).

Detecting splices. Finally, we consider the task of detecting two-factor splices, meaning cases where a composite image \(\hat{I}\) has been created by blending two captured images according to \(\hat{I}(i, j) = M(i, j)\hat{I}_1(i, j, c) + (1 - M(i, j))\hat{I}_2(i, j, c)\) with some mask \(M \in [0, 1]\). Our task is to infer the mask \(M\) from image \(\hat{I}\), without prior knowledge of \(\hat{I}_1, \hat{I}_2\). There are two cases to consider: (i) both source images are stegapos images, and (ii) only one of the source images is a stegapos image. In what follows, we evaluate these cases separately.

For intuition, Figure 3(b) visualizes an artificially-challenging situation where a composite image \(\hat{I}\) is created by copies of itself that are encoded with different relative positions, that is \(\hat{I} = (1 - M)\hat{I}_0(\hat{I}) + Mf^{\Delta} (\hat{I})\) with \(f^{\Delta}\) being an encoding with positional codes \(\Psi\) that are phase-shifted by \(\Delta = (\Delta_x, \Delta_y)\). This is challenging because the natural image content contradicts the existence of any splice. Yet, it is clear that the decoded positional field provides a salient signal for splice detection.

The most direct way to estimate mask \(M\) from input image \(\hat{I}\) is using residuals from the decoder’s linear regression. Using the decoder’s output positional field \(p = g_\psi (\hat{I})\) and its output scale and offset estimates \(\hat{\mu}, \hat{s}\), we construct a grid \(P([\hat{\mu}, \hat{s}], \hat{s})\) and estimate the splice mask \(\hat{M}\) as the pixels...
Figure 5: Crop detection errors versus crop size, from $50 \times 50$ (1/16 crop) to $400 \times 400$ (no crop), visualized using 1000 random samples from our 25,000-image dataset. Columns show errors for decreasing scales $s \in \{1.0, 0.8, 0.5, 0.3\}$. Top: Error in estimated top-left corner $(\mu_x, \mu_y)$, measured in pixels. Bottom: Error in estimated scale $s$.

whose raw positional values $p$ diverge most from this grid. That is, with some threshold $\alpha$,

$$\hat{M}(i, j) = |p(i, j) - P_{x,y,(\hat{\mu}_x, \hat{\mu}_y),s}(i, j)| > \alpha.$$ (8)

To evaluate performance, we use established two-factor splice benchmarks [1, 4, 22] that provide labeled pairs $(I^{(i)}, M^{(i)})$. We downsample the images and masks to $400 \times 400$, and we simulate splices that involve stegapos plus unencoded images, $\hat{I} = (1 - M)I + Mf_\theta(\hat{I})$, as well as splices that involve two stegapos images with differently-shifted positional codes, $\hat{I} = (1 - M)f_\theta(\hat{I}) + Mf_\theta\Delta\theta(\hat{I})$ as described above. We refer to the former case as enc-unenc (stegapos only applied to base image) and the latter case as enc-enc (stegapos applied to both the base and the splice images).

The third row of Figure 6 shows examples of our thresholding approach to splice detection, and $F_1$ scores are reported in rows marked “+ Linear” in Table 1. The table includes comparable scores reported in previous literature. A common practice for these benchmarks is to choose the best threshold value $\alpha$ for each test image (e.g., [31, 17, 27]), and the table distinguishes methods that follow this practice (“oracle threshold”) from those that do not require manual selection of per-image thresholds. The table also distinguishes models that are fine-tuned for each benchmark (“fine-tuned”) from those that are trained without exposure to the benchmark images. For completeness, we also include non-oracle thresholded results for our linear methods where a single threshold is selected for each dataset. Overall, we see that simple per-pixel thresholds of our model’s regression residuals provides competitive performance across the benchmarks, with state-of-the-art performance in some cases.

Next we measure improvement that can be gained by replacing the decoder’s U-Net $g_\psi$ by one that is specific to splice detection. The new U-Net $h_\phi(\hat{I})$ accepts an uncropped and undecimated stegapos image $\hat{I}$ and directly estimates a binary mask $\hat{M}$ by thresholding its output at 0.5. We train it after freezing the encoder, and we create a synthetic training set of labeled pairs $(\hat{I}^{(i)}, M^{(i)})$ by combining a 25,000-image subset of the MIRFLICKR 1M dataset with a generated set of simple masks (circles, squares, etc.) using the enc-enc rule. The weights $\phi$ are optimized using $\ell^2$-loss $||M - h_\phi(\hat{I})||_2$.

The splice U-Net benefits from being able to internalize and exploit spatial coherence of mask shapes, and it has the practical benefit of not requiring a manual choice of threshold $\alpha$ for each image. We find that it substantially outperforms all existing splice detectors, without fine-tuning and without oracle thresholding.

5 Conclusion

We propose an encoder-decoder network that takes a color image and injects a steganographic positional signature that allows for accurate patch localization while preserving visual quality. We
Figure 6: Examples of splice detection results. (a) CASIA [4], (b) COVERAGE [22], (c) COLUMBIA [9], (d) NIST-16 [1]. From top to bottom: stegapos images with enc-enc splicing; ground truth mask $\mathcal{M}$; estimated mask $\hat{\mathcal{M}}$ using thresholded linear regression; network-estimated mask $\hat{\mathcal{M}}$.

Table 1: Splice mask $F_1$-scores on four standard datasets. “Fine-tuned” indicates models that are exposed and adjusted to each benchmark during training, and “oracle threshold” indicates models that use the optimal threshold $\alpha$ for each image. For our model, “+ Linear” refers to thresholded linear regression residuals and “+ Network” refers to the dedicated splice detection U-Net. Higher scores are better. Missing entries are those not yet reported in existing literature.

| Fine-tuned | Oracle Threshold | Method         | NIST16 | Columbia | COVER | CASIA |
|------------|-----------------|----------------|--------|----------|-------|-------|
|            |                 | ELA [13]       | 0.236  | 0.470    | 0.222 | 0.214 |
|            |                 | NOI1 [15]      | 0.285  | 0.574    | 0.269 | 0.263 |
|            | ✓               | CFA1 [6]       | 0.174  | 0.467    | 0.190 | 0.207 |
| ✓          |                 | MFCN [17]      | 0.571  | 0.612    | -     | 0.541 |
| ✓          | ✓               | RGB-N [31]     | 0.722  | 0.697    | 0.437 | 0.408 |
| ✓          | ✓               | SPAN [10]      | 0.582  | -        | 0.558 | 0.382 |
| ✓          |                 | Ours + Linear (enc-unenc) | 0.535  | 0.500    | 0.650 | 0.390 |
| ✓          |                 | Ours + Linear (enc-unenc) | 0.658  | 0.569    | 0.706 | 0.585 |
| ✓          |                 | Ours + Network (enc-unenc) | **0.835** | **0.800** | **0.890** | **0.704** |
| ✓          |                 | Ours + Linear (enc-enc) | 0.526  | 0.535    | 0.639 | 0.388 |
| ✓          |                 | Ours + Linear (enc-enc) | 0.659  | 0.606    | 0.719 | 0.595 |
| ✓          |                 | Ours + Network (enc-enc) | **0.843** | **0.809** | **0.881** | **0.733** |

find that this encoding and decoding model provides a useful substrate for the forensics tasks of splice and crop localization. Our method provides competitive performance on popular splice detection benchmarks using a simple linear regression post-processing step and provides one of the first practical approaches to crop detection.

References

[1] Nist-16 dataset. [https://www.nist.gov/itl/iad/mig/open-media-forensics-challenge](https://www.nist.gov/itl/iad/mig/open-media-forensics-challenge).

[2] Why the way an image is cropped can change everything. *The Observers*. [https://observers.france24.com/en/20180824-verification-guide-cropped-photo-video](https://observers.france24.com/en/20180824-verification-guide-cropped-photo-video).
[3] Shumeet Baluja. Hiding images in plain sight: Deep steganography. In Neural Information Processing Systems, 2017.

[4] J. Dong, W. Wang, and T. Tan. Casia image tampering detection evaluation database 2010. http://forensics.idealtest.org

[5] W. Fan, S. Agarwal, and H. Farid. Rebroadcast attacks: Defenses, reattacks, and redefenses. In 2018 26th European Signal Processing Conference (EUSIPCO), pages 942–946, 2018.

[6] P. Ferrara, T. Bianchi, A. De Rosa, and A. Piva. Image forgery localization via fine-grained analysis of cfa artifacts. IEEE Transactions on Information Forensics and Security, 7(5):1566–1577, 2012.

[7] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. arXiv preprint arXiv:1406.2661, 2014.

[8] Jamie Hayes and George Danezis. Generating steganographic images via adversarial training. arXiv preprint arXiv:1703.00371, 2017.

[9] Y.-F. Hsu and S.-F. Chang. Detecting image splicing using geometry invariants and camera characteristics consistency. In International Conference on Multimedia and Expo, 2006.

[10] Xuefeng Hu, Zhihan Zhang, Zhene Yi Jiang, Syomantak Chaudhuri, Zhenheng Yang, and Ram Nevatia. Span: Spatial pyramid attention network for image manipulation localization. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, Computer Vision – ECCV 2020, pages 312–328, Cham, 2020. Springer International Publishing.

[11] David Hume Kennerley. Essay: Chop and crop. The New York Times. https://lens.blogs.nytimes.com/2009/09/17/essay-9/.

[12] Pawel Korus and Nasir Memon. Content authentication for neural imaging pipelines: End-to-end optimization of photo provenance in complex distribution channels. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8621–8629, 2019.

[13] Neal Krawetz and Hacker Factor Solutions. A picture’s worth. Hacker Factor Solutions, 6(2):2, 2007.

[14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[15] Babak Mahdian and Stanislav Saic. Using noise inconsistencies for blind image forensics. Image Vision Comput., 27(10):1497–1503, September 2009.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

[17] Ronald Salloum, Yuzhuo Ren, and C.-C. Jay Kuo. Image splicing localization using a multi-task fully convolutional network (mfcn). Journal of Visual Communication and Image Representation, 51:201–209, Feb 2018.

[18] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2117–2126, 2020.

[19] W. Tang, S. Tan, B. Li, and J. Huang. Automatic steganographic distortion learning using a generative adversarial network. IEEE Signal Processing Letters, 24(10):1547–1551, 2017.

[20] Basile Van Hoorick and Carl Vondrick. Dissecting image crops. arXiv preprint arXiv:2011.11831, 2020.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

[22] B. Wen, Y. Zhu, R. Subramanian, T. Ng, X. Shen, and S. Winkler. Coverage — a novel database for copy-move forgery detection. In 2016 IEEE International Conference on Image Processing (ICIP), pages 161–165, 2016.
[23] Eric Wengrowski and Kristin Dana. Light field messaging with deep photographic steganography. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[24] Pin Wu, Yang Yang, and Xiaoqiang Li. Stegnet: Mega image steganography capacity with deep convolutional network. Future Internet, 10(6):54, Jun 2018.

[25] Y. Wu, W. AbdAlmageed, and P. Natarajan. Mantra-net: Manipulation tracing network for detection and localization of image forgeries with anomalous features. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 9535–9544, 2019.

[26] Ido YerushalmY and Hagit Hel-Or. Digital image forgery detection based on lens and sensor aberration. International journal of computer vision, 92(1):71–91, 2011.

[27] Markos Zampoglou, Symeon Papadopoulos, and Yiannis Kompatsiaris. Large-scale evaluation of splicing localization algorithms for web images. Multimedia Tools Appl., 76(4):4801–4834, February 2017.

[28] Kevin Alex Zhang, Alfredo Cuesta-Infante, Lei Xu, and Kalyan Veeramachaneni. Steganogan: High capacity image steganography with gans. arXiv preprint arXiv:1901.03892, 2019.

[29] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. CoRR, abs/1801.03924, 2018.

[30] Yan Zhao, Shuozhong Wang, Xinpeng Zhang, and Heng Yao. Robust hashing for image authentication using zernike moments and local features. IEEE transactions on information forensics and security, 8(1):55–63, 2012.

[31] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Learning rich features for image manipulation detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1053–1061, 2018.

[32] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In Proceedings of the European conference on computer vision (ECCV), pages 657–672, 2018.
A  Additional crop detection results

Figure 7: Additional crop detection results for $s = \{1.0, 0.8, 0.6\}$. Top: authentic images. Middle: stegapos images. Bottom: ground truth crop (black) and estimated crop (white).
B Additional splice detection results

Figure 8: Additional splice detection results. From top to bottom: stegapos images \(\text{enc-\text{enc}}\), ground truth mask \(M\), linear regression estimated mask \(\hat{M}\), network estimated mask \(\hat{M}\).

C Proof of Theorem 3.1

**Theorem.** Let \(\Psi(x, y, \cdot)\) be defined for all \((x, y) \in [0, W - 1] \times [0, H - 1]\) for some \(W, H \in \mathbb{N}^+\) as

\[
\Psi(x, y, \cdot) = \{(\cos(\omega_k x), \sin(\omega_k x), \cos(\omega_k y), \sin(\omega_k y)) \mid \omega_k = \omega_0^{k/D}, k = 1, \ldots, D/4\}
\]

Then, \(\Psi(x, y, \cdot)\) and \(\Psi(x + \Delta x, y + \Delta y, \cdot)\) are related by a shift-invariant linear transformation:

\[
\Psi(x + \Delta x, y + \Delta y, \cdot) = \Psi(x, y, \cdot) T^T(\Delta x, \Delta y).
\]

**Proof.** We will use proof by construction. Let \(D \in \{4n \mid n \in \mathbb{N}^+\}\).

We define vector

\[
\psi_k(x, y) = [\cos(\omega_k x) \quad \sin(\omega_k x) \quad \cos(\omega_k y) \quad \sin(\omega_k y)]
\]
We provide a 1D walkthrough of our system in three cases shown in Figure 9: (1) no crop, no scale, (2) no crop, s = 0.5, (3) y₀ = 1.0, no scale. For this example, we set R = 1 for simplicity.

**no crop, no scale.** For the no crop, no scale case, we feed the full encoded image into the Decoder (D) which outputs the center coordinates for the 8 non-overlapping patches in the image as \( y = [0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5] \). Linearly regressing \( y \) with the pixel coordinates \( i = [0.5, 1.5, 2.5, 3.5] \), we estimate \( y₀ = 0.0 \) and \( s = 1.0 \).

**no crop, s = 0.5.** For the case where the image is scaled by \( s = 0.5 \), the Decoder outputs \( y = [1.0, 3.0, 5.0, 7.0] \) as the Decoder aims to localize each patch within the original image. Notice that here, unlike in the original case, the location estimates are separated by \( 0.5 \) instead of \( 1 \). This provides us with the primary cue we use to detect and calculate the scale \( s \).

Here, we note that for a no crop, no scale image with the same size as our tampered image, we would expect the pixel coordinates \( i = [0.5, 1.5, 2.5, 3.5] \). Linearly regressing \( y \) with the pixel coordinates \( i \), we estimate \( y₀ = 0.0 \) and \( s = 0.5 \).

**y₀ = 1.0, no scale.** For the case where the image is cropped at \( y₀ = 1.0 \), the Decoder outputs \( y = [1.5, 2.5, 3.5, 4.5] \).

Here, we again note that for a no crop, no scale image with the same size as our tampered image, we would expect the pixel coordinates \( i = [0.5, 1.5, 2.5, 3.5] \). Linearly regressing \( y \) with the pixel coordinates \( i \), we estimate \( y₀ = 1.0 \) and \( s = 0.5 \).
E  Training details

Hyper-parameters for training our encoder-decoder are given in Table 2.

Table 2: Hyper-parameters.

| Parameter            | Value    | Ramp   |
|----------------------|----------|--------|
| learning rate        | $10^{-7}$| -      |
| batch size           | 8        | -      |
| $\lambda_p$          | 6        | 550,000|
| $\lambda_I$          | 250      | 750,000|
| $\lambda_{LPIPS}$    | 350      | 15,000 |
| $\lambda_\gamma$     | 20,000   | 250,000|
| $\lambda_{critic}$   | 350      | 200,000|
| $\lambda_D$          | 350      | 100,000|
| no image loss steps  | 1,000    | -      |
| no critic loss steps | 90,000   | -      |

For hyper-parameters with a ramp value specified, we use a hyper-parameter schedule as

$$
\lambda_i = \min\{\lambda, \lambda_i^{ramp}\}
$$

where $i$ is the current iteration.

E.1  Critic loss ($\mathcal{L}_{critic}$) details

The critic loss employs an auxiliary discriminator that is trained in adversary to our model’s encoder. It derives from GAN losses of the form [7]

$$
\mathcal{L}_D = -\log D(x) - \log(1 - D(G(z))) \quad \text{(discriminator)}
$$

$$
\mathcal{L}_G = -\log D(G(z)) \quad \text{(generator)}
$$
with discriminator $D$, generator $G$, sample $x$ from an input set, and sample $G(z)$ that is generated from a sample of random variable $z$. In this formulation, the discriminator and generator are trained using back-propagation over $L_D$ and $L_G$ respectively. The discriminator’s objective is to correctly classify input samples as $D(x) = 1$ and generated samples as $D(G(z)) = 0$. The generator’s objective is to fool the discriminator by producing samples that evaluate to $D(G(z)) = 1$. At an equilibrium, the discriminator accurately discriminates generated samples from true ones taken from the input set, and the generator produces samples that are concentrated on the input sample distribution.

Inspired by [18] we use a modified version of this for our critic loss, with our encoder $f_\theta(\cdot)$ assuming the role of $G$ and the authentic and stegapos images assuming the roles of $x$ and $G(z)$ respectively. That is,

$$L_D = -\log D(I) - \log(1 - D(f_\theta(I))) \quad \text{(discriminator)}$$
$$L_{\text{critic}} = -\log D(f_\theta(I)). \quad \text{(from Equation 7)}$$

For the discriminator $D$ we use a 5-layer CNN with ReLU activation as in [18].

**F  Network architecture**

We provide breakdowns for the following networks:

- Encoder (sec. 3.1)
- Decoder (sec. 3.2)
- StegaPosDetector (sec. 4)
- SpliceSegmentationNetwork (sec. 4)
Table 3: Encoder architecture.

| Layer (type) | Output Shape | Param # |
|--------------|--------------|---------|
| Conv2d-1     | [-1, 32, 400, 400] | 3,200   |
| Conv2d-2     | [-1, 32, 400, 400] | 0       |
| Conv2d-3     | [-1, 32, 200, 200] | 9,248   |
| Conv2d-4     | [-1, 32, 200, 200] | 0       |
| Conv2d-5     | [-1, 64, 100, 100] | 18,496  |
| Conv2d-6     | [-1, 64, 100, 100] | 0       |
| Conv2d-7     | [-1, 128, 50, 50]  | 73,856  |
| Conv2d-8     | [-1, 128, 50, 50]  | 0       |
| Conv2d-9     | [-1, 256, 25, 25]  | 295,168 |
| Conv2d-10    | [-1, 256, 25, 25]  | 0       |
| Conv2d-11    | [-1, 128, 50, 50]  | 295,040 |
| Conv2d-12    | [-1, 128, 50, 50]  | 0       |
| Conv2d-13    | [-1, 128, 50, 50]  | 295,040 |
| Conv2d-14    | [-1, 128, 50, 50]  | 0       |
| Conv2d-15    | [-1, 64, 100, 100] | 73,792  |
| Conv2d-16    | [-1, 64, 100, 100] | 0       |
| Conv2d-17    | [-1, 64, 100, 100] | 73,792  |
| Conv2d-18    | [-1, 64, 100, 100] | 0       |
| Conv2d-19    | [-1, 32, 200, 200] | 18,464  |
| Conv2d-20    | [-1, 32, 200, 200] | 0       |
| Conv2d-21    | [-1, 32, 200, 200] | 18,464  |
| Conv2d-22    | [-1, 32, 200, 200] | 0       |
| Conv2d-23    | [-1, 32, 400, 400] | 9,248   |
| Conv2d-24    | [-1, 32, 400, 400] | 0       |
| Conv2d-25    | [-1, 32, 400, 400] | 21,632  |
| Conv2d-26    | [-1, 32, 400, 400] | 0       |
| Conv2d-27    | [-1, 3, 400, 400]  | 99      |
| Conv2d-28    | [-1, 3, 400, 400]  | 0       |

Total params: 1,205,539
Trainable params: 1,205,539
Non-trainable params: 0

Input size (MB): 1.83
Forward/backward pass size (MB): 346.68
Params size (MB): 4.60
Estimated Total Size (MB): 353.11
### Table 4: Decoder architecture.

| Layer (type) | Output Shape | Param # |
|--------------|--------------|---------|
| Conv2d-1     | [-1, 32, 400, 400] | 3,200   |
| Conv2d-1     | [-1, 16, 393, 393] | 3,088   |
| Conv2DCNN-2  | [-1, 16, 393, 393] | 0       |
| Conv2d-3     | [-1, 16, 386, 386] | 16,400  |
| Conv2DCNN-4  | [-1, 16, 386, 386] | 0       |
| Conv2d-5     | [-1, 16, 379, 379] | 16,400  |
| Conv2DCNN-6  | [-1, 16, 379, 379] | 0       |
| Conv2d-7     | [-1, 16, 372, 372] | 16,400  |
| Conv2DCNN-8  | [-1, 16, 372, 372] | 0       |
| Conv2d-9     | [-1, 16, 365, 365] | 16,400  |
| Conv2DCNN-10 | [-1, 16, 365, 365] | 0       |
| Conv2d-11    | [-1, 16, 358, 358] | 16,400  |
| Conv2DCNN-12 | [-1, 16, 358, 358] | 0       |
| Conv2d-13    | [-1, 16, 351, 351] | 16,400  |
| Conv2DCNN-14 | [-1, 16, 351, 351] | 0       |
| Linear-15    | [-1, 123201, 8]   | 136     |
| Dense-CNN-16 | [-1, 123201, 8]   | 0       |
| Linear-17    | [-1, 123201, 2]   | 18      |
| Dense-CNN-18 | [-1, 123201, 2]   | 0       |

Total params: 101,642
Trainable params: 101,642
Non-trainable params: 0

Input size (MB): 1.83
Forward/backward pass size (MB): 255.63
Params size (MB): 0.39
Estimated Total Size (MB): 257.85

### Table 5: StegaPosDetector architecture.

| Layer (type) | Output Shape | Param # |
|--------------|--------------|---------|
| Conv2d-1     | [-1, 6, 347, 347] | 456     |
| MaxPool2d-2  | [-1, 6, 173, 173] | 0       |
| Conv2d-3     | [-1, 6, 169, 169] | 906     |
| MaxPool2d-4  | [-1, 6, 84, 84]  | 0       |
| Conv2d-5     | [-1, 6, 80, 80]  | 906     |
| MaxPool2d-6  | [-1, 6, 40, 40]  | 0       |
| Conv2d-7     | [-1, 6, 36, 36]  | 906     |
| MaxPool2d-8  | [-1, 6, 18, 18]  | 0       |
| Conv2d-9     | [-1, 12, 14, 14] | 1,812   |
| MaxPool2d-10 | [-1, 12, 7, 7]   | 0       |
| Linear-11    | [-1, 120]       | 70,680  |
| Linear-12    | [-1, 16]        | 1,936   |
| Linear-13    | [-1, 1]         | 17      |

Total params: 77,619
Trainable params: 77,619
Non-trainable params: 0

Input size (MB): 1.41
Forward/backward pass size (MB): 8.98
Params size (MB): 0.30
Estimated Total Size (MB): 10.68
Table 6: SpliceSegmentationNetwork architecture.

| Layer (type)     | Output Shape      | Param #  |
|------------------|-------------------|----------|
| Conv2d-1         | [-1, 32, 400, 400] | 896      |
| Conv2D-Det-2     | [-1, 32, 400, 400] | 0        |
| Conv2d-3         | [-1, 32, 200, 200] | 9,248    |
| Conv2D-Det-4     | [-1, 32, 200, 200] | 0        |
| Conv2d-5         | [-1, 64, 100, 100] | 18,496   |
| BatchNorm2d-6    | [-1, 64, 100, 100] | 128      |
| Conv2D-Det-7     | [-1, 64, 100, 100] | 0        |
| Conv2d-8         | [-1, 128, 50, 50]  | 73,856   |
| BatchNorm2d-9    | [-1, 128, 50, 50]  | 256      |
| Conv2D-Det-10    | [-1, 128, 50, 50]  | 0        |
| Conv2d-11        | [-1, 256, 25, 25]  | 295,168  |
| Conv2D-Det-12    | [-1, 256, 25, 25]  | 0        |
| Conv2d-13        | [-1, 128, 50, 50]  | 295,040  |
| Conv2D-Det-14    | [-1, 128, 50, 50]  | 0        |
| Conv2d-15        | [-1, 128, 50, 50]  | 295,040  |
| BatchNorm2d-16   | [-1, 128, 50, 50]  | 256      |
| Conv2D-Det-17    | [-1, 128, 50, 50]  | 0        |
| Conv2d-18        | [-1, 64, 100, 100] | 73,792   |
| Conv2D-Det-19    | [-1, 64, 100, 100] | 0        |
| Conv2d-20        | [-1, 64, 100, 100] | 73,792   |
| BatchNorm2d-21   | [-1, 64, 100, 100] | 128      |
| Conv2D-Det-22    | [-1, 64, 100, 100] | 0        |
| Conv2d-23        | [-1, 32, 200, 200] | 18,464   |
| Conv2d-24        | [-1, 32, 200, 200] | 0        |
| Conv2d-25        | [-1, 32, 200, 200] | 18,464   |
| Conv2D-Det-26    | [-1, 32, 200, 200] | 0        |
| Conv2d-27        | [-1, 32, 400, 400] | 9,248    |
| Conv2D-Det-28    | [-1, 32, 400, 400] | 0        |
| Conv2d-29        | [-1, 32, 400, 400] | 19,328   |
| Conv2D-Det-30    | [-1, 32, 400, 400] | 0        |
| Conv2d-31        | [-1, 1, 400, 400]  | 33       |
| Conv2D-Det-32    | [-1, 1, 400, 400]  | 0        |

Total params: 1,201,633
Trainable params: 1,201,633
Non-trainable params: 0

Input size (MB): 1.83
Forward/backward pass size (MB): 356.45
Params size (MB): 4.58
Estimated Total Size (MB): 362.86